Lecture Notes in Computer Science 4042

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

David Bell   Jun Hong (Eds.)

Flexible and Efficient Information Handling

23rd British National Conference on Databases, BNCOD 23
Belfast, Northern Ireland, UK, July 18-20, 2006
Proceedings

Springer

Volume Editors

David Bell
Jun Hong
Queen’s University Belfast
School of Electronics, Electrical Engineering and Computer Science
Belfast BT7 1NN, UK
E-mail:{da.bell, j.hong}@qub.ac.uk

Library of Congress Control Number: 2006928025

CR Subject Classification (1998): H.2, H.3, H.4

LNCS Sublibrary: SL 3 – Information Systems and Application, incl. Internet/Web and HCI

ISSN 0302-9743
ISBN-10 3-540-35969-9 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-35969-2 Springer Berlin Heidelberg New York

Springer is a part of Springer Science+Business Media

springer.com

Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11788911 06/3142 5 4 3 2 1 0

Preface

Since 1980, BNCOD conferences have provided an opportunity for researchers to present and discuss advances in research on database systems and in the deployment of database technology.

Contributions have been made to the plotting of new directions in the development of the technology and in the exploration of opportunities and challenges.

The Programme Committee of BNCOD 2006 sought to continue this pattern, and this year we chose to place special emphasis on "flexibility and performance." The idea is to look at challenges and developments from these two complementary and sometimes competitive directions. In practice, there is often a 'tightrope' to be walked between the two — keeping a balance is clearly important. However, this is sometimes overlooked, and sometimes a focus on one side or the other cannot be avoided.

Authors from 16 countries contributed a total of 58 submissions to the members of the Programme Committee. Each submission was evaluated by three members on the basis of originality, technical quality, presentation and relevance to BNCOD. As a result we have selected 12 full, 6 short and 13 poster papers which appear in this published proceedings volume along with the invited papers.

Some of the papers considered exclusively performance-related issues, and others were focussed on the flexibility side. A few sought to address the balance directly. The papers were organized into six presentation sessions and a poster session. We summarize the papers by session below.

The first session was on Data Modelling and Architectures and Transaction Management. These papers address both performance issues and flexibility issues. Harith Al-Jumaily, César de Pablo, Dolores Cuadra and Paloma Martínez write on using UML sequence diagrams as termination analyzer for trigger-based executing. They describe a visualization tool for representing and verifying trigger execution by using UML sequence diagrams. Their tool is added to Rational Rose and it simulates the execution sequence of a set of triggers when a DML operation is produced. It uses the SQL standard to express the semantics and execution. Victor Gonzalez-Castro, Lachlan MacKinnon and David Marwick authored a paper on an experimental consideration of the use of the Transrelational Model for Data Warehousing. They present an implementation of the Transrelational^{TM} model, based on the public domain definition provided by C.J. Date, which they believe to be the first reported instantiation of the model. They also present the results of tests where the performance is compared against a traditional N-ary Relational implementation. The topic of the paper by Stefan Böttcher, Le Gruenwald and Sebastian Obermeier is reducing sub-transaction aborts and blocking time. They introduce an extension to existing atomic commit protocols. It decreases the blocked-out time for a resource manager involved in a web service. It also reduces the number of sub-transaction aborts that arise due to message loss or conflicting concurrent transactions.

The second session, entitled Data Integration and Interoperability and Information Retrieval looked primarily at performance issues. Fahad Al-Wasil, Nick Fiddian and Alex Gray address the problem of query translation for distributed heterogeneous structured and semi-structured databases. They target both distributed heterogeneous structured data in relational databases and semi-structured data held in remote or local XML documents with no referenced DTD or XML schema. Mappings between the master view and the participating source schemas are defined in an XML Metadata Knowledge Base (XMKB). The paper by Shengli Wu and Sally McClean is about information retrieval evaluation with partial relevance judgment. Their investigation shows that when only partial relevance judgment is available, mean average precision suffers from several drawbacks. Their experiments suggest that average precision over all documents would be a good system measure. The paper by Alasdair Gray, Werner Nutt and Howard Williams studies sources of incompleteness in grid publishing. They identify different types of incompleteness in the context of RGMA with a view to finding solutions to some of these.

The third session on Query Processing and Optimization was primarily about performance. The first paper is by Guoqiang Zhan, Zude Li, Xiaojun Ye and Jianmin Wang. It looks at privacy preservation and protection by extending generalized partial indices. They propose an efficient Privacy Aware Partial Index mechanism based on the concept of purposes and the theory of partial indexing. All purposes are independent from each other, and they are organized in a flattened purpose tree (FPT) which can be updated. They extend existing query optimization and execution to enforce privacy policies, and report experiments demonstrating the feasibility and efficiency of the mechanism. The paper by Thomas Neumann, Sven Helmer and Guido Moerkotte is on the optimal ordering of maps, selections and joins under factorization. They show that if factorization is ignored during optimization, plans can be very far off the best possible. They introduce optimization strategies that produce optimal left-deep and bushy plans when factorization is taken into account, and their experiments show that factorization is critical in generating optimal plans, and that considering it does not incur serious performance penalties. Yunjun Gao, Gencai Chen, Ling Chen and Chun Chen write on an I/O optimal and scalable skyline query algorithm. They present an improved skyline computation algorithm based on best-first nearest neighbor search. It determines the optimal I/O and is economical with memory space. Their experimental evaluations show that their method can outperform conventional branch and bound skyline in both scalability and efficiency, especially in low dimensions.

The fourth session was on Data Mining. Papers in this session and the following two sessions addressed both performance and flexibility. Hui Wang presents a novel generic clustering method based on spatial operations. He describes an approach based on an extension of database relation — hyperrelations, which are sets of hypertuples, which are vectors of sets. He shows that hyperrelations can be exploited to develop a new method for clustering both numerical and categorical data. No parameters are needed from users. Initial experiments with artificial and real-world data show this novel approach is promising. Yuhong Guo, Yuhai Tong, Shiwei Tang and Dongqing Yang introduce an FP-tree-based method for inverse frequent set mining. They propose a tree-based method for the NP-complete problem of such mining. It saves on computational costs and provides a good heuristic search strategy to rapidly find the tree and the set of compatible databases.

The fifth session was also Data Mining. Ben Wang and John Gan present SC-Tree: an efficient structure for high-dimensional data indexing. To avoid inherent disadvantages veiled in the M-tree and its variants, which prevent them from further improvement of indexing and query efficiency, the paper proposes a sorted clue tree (SC-tree), which essentially modifies the nodes, entries, indexing algorithm, and query algorithm of the M-tree but retains its advantages. Experimental results and complexity analyses show that the SC-tree is much more efficient than the M-tree with respect to the query time and indexing time without sacrificing query accuracy. Ping Luo, Kevin Lü, Qing He and Zhongzhi Shi use a heterogeneous computing approach to solve the computing-intensive problem in data mining. This approach requires an effective and efficient scheduling framework to orchestrate different computing resources used for multiple competitive jobs in the data mining workflow. They introduce a dynamic DAG algorithm for scheduling data mining jobs based on an approximate estimation of their execution time. Carson Leung, Ruppa Thulasiram and Dmitri Bondarenko propose a parallel system for detecting outliers from financial time series. In their paper, they develop an efficient parallel system to detect noise in time series security prices. The system consists of a data mining algorithm and a statistical algorithm, which run in parallel to efficiently detect noise.

The final session was on Data Warehousing and Decision-support Systems and Data Streaming. Carson Leung and Wookey Lee write about efficient update of data warehouse views with generalized referential integrity differential files, (GRIDFs). These self-maintain the views modelled in any schema, in a way that avoids accessing the underlying source data. The authors present experimental results to show that the method leads to efficient update of data warehouse views. Ladjel Bellatreche, Kamel Boukhalfa and Hassan Abdalla introduce a combination of genetic and simulated annealing algorithms for physical data warehouse design. They formalize the horizontal fragmentation selection problem as an optimization problem with a maintenance constraint representing the number of fragments that the data warehouse administrator may manage. They present SAGA, a hybrid method combining a genetic and a simulated annealing algorithm to solve the problem. Several experiments are presented to validate the algorithms. Sharma Chakravarthy and Vamshi Pajjuri discuss trust scheduling strategies and their evaluation in a data stream management system. They introduce a path capacity scheduling strategy to minimize tuple latency by scheduling operator paths with maximum processing capacity. They also discuss a segment-scheduling strategy to minimize the total memory needed by scheduling operator segments with maximum memory release capacity, and another simplified strategy. They simulate real-time streams in their experiments to validate the analytical conclusions. Altaf Gilani, Satyajeet Sonune, Balakumar Kendai and Sharma Chakravarthy present a paper on the anatomy of a stream processing system. Their paper describes the architecture of MavStream. In it the user can present or query GUI, which is instantiated, scheduled, and executed by the MavStream server. Experimental results are presented to demonstrate the utility of the system and the effect of different scheduling strategies and buffer sizes on the performance and output.

The poster session contained 13 papers on various topics such as Query Processing and Optimization, Data Integration and Interoperability, Data Security and Privacy, XML, Data Warehousing, Semantic Web and Ontologies, Data Modelling and Architectures, and Spatial, Temporal and Scientific Data.

We want to thank the invited speakers, Georg Gottlob and Witold Litwin, for their insight into the state of the art and their visions for the future. Their contributions are in keeping with the conference theme. If you look at their short CVs and projects below you will see that they reflect the two aspects of our conference theme well.

Georg Gottlob is a Professor of Computing Science at Oxford University. His research interests are database theory (in particular, query languages), Web information processing, graph algorithms, constraint satisfaction problems, nonmonotonic reasoning, finite model theory, and computational complexity. On the more applied side, he supervises a number of industry projects dealing with Web information extraction and related topics. From 1989 to 1996 he directed the industry-funded Christian Doppler Laboratory for Expert Systems. He is a co-founder of the Lixto Corporation (www.lixto.com).

Prof. Gottlob was awarded his Engineer and Ph.D. degrees in Computer Science from TU Vienna, Austria in 1979 and 1981, respectively. Before moving to Oxford, he was a Professor of Computer Science at the Vienna Technical University (TU Vienna) since 1988. He continues to be affiliated with TU Vienna as a part-time Adjunct Professor. From 1981 to 1987, he was affiliated with the Italian National Research Council in Genoa, Italy, and with the Politecnico di Milano, Italy. He was a research scholar at Stanford University, an invited McKay Professor at UC Berkeley in 1999, and a Visiting Professor of the University Paris of VII in 2002.

He received the Wittgenstein Award from the Austrian National Science Fund, a Royal Society Wolfson Research Merit Award, and was elected a member of the Austrian Academy of Sciences and a member of the German Academy of Sciences Leopoldina.

He chaired the Programme Committees of ACM PODS 2000 and of IJCAI 2003, and he has been an invited speaker at many international conferences.

His talk at BNCOD was on the Lixto Project, and with a number of his colleagues he prepared a paper on the Lixto Project - Exploring New Frontiers of Web Data Extraction.

The Lixto project is an ongoing research effort in the area of Web data extraction. The project originally started out with the idea to develop a logic-based extraction language and a tool to visually define extraction programs from sample

Conference Committees

Programme Committee

David Bell (Chair)	Queen's University Belfast
Jun Hong (Co-chair)	Queen's University Belfast
Abbes Amira	Queen's University Belfast
Richard Connor	University of Strathclyde
Richard Cooper	University of Glasgow
Pat Corr	Queen's University Belfast
Barry Eaglestone	University of Sheffield
Suzanne Embury	University of Manchester
Alvaro Fernandes	University of Manchester
Jane Grimson	Trinity College, Dublin
Alex Gray	Cardiff University
Jiwen Guan	Queen's University Belfast
Mike Jackson	University of Central England
Anne James	Coventry University
Keith Jeffery	CLRC Rutherford Appleton
Jessie Kennedy	Napier University
Brian Lings	University of Exeter
Weiru Liu	Queen's University Belfast
Michael Madden	National University of Ireland, Galway
Nigel Martin	Birkbeck College, University of London
Sally McClean	University of Ulster
Lachlan MacKinnon	University of Abertay Dundee
Peter Milligan	Queen's University Belfast
Ken Moody	University of Cambridge
Fionn Murtagh	Royal Holloway, University of London
David Nelson	University of Sunderland
Werner Nutt	Free University of Bozen-Bolzano
Norman Paton	University of Manchester
Alexandra Poulovassilis	Birkbeck College, University of London
Paul Sage	Queen's University Belfast
Jianhua Shao	Cardiff University
Hui Wang	University of Ulster
Howard Williams	Heriot-Watt University

Web pages. However, the scope of the project has been extended over time. New issues such as employing learning algorithms for the definition of extraction programs, automatically extracting data from Web pages featuring a table-centric visual appearance, and extracting from alternative document formats such as PDF are being investigated currently.

Witold Litwin is the Director of Centre d'Etudes et de Recherches en Informatique Appliqué (CERIA) at University of Paris 9 and he has been Professor of Computer Science at the University of Paris 9 (Dauphine) since 1990. He was made an ACM Fellow in 2002 for pioneering research in dynamic storage structures, scalable distributed file structures and multidatabases. His research areas are in multidatabase systems and data structures, and in scalable distributed data structures. The techniques he proposed, including the linear hash data structure invented in the 1980s, are among the most renowned contributions to these domains. They are present in major database systems and in other products, including Netscape and Microsoft popular offerings.

Professor Litwin has worked in several universities and in industry. In the USA, he taught at UC Berkeley in 1992-94, at Santa Clara University in 1991 and Stanford University in 1990-91. He was a visiting scientist at, for example, IBM Almaden Research Center, in 1997 and 1998, and part time at Hewlett Packard Palo Alto Laboratories, between 1990 and 1994, as well as at the University of Maryland in 1989. Between 1980 and 1989 he was Research Director at Institut National de Recherche en Informatique et Automatique (INRIA, France), and Expert for ISO Committee on Open Systems. He has written over 150 research papers, edited or contributed to 11 books, and served on the Program Committees of over 50 international database conferences.

His talk was on the SD-SQL Server which incrementally repartitions growing tables on more and more linked SQL Server nodes. With his colleagues Soror Soror and Thomas Schwarz, he prepared a paper for this volume, which introduces SD-SQL as the first DBMS to avoid the cumbersome manual repartitioning, characteristic of current DBMS technology. A table expands by splits of its overflowing segments, dynamically triggered by inserts. The partitioning is invisible to users, hidden by scalable distributed updatable partitioned views. With the comfort of a single node SQL Server user, the SD-SQL Server user manages larger tables or processes the queries faster through dynamic parallelism. The paper presents the architecture of the system, its implementation and the performance analysis. The analysis shows that the overhead of their scalable distributed table management should be typically negligible.

We want to thank all the authors for providing us with these high-quality papers. We are also grateful to the members of the Programme Committee and the other referees for their professionalism and dedication in the process of judging the contributions of papers.

April 2006

David Bell (Programme Chair)
Jun Hong (Programme Co-chair)
BNCOD 2006

Steering Committee

Brian Lings (Chair)	University of Exeter
Barry Eaglestone	University of Sheffield
Alex Gray	Cardiff University
Anne James	Coventry University
Keith Jeffery	CLRC Rutherford Appleton
Roger Johnson	Birbeck College, University of London
Lachlan MacKinnon	University of Abertay Dundee
Alexandra Poulovassilis	Birkbeck College, University of London

Organizing Committee

Peter Milligan (Chair)	Queen's University Belfast
Abbes Amira (Local Arrangement Chair)	Queen's University Belfast
David Bell	Queen's University Belfast
Pat Corr (Sponsorship Chair)	Queen's University Belfast
Jun Hong	Queen's University Belfast
Weiru Liu (PhD Forum Chair)	Queen's University Belfast
Paul Sage (Workshop Chair)	Queen's University Belfast
Colette Tipping (Conference Secretary)	Queen's University Belfast

Additional Referees

Qingyuan Bai
Khalid Belhajjame
Rob Hidderley
Steven Lynden
Rudy Prabowa
Karen Renaud

Table of Contents

Invited Papers

Data Modelling and Architectures and Transaction Management

Data Integration and Interoperability and Information Retrieval

Query Processing and Optimization

Data Mining

Data Warehousing and Decision-Support Systems

Data Streaming

Poster Papers

The Lixto Project: Exploring New Frontiers of Web Data Extraction*

Julien Carme[1], Michal Ceresna[1], Oliver Frölich[1], Georg Gottlob[2],
Tamir Hassan[1], Marcus Herzog[1], Wolfgang Holzinger[1], and Bernhard Krüpl[1]

[1] Vienna University of Technology, Database and Artificial Intelligence Group,
Favoritenstraße 9-11, A-1040 Wien, Austria
[2] Oxford University Computing Laboratory, Wolfson Building,
Parks Road, Oxford, OX1 3QD, United Kingdom

Abstract. The Lixto project is an ongoing research effort in the area of Web data extraction. Whereas the project originally started out with the idea to develop a logic-based extraction language and a tool to visually define extraction programs from sample Web pages, the scope of the project has been extended over time. Today, new issues such as employing learning algorithms for the definition of extraction programs, automatically extracting data from Web pages featuring a table-centric visual appearance, and extracting from alternative document formats such as PDF are being investigated.

1 Introduction

Web data extraction is an active research field, dealing with the subject of extracting structured data from semi-structured Web sites. In order to extract structured data from a Web page, we need to generate an appropriate extraction program, called a wrapper. We can distinguish two main methodological approaches for constructing such extraction programs: the supervised and the unsupervised approach. In the supervised approach, an operator needs to define the wrapper program either by coding the program manually or by using a visual development environment to generate the program code. In the unsupervised approach, the system generates wrappers automatically from a given set of heuristics and domain knowledge.

Whilst the unsupervised approach is very scalable in terms of the number of input Web pages that can be processed, this comes at the cost of precision. Due to the quality control inherent in the supervised approach, this approach is best suited where highly accurate data with an almost zero failure rate is required.

The Lixto Visual Wrapper introduced in [3] employs a supervised approach to generate wrapper programs from a given sample Web page. The operator highlights relevant data items on the Web page and the system generates logic-based extraction rules to extract these data items from the sample Web page

* This work is funded in part by the Austrian Federal Ministry for Transport, Innovation and Technology under the FIT-IT Semantic Systems program.

D. Bell and J. Hong (Eds.): BNCOD 2006, LNCS 4042, pp. 1–15, 2006.

and other web pages with a similar structure. These rules utilize the document structure and specific attributes of the user-selected data items to locate other relevant data instances to be retrieved. The logical foundation on Web data extraction, and on the complexity and expressive power on data extraction using the Lixto approach were studied in [10, 11, 9].

In the first part of this paper, we will report on a recent addition within this Web data extraction framework to employ a learning strategy to select an optimal set of attributes for identifying the data items on a Web page. In the next part, we will discuss our approach to unsupervised data extraction from Web pages. Another topic that we are currently investigating is the extraction from non-HTML formats such as PDF. We will use a use case, i.e., extracting data in the domain of digital cameras, to illustrate the techniques that we have developed in these various fields of Web data extraction. Furthermore, we will report on industrial applications of Web data extraction and highlight the benefits that these applications can generate in a real-world setting.

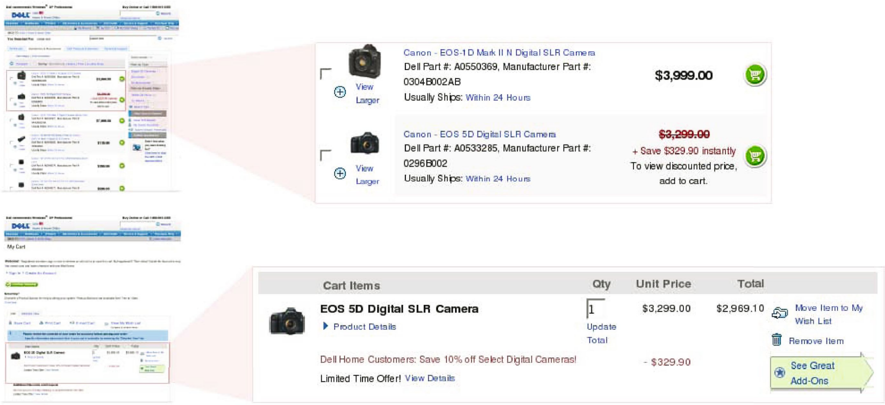
2 Supervised Wrapper Generation

We will use the digital camera domain to illustrate typical use cases for Web data extraction. Let us imagine the application of monitoring camera prices. Assume that we have several competitors and we want to *continuously* monitor their websites for price development of the listed goods. We store the prices collected from their websites into a local database. We can then use business intelligence tools to analyse the aggregated prices, and this will allow us to react to market changes with more effective pricing strategies and advertisement campaigns. Figure 1 shows a sample of Web pages from the Dell online shop that serves as an example for similar online shops from which price information could be extracted.

This use case favors the usage of a supervised approach due to the following requirements:

- if the wrapping algorithm does not work on the given website, we cannot go to another site and collect the prices there;
- accuracy (precision/recall) must be high, because business decisions rely on the gathered data, and the solution therefore must guarantee the quality of the results obtained with the wrapping service;
- deep Web navigation is required before the actual data can be wrapped, e.g., it is required to fill Web forms or handle JavaScript (AJAX) execution.

In the following example, we need to collect prices of digital cameras from the Web site `www.dell.com`. To obtain prices for some of the cameras we have to navigate to the detail pages of the shopping cart. Informally, the problem we are trying to solve is: given a Web site (or a set of Web pages) as input and a user knowing what should be extracted from the Web site, we need to construct a wrapper to extract exactly the required information items.



Canon - EOS-1D Mark II N Digital SLR Camera
Dell Part #: A0550369, Manufacturer Part #: 0304B002AB
Usually Ships: Within 24 Hours
$3,999.00

Canon - EOS 5D Digital SLR Camera
Dell Part #: A0533285, Manufacturer Part #: 0296B002
Usually Ships: Within 24 Hours
~~$3,299.00~~
+ Save $329.90 instantly
To view discounted price, add to cart.

Cart Items	Qty	Unit Price	Total	
EOS 5D Digital SLR Camera Product Details	1 Update Total	$3,299.00	$2,969.10	Move Item to My Wish List Remove Item
Dell Home Customers: Save 10% off Select Digital Cameras! Limited Time Offer! View Details		- $329.90		See Great Add-Ons

Fig. 1. Example of an information extraction scenario where a combination of wrapping and focused crawling is required

2.1 Wrapper Structure

The navigation sequence, together with the wrapper, is captured in a navigation language that we have described in one of our previous publications [4]. The navigation sequence is created by an algorithm that records interaction of the user with the Web browser and stores all mouse and key events that occur. An example of the navigation sequence with an embedded wrapper for the Dell shop scenario is shown in Algorithm 1.

The wrapping itself is embedded in the **extract** function. This function operates on DOM trees – the standard tree model of Web pages in modern Web browsers. It receives as input a list of DOM nodes and a definition of the extraction (called *pattern*), and outputs another list of DOM nodes. For example, in Algorithm 1, one of the extract functions receives a *camera* node and $pattern\_price_1$ as input and returns a *price* node as output.

2.2 Learning Patterns

Patterns describe one specific extraction task, for example, the extraction of the $pattern\_camera$ or $pattern\_price_1$ from the wrapper in Algorithm 1. To define each pattern, we use a boolean combination of *basic conditions*, denoted as $C(p, n)$.

The basic condition $C(p, n)$ is a function that tests the correct position of the target DOM node n with respect to the context DOM node p, and local properties of the DOM node n. Examples of local properties of n are the presence of a given attribute with some value, or the existence of another sibling node s. Formally, $C(p, n)$ is a triple $(path_1, path_2, test)$ such that

$$C(p, n) \Leftrightarrow path_1(p, n) \wedge \exists s \; path_2(n, s) \wedge test(s)$$

Algorithm 1. Navigation sequence and wrapper for the Dell online shop

```
# load main page
load('http://www1.us.dell.com/content/...')
# wrap all camera entries
$cameras$ = extract(#$doc$, $pattern\_camera$)
for $c$ in $cameras$ do
  $c.name$ = extract($c$, $pattern\_name$)
  $c.image$ = extract($c$, $pattern\_image$)
  $c.price$ = extract($c$, $pattern\_price_1$)
  if $c.price == None$ then
    # load next page
    $cartImg$ = extract($c$, $pattern\_cartImg$)
    sendClick($cartImg$)
    # wrap price from cart
    $c.price$ = extract(#$doc$, $pattern\_price_2$)
    # return to main page
    goBack()
    goBack()
  end if
end for
```

The expression $path_1$ and $path_2$ are called XPath [19, 12] expressions. As we have shown earlier [7], there are boundaries of query-based learnability for XPath expressions. Therefore, here we use XPath expressions of simplified for, which contain only the child (/) and ancestor (//) steps, index test ([i]) and no wildcards (*) in tag names.

For example, for the wrapper in Algorithm 1, *pattern_camera* is defined as $(//table/tr, ., true)$, $pattern\_price_1$ as $(./td/b, ., @bgcolor! = `red`)$ and $pattern\_price_2$ is defined as $(//table/tr/td[3], ., @bgcolor! = `red`) \wedge (., ./parent :: */parent :: */tr/td[3], text() == `Unit\ Price`)$.

When building our wrapper, no annotated data (Web page) exists in advance. Therefore, we interact with the user to query the required annotations, but with a goal to minimize the number of requested inputs. The patterns are then induced from the positive and negative examples received from the user during the interaction outline in Figure 2.

In each iteration cycle of the interaction, an exhaustive set of basic conditions is generated from the set of examples that have so far been received. Then, an optimal combination of some of these conditions is learned using a DNF-learning algorithm. Unfortunately, the number of variables and therefore basic conditions in the target formula is not bounded. This implies that the Vapnik-Chervonenkis dimension of the hypothesis space of all boolean combinations of the basic conditions is not bounded, and the problem is therefore not PAC learnable [5].

Also, to the best of our knowledge, PAC- and query-based learnability of the DNF itself is not known. Therefore, for our implementation to remain tractable, we limit ourselves to only learning the k-DNF. The learning algorithm works

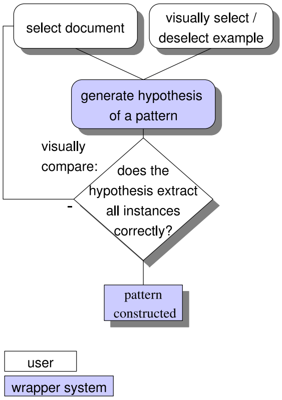

select document
visually select / deselect example
generate hypothesis of a pattern
visually compare:
does the hypothesis extract all instances correctly?
-
pattern constructed
user
wrapper system

Fig. 2. Interaction of a user with the learning algorithm

in the following way: first all conjunctions of size smaller than k are generated, and then the minimum disjunction of these conjunctions, still consistent with the examples, is searched. Note that although finding the minimal disjunction is NP-complete, efficient heuristics are known.

2.3 Results

The main advantage of the learning algorithm is that it allows non-experienced users to build wrappers in an easy to understand way—point and select (or deselect) the wanted (or unwanted) instances with the mouse.

Our experiments show that the interactivity of the learning algorithm reduces the number of required examples to define a pattern. Usually, only 3–5 examples are required to build a pattern. This is due to the fact that redundant examples are not part of the annotation, e.g. receiving an image as a redundant positive example, if our hypothesis already extracts images, or receiving a hyperlink as an unrelated negative example if the pattern should extract only images.

Another advantage of the approach using boolean combination of basic conditions is that it generates wrappers that are understandable by human users. This allows the user to check or manually alter the learned pattern. The easiness of understanding is in contrast with other approaches used in information extraction, such as Hidden Markov Models or Conditional Random Fields, which learn a vector of real-valued parameters that is not intuitive for humans to understand.

3 Unsupervised Wrapper Generation

In contrast to the supervised approach, the unsupervised approach is feasible if the goal is to extract from a huge number of Web resources without the need to be able to successfully process every single Web resource that potentially holds relevant data items. For example, in order to harvest domain knowledge about cameras, it is sufficient to process a fraction of all potential descriptions of digital camera models on the Web in order to generate a knowledge base on camera models describing all available camera models and their features.

3.1 Resource Discovery and Focused Crawling

In the unsupervised, approach the system needs to navigate and extract data from Web pages fully automatically. The basic idea here is to mimic the behaviour of an human expert. A person, given the task of collecting addresses of pages containing useable tabular information about digital cameras, will typically apply the following strategy: First, the person uses an internet search engine to find websites about cameras. To do this, the user will pose queries to the search engine that contain terms likely to appear on such a website. The user must have knowledge about the digital camera domain, i.e., the specific language and technical terms used. From the search engine's results, the user then sorts out the relevant items and repeats the process, this time using a slightly different query. After some iterations, some websites will stand out as exceptionally valuable sources of information—appearing in the search engine results in each of the queries—thus indicating that they are not only relevant to the exact phrase that was submitted during a single query but are indeed relevant to the whole domain.

At this point, the user will deviate from the initial strategy of using a search engine and begin to investigate these interesting websites directly. The user knows that information on the Web comes in a clustered form: a website that talks about some cameras is likely to talk about all cameras, or will at least contain links to such sites. The user will start *browsing* websites, having basic prior knowledge of how navigation on a website works (using fold-out menus, hyperlinks, forms, etc). The user will then need to learn the particular idiosyncracies that are used by each website to organize information. The user already knows some of the relevant pages on the site from the search engine results; now he has to find the path through the website's navigational structure that leads to these pages. Once this path is found, slight variations of it (i.e., go one step back and try alternatives) will uncover a wealth of relevant pages.

Following this two-step strategy that a human expert would employ, we constructed a software implementation that works in two stages: In stage 1 we select a small sample S from a collection C of phrases pertinent to our domain at random. Very common words appearing in C are given a smaller probability of being selected. S is converted to a (conjunctive) query and submitted to a search engine; the top rated results are grouped by website and stored for further processing. This query process is repeated until no significant new information shows up, typically after several dozen iterations.

After this, we use the table extraction algorithm explained later in this section to iterate through all the pages found and determine which pages contain extractable tables, and are therefore relevant for further processing. Eventually, we obtain a list of relevant websites p_i, each associated with a set of relevant pages that we call templates T_i.

In stage 2 we apply a Web crawler to all relevant websites, starting with the website that contains the most templates and therefore looks the most promising. This crawler is focused on finding pages that match the templates T_i closely. For the matching algorithm we use a measure of *structural similarity* of pages as follows: A HTML page is reduced to its *skeleton code* by first removing all text nodes, tag attributes and closing tags. The remaining sequence of tags is then converted to a string by replacing each tag by a unique character. Once we have determined the skeleton codes s_1, s_2 of two pages we can measure their structural similarity $d(s_1, s_2)$ with an appropriate string distance function d; at the moment we use the Levenshtein distance [15]. The threshold distance d_T, above which we reject a page as being not sufficiently similar to our template pages, is determined by computing the pairwise distances d_{ij} among the T_i. We observed a normal distribution of these d_{ij} and use the 99% quantile of the observed distribution as the threshold d_T. Pages that fall below the threshold are called extraction candidate pages, or candidate pages for short.

Learning how to navigate a site in the same way as a human user turned out to be the most difficult part of the problem: to rely on structure and image pattern recognition—both of which are computationally expensive tasks—would not be feasible in a crawler that is expected to process thousands of pages in a short time. Instead, we turned to analysing the graph G spanned by the hyperlinks among the pages of a website. Our assumption is that the navigational pages have a special, central position in G that we can identify. After all, functional navigation should be available to the user at any time and can be used to reach even the remotest places on the website—the "trunk" of the website, so to speak. Therefore, a random surfer on the website would invariably visit those central pages more often—and we can use the PageRank algorithm [17] to identify these navigational pages.

The crawler keeps following outgoing links on pages with the highest page ranks until it hits upon one of the T_i or a page sufficiently similar to it. Once this happens, a new heuristic comes into play: *hubs*, pages that contain links to a large number of candidates, can be recognized. A page gets a *hub score* proportional to the number of extraction candidate pages it links to. The heuristic that decides which page to expand next is based on a weighted sum of both the page rank and the hub score of the page. This way, the crawler turns its attention from navigating the website to exploiting the nest of candidate pages it has stumbled upon.

3.2 Results

Figure 3 gives an indication of the crawler's performance when set to explore a typical digital camera review website. It shows the harvest rate; the ratio of candidate pages to visited pages, against the number of iterations. The crawler first begins by visiting pages leading out from the main navigation, then hits upon the first candidate at around iteration 50. It quickly focuses on the area of this result and stabilizes, turning out new candidate pages every four iterations. Due to irrelevant links present on the hub pages, the performance here never gets closer to 1.

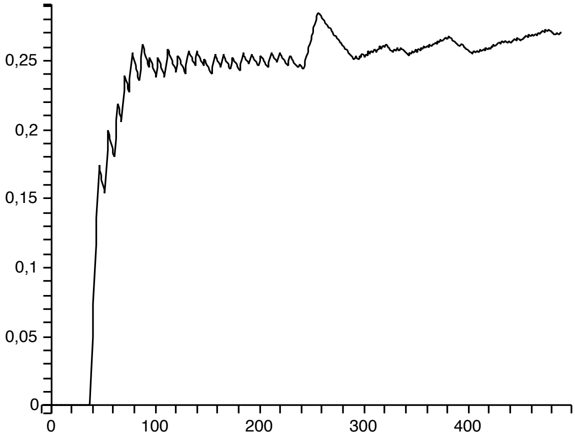


Fig. 3. Harvest rate over iteration

3.3 Automatic Table Extraction from Web Documents

There are situations where it is desirable to make the extraction process completely unsupervised. For example, many technical product descriptions on the Web are in the form of tables. A system that could locate, segment and analyse these tables automatically would therefore be of great value: it could assist the designer in the wrapper construction process by providing easier navigation within the document under consideration, or it could be used on its own to build a fully unsupervised extraction system to automatically extract data tables from a large amount of Web pages. We are currently implementing such a table location and analysis component.

What makes tables different from free text is that they are inherently concise. By extracting information from tables, we can avoid dealing with most of the complexity of natural language, since tables usually contain information in a condensed style. Thus, analysing the content with extraction ontologies [8] is more promising than in the case of free text.

Several articles deal with the problem of classifying HTML tables as genuine or non-genuine. This comes from the fact that the HTML <`table`> element is often used just for the purpose of implementing a specific page layout. It is therefore crucial for methods that analyse the source code of a Web page to identify only those genuine table elements that are not for layout purposes. By operating directly on the visual rendition, such a classification becomes obsolete.

Traditional wrappers operate on HTML input either in the form of a sequential character string or a pre-parsed document tree. With the Lixto Visual Wrapper, the wrapper is specified visually by interactively clicking on the rendition of a page to annotate relevant content. The Lixto software then determines the node in the pre-parsed document tree that best matches the selected region and generates appropriate extraction statements. For automatic table extraction, we decided not to use the HTML source code at all: if it is possible to visually define the relevant data area, all the data required to locate the extraction instance is clearly contained in the visual rendition. Going back to the document source code, whether pre-parsed or not, is therefore an unnecessary step. If the wrapper can be grounded on the same visual properties of a document that enable the user to mark the relevant parts of the page, it should also be more robust regarding future changes of the page.

3.4 Table Extraction Algorithm

We are currently implementing an automatic table location and analysis algorithm. This algorithm operates on the rendition of a Web page provided by a Web browser and thus avoids all the peculiarities and complications involved with the interpretation of HTML and associated CSS code. The goal of the algorithm is the identification of data-centric tables and the subsequent transformation of the data contained in the table into a structural form preserving the relationships between table cells. According to the literature [14], the steps involved are: table location, segmentation, functional analysis and structural analysis. So far, we have concentrated our work on a limited number of physical table models; essentially simple, unnested tables with the possibility for intermediate headings appearing as additional lines in the table.

Unlike most of the other implementations in the literature, our table algorithm works in a bottom-up fashion by starting from pixel positions of single words that have been determined with the help of the Web browser. These word bounding boxes are grouped into larger clusters of possible cells based on their adjacency, which is illustrated as step 1 in Figure 4. Since the pixel coordinates are derived from the Web browser layout engine, we can assume that there is no noise in the data and that there is a true adjacency of neighbouring cells with a pixel distance of zero.

As in the table definition given in [18], we do not consider line-art or other graphical properties of tables; we identify and segment a table just from the positions of its inherent word bounding boxes. Rather than looking for tables as a whole, we start by trying to identify possible table columns. A column candidate in our context is a collection of cells that, within a small tolerance, share a common coordinate on the horizontal axis. This can be either the left border, centre, or right border pixel coordinate of the cell to account for left aligned, centered, or right aligned columns. Figure 4 shows the column identification as step 2.

On the vertical axis, we allow up to one non-column cell between every two column candidate cells to account for the possible intermediate headings mentioned above; if this limit is exceeded, the column candidate is split into two column candidates. Then we investigate whether all the separating cells (in other words, the intermediate heading candidates) share a common coordinate. If they do, we have found a column candidate. Otherwise, we split our column candidates into column candidates that will be treated separately.

In the next stage, we try to find the best column candidate combination that could possibly form a data table. Here we follow a strategy that we call

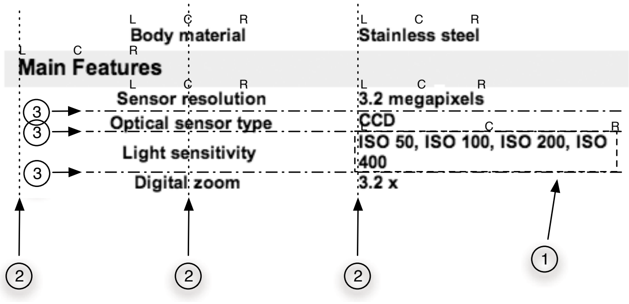


Fig. 4. Operation of the table extraction algorithm on part of a sample page

comb alignment of columns: we look for adjacent columns where we observe only 1:n or m:1 relationships between adjacent table cells. This means that we can handle cases where a cell in one column corresponds to several cells in the second column, usually representing a hierarchical relationship between these cells. Step 3 in Figure 4 illustrates the comb alignment between two columns. If we can establish such an alignment, we can derive a proper table segmentation.

In the final stage, an analysis of the segmented table is performed. Here we try to recover the relation of cells in the table or, put differently, the reading order of the table. With the information about intermediate headings and the direction of the comb alignment, we can already make a judgement about the functional role of the respective cells. The analysis is finally finished by assigning subject, predicate or object roles to the table cells based on cell neighbourhoods and on our knowledge of table models.

Because our unsupervised table extraction algorithm generates a large number of triples, its results are well suited for a statistical analysis aimed at leveraging the great redundancy of information on the Web. This is in stark contrast to other approaches that rely just on a few sources. The aggregation and integration of all these information fragments is the objective of another research effort in the Lixto context.

4 Wrapping from PDF Files

In today's Web, the vast amount of HTML data is complemented by a significant number of documents published in Adobe's Portable Document Format (PDF). In general, these documents are primarily intended for printing, and many business-critical documents fall into this category. Examples of such documents include financial reports, newsletters and price lists, such as our example in figure 5 (left) from the digital camera domain. Clearly, the ability to semi-automatically extract information from these documents proves to be extremely useful for a number of business applications.

The success of PDF can be attributed to its roots as a page-description language. Any document can be converted to PDF as easily as sending it to the printer, with the confidence that the formatting and layout will be preserved when it is viewed or printed across different computing platforms. This ease of publication has led to a lot of data on the Web being available only in PDF format, with no corresponding HTML alternative.

Unfortunately, this approach presents one major drawback: most PDFs have little or no explicit structural information, making automated machine processing and data extraction a difficult task. Although later versions of the PDF specification support the use of XML tags to denote logical elements, these are seldom found in business documents.

Our PDF extraction functionality within Lixto utilizes a variety of techniques from document understanding literature to attempt to rediscover the logical structure from the layout of the document. This structure can then be used in a similar way to the HTML parse tree to locate data instances for wrapping.

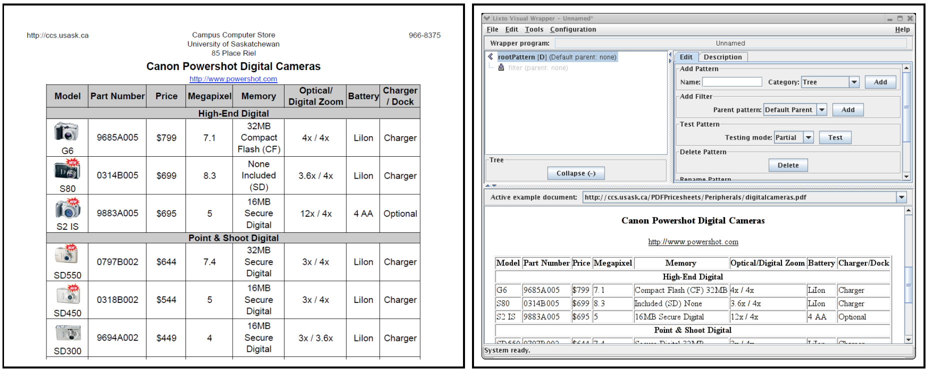
In this section we describe our recent advances in PDF wrapping within Lixto, and present an insight into our current work in this area, and what our future releases may offer.

4.1 The Wrapping Process

The PDF import filter within Lixto is automatically activated when the input document is detected as a PDF. Our algorithms detect structures on the page, such as columns, lists and tables, and represent them in XHTML, much like a web page. The wrapper designer is then able to interact with this representation in the same way as with a web page, as shown in the example in figure 5 (right).

The latest version of our PDF filter benefits from a several improvements to our document understanding algorithms, and can now produce good results even with relatively complex layouts. The remainder of this section details some of the techniques that we have used.

Document pre-processing: In general, the first step in understanding a document is to segment it into blocks that can be said to be *atomic*, i.e. to represent



http://ccs.usask.ca

Campus Computer Store
University of Saskatchewan
85 Place Riel

966-8375

Canon Powershot Digital Cameras

http://www.powershot.com

Model	Part Number	Price	Megapixel	Memory	Optical/ Digital Zoom	Battery	Charger / Dock
High-End Digital							
G6	9685A005	$799	7.1	32MB Compact Flash (CF)	4x / 4x	LiIon	Charger
S80	0314B005	$699	8.3	None Included (SD)	3.6x / 4x	LiIon	Charger
S2 IS	9883A005	$695	5	16MB Secure Digital	12x / 4x	4 AA	Optional
Point & Shoot Digital							
SD550	0797B002	$644	7.4	32MB Secure Digital	3x / 4x	LiIon	Charger
SD450	0318B002	$544	5	16MB Secure Digital	3x / 4x	LiIon	Charger
SD300	9694A002	$449	4	16MB Secure Digital	3x / 3.6x	LiIon	Charger

Wrapper program: Unnamed

rootPattern [D] (Default parent: none)
filter (parent: none)

Name:
Category: Tree

Parent pattern: Default Parent

Testing mode: Partial

Active example document: http://ccs.usask.ca/PDFPricesheets/Peripherals/digitalcameras.pdf

Canon Powershot Digital Cameras

http://www.powershot.com

Model	Part Number	Price	Megapixel	Memory	Optical/Digital Zoom	Battery	Charger/Dock
High-End Digital							
G6	9685A005	$799	7.1	Compact Flash (CF) 32MB	4x / 4x	LiIon	Charger
S80	0314B005	$699	8.3	Included (SD) None	3.6x / 4x	LiIon	Charger
S2 IS	9883A005	$695	5	16MB Secure Digital	12x / 4x	4 AA	Optional
Point & Shoot Digital							
SD550	0797B002	$644	7.4	Secure Digital 32MB	3x / 4x	LiIon	Charger

System ready.

Fig. 5. Example of wrapping from a PDF price list

one distinct logical entity in the document's structure. Many of the segmentation techniques in document understanding, such as those utilized in [2] and [1], have been developed by the OCR community, and take a scanned, binarized image of the page as input. Whilst we could make use of these techniques by rasterizing each page of the PDF, this process would throw away useful information, introduce noise, waste processing time and essentially take us backwards. Therefore, we choose to segment the page directly on the object data that is contained within the PDF.

A PDF file is little more than a collection of characters and graphic objects placed on a page. Referring again to our example in figure 5, we consider the title, the address of the store and the other single lines of text all to be distinct logical entities. Inside the table, each individual cell is a distinct logical entity. This definition gives us sufficient granularity for locating these data items later.

– **Line finding:** In a PDF file, text is stored in discrete blocks, usually with no more than 2–3 characters per block (although this can depend on the program used to generate the document). The first step is therefore to merge these text fragments into complete lines. Space characters are not always included in the original source, and therefore must be added to separate words if the distance between two neighbouring blocks is too large. Our algorithm examines the spacing between each character and, therefore, copes with a variety of different character spacings.
– **Clustering:** The next stage is to merge these lines into discrete blocks that are logically distinct. As text in a PDF can use a variety of different fonts, sizes and leadings, we make use of a *variable-threshold* clustering algorithm. This algorithm examines a variety of different possible groupings of paragraphs, and a consistency heuristic is used to determine the correct grouping from this set. Further heuristics are used to detect tabular structures and ensure that each cell is distinct.

Logical structure understanding: After page segmentation, the task is to identify higher-level logical relationships and detect substructures, such as lists and tables, within the page. Currently, we have a set of heuristics that detect multiple layers of headings, and cope with multiple column layouts. Our table understanding algorithm converts tabular structures to <`table`> elements in our XHTML representation, and can detect spanning columns or rows.

We are now investigating the use of an ontological framework to abstract these rules and heuristics from our code. This will enable the rules in future releases to be more easily adapted, and for domain-specific rules to be modularly "plugged in".

4.2 Future Developments

Currently, wrapping from PDF is a two-step process. First, the PDF document is imported, and the user then interacts with its representation in XHTML. To improve interaction with the user, we are also developing a method that will allow the user to select the desired wrapping instances directly on a rendition of the PDF.

Behind this graphical rendition, the document is represented as an attributed relational graph. Each block is represented as a vertex, and the vertices are interconnected with various logical and geometric relationships. Wrapping is then performed by the application of error-tolerant graph matching algorithms, such as those described in [16]. This approach is described in more detail in our forthcoming paper [13]. As well as the obvious benefits in user-friendliness, this method will also allow more powerful wrappers to be generated, for a wider variety of applications.

5 Application in Competitive Intelligence

In this chapter, we will give an example of a business case in the domain of competitive intelligence. This business case describes the process using Lixto for Web data extraction, transformation, and delivery to the data warehouse of the SAP Business Information Warehouse (SAP BW).

A company sells consumer electronics, such as digital cameras, computers and cellular phones, with a product catalogue of more than 1000 items (short: P1000). Before using the Lixto software, many employees of the company spent many hours a day searching the Web to collect information about their competitors' pricing for items from the P1000 catalogue. The price information retrieved was used for monthly price definitions. Product availability and regional price differences should also be included in the data analysis.

By using the Lixto suite, Web pages of online shops of several competitors are automatically searched on a daily basis. For a complete Web site, just one Lixto wrapper is necessary. For every product on an overview page in the online shop, the wrapper extracts all information (even from sub-pages with detailed information). By automatically clicking on the "next overview page" button at the bottom of the page and applying the same wrapping procedure to the succeeding overview Web page, all necessary information can be retrieved for all items sold in the online shop, from all overview pages and all sub-pages. Complete product information is retrieved, i.e. *price*, *manufacturer*, *model name*, *model description*, *availability*, *discount rates*, *combined offers*, etc. The wrapper generates a hierarchically organized XML data file in a defined standard data model. Highly nested structures representing the connections and interrelations between the information items, such as *price* and *combined offers*, are possible and allow for a detailed data analysis later.

Within the Lixto Transformation Server, the XML data from different wrappers is then aggregated, reformatted and normalized. For example, all price information (e.g. in £ Sterling or Swiss Francs) are normalized to the company group standard currency (Euros), and differences in taxation are accordingly considered to allow for a standardized price comparison. Finally, the data is reformatted within the Lixto Transformation Server into SOAP, so that the retrieved information can be integrated into the SAP BW using Web services.

The data is then automatically transferred to the SAP BW in an automatic ETL process. Within SAP BW, sophisticated pre-defined data analysis and workflow capabilities exist. Together with the Web data supplied by the Lixto Suite, it is now possible to automatically define prices on a weekly or even on a daily basis, taking into account short-term and regional market price fluctuations. This "intelligent pricing" can increase the company's revenue margins for their products and altogether increase the revenue per product. In practice, an increase between 1% and 4% can be achieved.

With Lixto, the whole process of defining wrappers and data flows is performed semi-automatically in a graphical user interface. Within the Lixto Transformation Server, graphical objects symbolize components, such as an integrator for the aggregation of data, or a deliverer for the transmission of information to other software systems. By drawing connecting arrows between these objects, the flow of data and the workflow are graphically defined. In our example, the time-consuming and mostly manual process of mapping items from the competitors' Web sites to equivalent items from the P1000 product list is accomplished in the GUI, allowing for fast and effective data mapping.

6 Conclusion

In this paper we have reported on the latest developments in the Lixto project. In the field of supervised data extraction from Web documents, we have highlighted the benefits of employing learning strategies to guide the user in selecting relevant information items to define patterns in wrappers. For unsupervised data extraction, we have revealed strategies for resource discovery and focused crawling, as well as automatic table extraction from Web documents. We have also discussed the related issue of extracting from non-HTML document formats such as PDF. Finally, we have given a brief description of a real-world business case that illustrates the applicability of Web data extraction technology in competitive intelligence solutions.

References

[1] Aiello, M., Monz, C., Todoran, L. and Worring, M: Document understanding for a broad class of documents. Int. J. of Document Anal. and Recog. **5(1)** (2002) 1–16

[2] Altamura, O., Esposito, F. and Malerba, D.: Transforming Paper Documents into XML Format with WISDOM++. Intl. J. of Doc. Anal. and Recog. **4(1)** (2001) 2–17

[3] Baumgartner, R., Flesca, S. and Gottlob, G.: Visual Web Information Extraction with Lixto. Proceedings of the 27th International Conference on Very Large Data Bases (VLDB 2001), Rome, Italy, (2001) 119–128

[4] Baumgartner, R., Ceresna, M. and Ledermüller G.: Automating Web Navigation in Web Data Extraction. Proceedings of International Conference on Intelligent Agents, Web Technology and Internet Commerce, Vienna, Austria (2005) (to appear)

[5] Blumer, A., Ehrenfeucht, A., Haussler, D. and Warmuth M. K.: Learnability and the Vapnik-Chervonenkis dimension. J. ACM **36(4)** (1989) 929–965

[6] Chakrabarti, S., van den Berg, M., Dom, B.: Focused Crawling: A New Approach to Topic-Specific Web Resource Discovery. Computer Networks. **31(11–16)** (1999) 1623–1640

[7] Ceresna, M. and Gottlob G.: Query Based Learning of XPath Fragments. Proceedings of Dagstuhl Seminar on Machine Learning for the Semantic Web (05071), Dagstuhl, Germany (2005)

[8] Embley, D. W.: Toward Semantic Understanding – An Approach Based on Information Extraction Ontologies. Proceedings of the Fifteenth Australasian Database Conference, Dunedin, New Zealand (2004) 3

[9] Gottlob, G., Koch, C.: A Formal Comparison of Visual Web Wrapper Generators. SOFSEM 2006: Theory and Practice of Computer Science, 32nd Conference on Current Trends in Theory and Practice of Computer Science, Merín, Czech Republic, (2006) 30–48

[10] Gottlob, G., Koch, C.: Monadic datalog and the expressive power of languages for Web information extraction. J. ACM **51(1)** (2004) 74–113

[11] Gottlob, G., Koch, C., Baumgartner, R., Herzog, M., Flesca, S.: The Lixto Data Extraction Project - Back and Forth between Theory and Practice. Proceedings of the Twenty-third ACM SIGACT-SIGMOD-SIGAR Symposium on Principles of Database Systems, Paris, France (2004) 1–12

[12] Gottlob, G., Koch, C., Pichler, R.: Efficient algorithms for processing XPath queries. ACM Trans. Database Syst. **30(2)** (2005) 444–491

[13] Hassan, T., Baumgartner, R.: Using Graph Matching Techniques to Wrap Data from PDF Documents. To appear in Proceedings of the 15th International World Wide Web Conference (Poster Track), Edinburgh, UK (2006)

[14] Hurst, M.: The Interpretation of Tables in Texts. PhD thesis, University of Edinburgh (2000)

[15] Levenshtein, V. I.: Binary Codes Capable of Correcting Spurious Insertions and Deletions of Ones. Russian Problemy Peredachi Informatsii. **1** (1965) 12–25

[16] Llados, J., Marti, E. and Villanueva, J. J.: Symbol Recognition by Error-Tolerant Subgraph Matching between Region Adjacency Graphs. IEEE Tran. on Pattern Anal. and Mach. Intel. **23(10)** (2001) 1137–1143

[17] Page, L., Brin, S.: The Anatomy of a Large-Scale Hypertextual Web Search Engine. Computer Networks. **30(1–7)** (1998) 107–117

[18] Silva, A. C., Alipio, J., Torgo, L.: Automatic Selection of Table Areas in Documents for Information Extraction. 11th Protuguese Conference on Artificial Intelligence, EPIA (2003) 460–465

[19] XML Path Language (XPath), Version 1.0. http://www.w3.org/TR/xpath

An Overview of a Scalable Distributed Database System SD-SQL Server

Witold Litwin[1], Soror Sahri[1], and Thomas Schwarz[2]

[1] CERIA, Paris-Dauphine University
75016 Paris, France
witold.litwin@dauphine.fr, Soror.Sahri@Dauphine.fr
[2] Santa Clara University,
California, USA
tjschwarz@scu.edu

Abstract. We present a scalable distributed database system called SD-SQL Server. Its original feature is dynamic and transparent repartitioning of growing tables, avoiding the cumbersome manual repartitioning that characterize current technology. SD-SQL Server re-partitions a table when an insert overflows existing segments. With the comfort of a single node SQL Server user, the SD-SQL Server user has larger tables or gets a faster response time through the dynamic parallelism. We present the architecture of our system, its implementation and the performance analysis. We show that the overhead of our scalable table management should be typically negligible.

1 Introduction

Databases (DBs) are now often huge and grow fast. Large tables are typically hash or range partitioned into segments stored at different storage sites. Current Data Base Management Systems (DBSs) such as SQL Server, Oracle or DB2, provide only static partitioning [1,5,11]. Growing tables have to overflow their storage space after some time. The database administrator (DBA) has then to manually redistribute the database. This operation is cumbersome, users need a more automatic solution, [1].

This situation is similar to that for file users forty years ago in the centralized environment. Efficient management of distributed data presents specific needs. Scalable Distributed Data Structures (SDDSs) address these needs for files, [6,7]. An SDDS scales transparently for an application through distributed splits of its buckets, whether hash, range or k-d based. In [7], we derived the concept of a Scalable Distributed DBS (SD-DBS) for databases. The SD-DBS architecture supports *scalable (distributed relational) tables*. As an SDDS, a scalable table accommodates its growth through the splits of its overflowing segments, located at SD-DBS *storage* nodes. Also like in an SDDS, we can use hashing, range partitioning, or k-d-trees. The storage nodes can be P2P or grid DBMS nodes. The users or the application, manipulate the scalable tables from a *client* node that is not a storage node, or from a *peer* node that is both, again as in an SDDS. The client accesses a scalable table only through its specific view, called the (*client*) *image*. It is a particular updateable distributed partitioned union view stored at a client. The application manipulates

D. Bell and J. Hong (Eds.): BNCOD 2006, LNCS 4042, pp. 16 – 35, 2006.

scalable tables using images directly, or their *scalable* views. These views involve scalable tables through the references to the images.

Every image, one per client, hides the partitioning of the scalable table and dynamically adjusts to its evolution. The images of the same scalable table may differ among the clients and from the actual partitioning. The image adjustment is lazy. It occurs only when a query to the scalable table finds an outdated image. To prove the feasibility of an SD-DBS, we have built a prototype called SD-SQL Server. The system generalizes the basic SQL Server capabilities to the scalable tables. It runs on a collection of SQL Server linked nodes. For every standard SQL command under SQL Server, there is an SD-SQL Server command for a similar action on scalable tables or views. There are also commands specific to SD-SQL Server client image or node management.

Below we present the architecture and the implementation of our prototype as it stands in its 2005 version. Related papers [14, 15] discuss the user interface. Scalable table processing creates an overhead and our design challenge was to minimize it. The performance analysis proved this overhead negligible for practical purpose. The present capabilities of SQL Server allow a scalable table to reach 250 segments at least. This should suffice for scalable tables reaching very many terabytes. SD-SQL Server is the first system with the discussed capabilities, to the best of our knowledge. Our results pave the way towards the use of the scalable tables as the basic DBMS technology.

Below, Section 2 presents the SD-SQL Server architecture. Section 3 recalls the basics of the user interface. Section 4 discusses our implementation. Section 5 shows experimental performance analysis. Section 6 discusses the related work. Section 7 concludes the presentation.

2 SD-SQL Server Architecture

Fig. 1 shows the current SD-SQL Server architecture, adapted from the reference architecture for an SD-DBS in [8]. The system is a collection of SD-SQL Server nodes. An *SD-SQL Server node* is a linked SQL Server node that in addition is declared as an SD-SQL Server node. This declaration is made as an SD-SQL Server command or is part of a dedicated SQL Server script run on the first node of the collection. We call the first node the *primary node.* The primary node registers all other current SD-SQL nodes. We can add or remove these dynamically, using specific SD-SQL Server commands. The primary node registers the nodes on itself, in a specific SD-SQL Server database called the *meta-database* (MDB). An *SD-SQL Server database* is an SQL Server database that contains an instance of SD-SQL Server specific *manager* component. A node may carry several SD-SQL Server databases.

We call an SD-SQL Server database in short a *node database* (NDB). NDBs at different nodes may share a (proper) database name. Such nodes form an SD-SQL Server *scalable (distributed) database* (SDB). The common name is the *SDB name.* One of NDBs in an SDB is *primary*. It carries the meta-data registering the current NDBs, their nodes at least. SD-SQL Server provides the commands for scaling up or down an SDB, by adding or dropping NDBs. For an SDB, a node without its NDB is (an SD-SQL Server) *spare* (node). A spare for an SDB may already carry an NDB of another SDB. Fig 1 shows an SDB, but does not show spares.

Each manager takes care of the SD-SQL Server specific operations, the user/application command interface especially. The procedures constituting the manager of an NDB are themselves kept in the NDB. They apply internally various SQL Server commands. The SQL Servers at each node entirely handle the inter-node communication and the distributed execution of SQL queries. In this sense, each SD-SQL Server runs at the top of its linked SQL Server, without any specific internal changes of the latter.

An SD-SQL Server NDB is a *client*, a *server*, or a *peer*. The client manages the SD-SQL Server node user/application interface only. This consists of the SD-SQL Server specific commands and from the SQL Server commands. As for the SQL Server, the SD-SQL specific commands address the schema management or let to issue the queries to scalable tables. Such a *scalable* query may invoke a scalable table through its image name, or indirectly through a scalable view of its image, involving also, perhaps, some *static* tables, i.e., SQL Server only.

Internally, each client stores the images, the local views and perhaps *static* tables. These are tables created using the SQL Server *CREATE TABLE* command (only). It also contains some SD-SQL Server meta-tables constituting the catalog **C** at Fig 1. The catalog registers the client images, i.e., the images created at the client.

When a scalable query comes in, the client checks whether it actually involves a scalable table. If so, the query must address the local image of the table. It can do it directly through the image name, or through a scalable view. The client searches therefore for the images that the query invokes. For every image, it checks whether it conforms to the actual partitioning of its table, i.e., unions all the existing segments. We recall that a client view may be outdated. The client uses **C**, as well as some server meta-tables pointed to by **C** that define the actual partitioning. The manager dynamically adjusts any outdated image. In particular, it changes internally the scheme of the underlying SQL Server partitioned and distributed view, representing the image to the SQL Server. The manager executes the query, when all the images it uses prove up to date.

A *server* NDB stores the segments of scalable tables. Every segment at a server belongs to a different table. At each server, a segment is internally an SQL Server table with specific properties. First, SD-SQL Server refers to in the specific catalogue in each server NDB, called **S** in the figure. The meta-data in **S** identify the scalable table each segment belongs to. They indicate also the segment size. Next, they indicate the servers in the SDB that remain available for the segments created by the splits at the server NDB. Finally, for a *primary* segment, i.e., the first segment created for a scalable table, the meta-data at its server provide the actual partitioning of the table.

Next, each segment has an *AFTER* trigger attached, not shown in the figure. It verifies after each insert whether the segment overflows. If so, the server splits the segment, by range partitioning it with respect to the table (partition) key. It moves out enough upper tuples so that the remaining (lower) tuples fit the size of the splitting segment. For the migrating tuples, the server creates remotely one or more new segments that are each half-full (notice the difference to a B-tree split creating a single new segment). Furthermore, every segment in a multi-segment scalable table carries an SQL Server *check constraint*. Each constraint defines the partition (primary) key range of the segment. The ranges partition the key space of the table. These conditions let the SQL Server distributed partitioned view to be updateable, by the inserts and deletions in particular. This is a necessary and sufficient condition for a scalable table under SD-SQL Server to be updateable as well.

Finally a *peer* NDB is both a client and a server NDB. Its node DB carries all the SD-SQL Server meta-tables. It may carry both the client images and the segments. The meta-tables at a peer node form logically the catalog termed **P** at the figure. This one is operationally, the union of **C** and **S** catalogs.

Every SD-SQL Server node is *client*, *server* or *peer* node. The peer accepts every type of NDB. The client nodes only carry client NDBs & server nodes accept server NDBs only. Only a server or peer node can be the primary one or may carry a primary NDB. To illustrate the architecture, Fig 1 shows the NDBs of some SDB, on nodes *D1…Di+1*. The NDB at *D*1 is a client NDB that thus carries only the images and views, especially the scalable ones. This node could be the primary one, provided it is a peer. It interfaces the applications. The NDBs on all the other nodes until *Di* are server NDBs. They carry only the segments and do not interface (directly) any applications. The NDB at *D2* could be here the primary NDB. Nodes *D2…Di* could be peers or (only) servers. Finally, the NDB at *Di*+1 is a peer, providing all the capabilities. Its node has to be a peer node.

The NDBs carry a scalable table termed *T*. The table has a scalable index *I*. We suppose that *D*1 carries the *primary* image of *T*, named *T* at the figure. This image name is also as the SQL Server view name implementing the image in the NDB. SD-SQL Server creates the primary image at the node requesting the creation of a scalable table, while creating the table. Here, the primary segment of table *T* is supposed at *D*2. Initially, the primary image included only this segment. It has evolved since, following the expansion of the table at new nodes, and now is the distributed partitioned union-all view of *T* segments at servers *D*2…*Di*. We symbolize this image with the dotted line running from image *T* till the segment at *Di*. Peer *Di*+1 carries a *secondary* image of table *T*. Such an image interfaces the application using *T* on a node other than the table creation one. This image, named *D1_T*, for reasons we discuss below, differs from the primary image. It only includes the primary segment. We symbolize it with the dotted line towards *D*2 only. Both images are outdated. Indeed, server *Di* just split its segment and created a new segment of *T* on *Di*+1. The arrow at *Di* and that towards *Di*+1 represent this split. As the result of the split, server *Di* updated the meta-data on the actual partitioning of *T* at server *D*2 (the dotted arrow from *Di* to *D*2). The split has also created the new segment of the scalable index *I*. None of the two images refers as yet to the new segment. Each will be actualized only once it gets a scalable query to *T*. At the figure, they are getting such queries, issued using respectively the SD-SQL Server *sd_select* and *sd_insert* commands. We discuss the SD-SQL Server command interface in the next sections.

Notice finally that in the figure that the segments of *T* are all named *_D1_T*. This represents the couple (creator node, table name). It is the proper name of the segment as an SQL Server table in its NDB. Similarly for the secondary image name, except for the initial '_'. The image name is the local SQL Server view name. We explain these naming rules in Section 4.1.

3 Command Interface

3.1 Overview

The application manipulates SD-SQL Server objects essentially through new SD-SQL Server dedicated commands. Some commands address the node management, including the management of SDBs, NDBs. Other commands manipulate the scalable tables. These commands perform the usual SQL schema manipulations and queries that can however now involve scalable tables (through the images) or (scalable) views of the scalable tables. We call the SD-SQL Server commands scalable. A scalable command may include additional parameters specific to the scalable environment, with respect to its static (SQL Server) counterpart. Most of scalable commands apply also to static tables and views. The application using SD-SQL Server may alternatively directly invoke a static command. Such calls are transparent to SD-SQL Server managers.

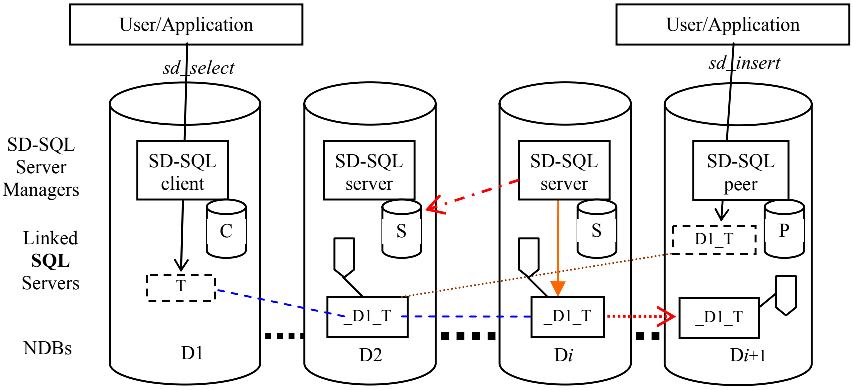


Fig. 1. SD-SQL Server Architecture

Details of all the SD-SQL Server commands are in [14, 15]. The rule for an SD-SQL Server command performing an SQL operation is to use the SQL command name (verb) prefixed with *'sd_'* and with all the blanks replaced with '_'. Thus, e.g., SQL *SELECT* became SD-SQL *sd_select*, while SQL *CREATE TABLE* became *sd_create_table*. The standard SQL clauses, with perhaps additional parameters follow the verb, specified as usual for SQL. The whole specification is however within additional quotes ' '. The rationale is that SD-SQL Server commands are implemented as SQL Server stored procedures. The clauses pass to SQL Server as the parameters of a stored procedure and the quotes around the parameter list are mandatory.

The operational capabilities of SD-SQL Server should suffice for many applications. The *SELECT* statement in a scalable query supports the SQL Server allowed selections, restrictions, joins, sub-queries, aggregations, aliases…etc. It also allows for the INTO clause that can create a scalable table. However, the queries to the scalable multi-database views are not possible at present. The reasons are the limitation of the SQL Server meta-tables that SD-SQL Server uses for the parsing.

Moreover, the *sd_insert* command over a scalable table lets for any insert accepted by SQL Server for a distributed partitioned view. This can be a new tuple insert, as well as a multi-tuple insert through a *SELECT* expression, including the *INTO* clause. The *sd_update* and *sd_delete* commands offer similar capabilities. In contrast, some of SQL Server specific SQL clauses are not supported at present by the scalable commands; for instance, the *CASE OF* clause.

We recall the SD-SQL Server command interface by the motivating example. modelled upon our benchmark application that is SkyServer DB, [2].

3.2 Motivating Example

A script file creates the first ever (primary) SD-SQL Server (scalable) node at a collection of linked SQL Server nodes. We can create the primary node as a peer or a server, but not as a client. After that, we can create additional nodes using the *sd_create_node* command. Here, the script has created the primary SD-SQL Server node at SQL Server linked node at our *Dell1* machine. We could set up this node as server or peer, we made the latter choice. The following commands issued at *Dell1* create then further nodes, alter *Dell3* type to peer, and finally create our *SkyServer* SDB at *Dell1*:

```
sd_create_node ‘Dell2’   /* Server by default */;

sd_create_node ‘Dell3, ‘client’ ;

sd_create_node ‘Ceria1’,’peer’

sd_alter_ node ‘Dell3’, ‘ADD server’ ;

sd_create_scalable_database ‘SkyServer, ‘Dell1’
```

Our SDB has now one NDB termed *SkyServer*. This NDB is the primary one of the SDB and is a server NDB. To query , *SkyServer* one needs at least one client or peer NDB. We therefore append a client NDB at *Dell3* client node:

```
sd_create_node_database ‘SkyServer’, ‘Dell3’, ‘client’
```

From now on, *Dell3* user opens *Skyserver* SDB through the usual SQL Server USE *Skyserver* command (which actually opens *Dell3.Skyserver* NDB). The *Skyserver* users are now furthermore able to create scalable tables. The *Dell3* user starts with a *PhotoObj* table modelled on the static table with the same name, [2]. The user wishes the segment capacity of 10000 tuples. S/he chooses this parameter for the efficient distributed query processing. S/he also wishes the *objid* key attribute to be the partition key. In SD-SQL Server, a partition key of a scalable table has to be a single key attribute. The requirement comes from SQL Server, where it has to be the case of a table, or tables, behind a distributed partitioned updatable view. The key attribute of *PhotoObj* is its *objid* attribute. The user issues the command:

```
sd_create_table ‘PhotoObj (objid BIGINT PRIMARY KEY…)’, 10000
```

We did not provide the complete syntax, using ‘…’ to denote the rest of the scheme beyond the key attribute. The *objid* attribute is the partition key implicitly, since it is here the only key attribute. The user creates furthermore a scalable table *Neighbors*, modelled upon the similar one in the static *Skyserver*. That table has three key attributes. The *objid* is one of them and is the foreign key of *PhotoObj*. For this reason, the user wishes it to be the partition key. The segment capacity should now be 500 tuples. Accordingly, the user issues the command:

```
sd_create_table 'Neighbors (htmid BIGINT, objid BIGINT, Neighborobjid
                BIGINT) ON PRIMARY KEY…)', 500, 'objid'
```

The user indicated the partition key. The implicit choice would go indeed to *htmid*, as the first one in the list of key attributes. The *Dell3* user decides furthermore to add attribute *t* to *PhotoObj* and prefer a smaller segment size:

```
sd_alter_table 'PhotoObj ADD t INT, 1000
```

Next the user decides to create a scalable index on *run* attribute:

```
sd_create_index 'run_index ON Photoobj (run)'
```

Splits of *PhotoObj* will propagate *run_index* to any new segment.

The *PhotoObj* creation command created the primary image at *Dell3*. The *Dell3* user creates now the secondary image of *PhotoObj* at *Ceria1* node for the *SkyServer* user there::

```
sd_create_image 'Ceria1', 'PhotoObj'
```

The image internal name is *SD.Dell3_Photoobj*, as we discuss in Section 0 and [15]. The *Ceria1* user who wishes a different local name such as *PhotoObj* uses the SQL Server *CREATE VIEW* command. Once the *Ceria1* user does not need its image anymore, s/he may remove it through the command:

```
sd_drop_image 'SD.Dell3_Photoobj'
```

Assuming that the image was not dropped however yet, our *Dell3* user may open *Skyserver* SDB and query *PhotoObj*:

```
USE Skyserver                    /* SQL Server command */

sd_insert 'INTO PhotoObj SELECT * FROM Ceria5.Skyserver-S.PhotoObj

sd_select '* FROM PhotoObj' ;

sd_select 'TOP 5000 * INTO PhotoObj1 FROM PhotoObj', 500
```

The first query loads into our *PhotoObj* scalable table tuples from some other *PhotoObj* table or view created in some *Skyserver* DB at node *Ceria5*. This DB could be a static, i.e., SQL Server only, DB. It could alternatively be an NDB of "our" *Skyserver* DB. The second query creates scalable table *PhotoObj1* with segment size of 500 and copies there 5000 tuples from *PhotoObj*, having the smallest values of *objid*. See [15] for examples of other scalable commands.

4 Command Processing

We now present the basics of SD_SQL Server command processing. We start with the naming rules and the meta-tables. Next, we discuss the scalable table evolution.

We follow up with the image processing. For more on the command processing see [14].

4.1 Naming Rules

SD-SQL Server has its own system objects for the scalable table management. These are the node DBs, the meta-tables, the stored procedures, the table and index segments and the images. All the system objects are implemented as SQL Server objects. To avoid the name conflicts, especially between the SQL Server names created by an SD-SQL Server application, there are the following naming rules, partly illustrated at Fig. 1. Each NDB has a dedicated user account 'SD' for SD-SQL Server itself. The application name of a table, of a database, of a view or of a stored procedure, created at an SD-SQL Server node as public (*dbo*) objects, should not be the name of an SD-SQL Server command. These are SD-SQL server keywords, reserved for its commands (in addition to the same rule already enforced by SQL Server for its own SQL commands). The technical rationale is that SD-SQL Server commands are public stored procedures under the same names. An SQL Server may call them from any user account.

A scalable table *T* at a NDB is a public object, i.e., its SQL Server name is *dbo.T*. It is thus unique regardless of the user that has created it1. In other words, two different SQL Server users of a NDB cannot create each a scalable table with the proper name T. They can still do it for the static tables of course. Besides, two SD-SQL Server users at different nodes may each create a scalable table with the proper name T.

A segment of scalable table created with proper name *T*, at SQL Server node *N*, bears for any SQL Server the table name *SD._N_T* within its SD-SQL Server node (its NDB more specifically, we recall). We recall that SD-SQL Server locates every segment of a scalable table at a different node.

A primary image of a scalable table *T* bears the proper name *T*. Its global name within the node is *dbo.T*. This is also the proper name of the SQL Server distributed partitioned view implementing the primary view.

Any secondary image of scalable table created by the application with proper name *T*, within the table names at client or peer node *N*, bears the global name at its node *SD.N_T*. In Fig 1, e.g., the proper name of secondary image denoted *T* (2) would actually be *D1_T*.

We recall that since SD-SQL Server commands are public stored procedures, SQL Server automatically prefixes all the proper names of SD-SQL public objects with *dbo.* in every NDB, to prevent a name conflict with any other owner within NDB. The rules avoid name conflicts between the SD-SQL Server private application objects and SD-SQL system objects, as well as between SD-SQL Server system objects themselves. See [14] for more examples of the naming rules.

[1] In the current version of the prototype.

4.2 Meta-tables

These tables constitute internally SQL Server tables searched and updated using the stored procedures with SQL queries detailed in [8]. All the meta-tables are under the user name *SD*, i.e., are prefixed within their NDB with '*SD.*'.

The *S*-catalog exists at each server and contains the following tables.

- *SD.RP (SgmNd, CreatNd, Table)*. This table at node *N* defines the scalable distributed partitioning of every table *Table* originating within its NDB, let it be *D*, at some server *CreatNd*, and having its primary segment located at *N*. Tuple (*SgmNd, CreatNd, Table*) enters *N.D.SD.RP* each time *Table* gets a new segment at some node *SgmNd*. For example, tuple (*Dell5, Dell1, PhotoObj)* in *Dell2.D.SD.RP* means that scalable table *PhotoObj* was created in *Dell1.D*, had its primary segment at *Dell2.D,* and later got a new segment *_Dell1_PhotoObj* in *Dell5.D*. We recall that a segment proper name starts with '_', being formed as in Fig. 1.
- *SD.Size (CreatNd, Table, Size).* This table fixes for each segment in some NDB at SQL Server node *N* the maximal size (in tuples) allowed for the segment. For instance, tuple (*Dell1, PhotoObj, 1000*) in *Dell5.DB1.SD.Size* means that the maximal size of the *Dell5* segment of *PhotoObj* scalable table initially created in *Dell1.DB1* is 1000. We recall that at present all the segment of a scalable table have the same sizes.
- *SD.Primary (PrimNd, CreatNd, Table).* A tuple means here that the primary segment of table T created at client or peer CreatNd is at node PrimNd. The tuple points consequently to *SD.RP* with the actual partitioning of *T*. A tuple enters *N.SD.Primary* when a node performs a table creation or split and the new segment lands at *N*. For example, tuple *(Dell2, Dell1, PhotoObj)* in *SD.Primary* at node *Dell5* means that there is a segment _Dell1_PhotoObj resulting from the split of *PhotoObj* table, created at *Dell1* and with the primary segment at *Dell2*.

The *C*-catalog has two tables:

- Table *SD.Image (Name, Type, PrimNd,Size)* registers all the local images. Tuple (I, T, P, S) means that, at the node, there is some image with the (proper) name I, primary if T = .true, of a table using P as the primary node that the client sees as having S segments. For example, tuple *(PhotoObj, true, Dell2, 2)* in *Dell1.SD.C-Image* means that there is a primary image dbo.PhotoObj at Dell1 whose table seems to contain two segments. SD-SQL Server explores this table during the scalable query processing.
- Table *SD.Server (Node)* provides the server (peer) node(s) at the client disposal for the location of the primary segment of a table to create. The table contains basically only one tuple. It may contain more, e.g., for the fault tolerance or load balancing.

Finally, the *P*-catalogue, at a peer, is simply the union of *C*-catalog and *S*-catalog. In addition, each NDB has two tables:

- *SD.SDBNode (Node)*. This table points towards the primary NDB of the SDB. It could indicate more nodes, replicating the SDB metadata for fault-tolerance or load balancing.
- *SD.MDBNode (Node)*. This table points towards the primary node. It could indicate more nodes, replicating the MDB for the fault-tolerance or load balancing.

There are also meta-tables for the SD-SQL Server node management and SDB management. These are the tables:

- *SD.Nodes (Node, Type)*. This table is in the MDB. Each tuple registers an SD-SQL Server node currently forming the SD-SQL configuration. We recall that every SD-SQL Server node is an SQL Server linked server declared SD-SQL Server node by the initial script or the *sd_create_node* command. The values of *Type* are '*pee*r', '*server*' or '*client*'.
- *SD.SDB (SDB_Name, Node, NDBType).* This table is also in the MDB. Each tuple registers an SDB. For instance, tuple *(DB1, Dell5, Peer)* means that there is an SDB named *DB1*, with the primary NDB at *Dell5*, created by the command *sd_create_scalable_database 'DB1', 'Dell5', 'peer'.*
- *SD.NDB (Node, NDBType)*. This meta-table is at each primary NDB. It registers all the NDBs currently composing the SDB. The *NDBType* indicates whether the NDB is a peer, server or client.

4.3 Scalable Table Evolution

A scalable table *T* grows by getting new segments, and shrinks by dropping some. The dynamic *splitting* of overflowing segments performs the former. The *merge* of under-loaded segments may perform the latter. There seems to be little practical interest for merges, just as implementations of B-tree merges are rare. We did not consider them for the current prototype. We only present the splitting now. The operation aims at several goals. We first enumerate them, then we discuss the processing:

1. The split aims at removing the overflow from the splitting segment by migrating some of its tuples into one or several new segment(s). The segment should possibly stay at least half full. The new segments should end up half full. The overall result is then at least the typical "good" load factor of 69 %.
2. Splitting should not delay the commit of the insert triggering it. The insert could timeout otherwise. Through the performance measures in Section 0, we expect the split to be often much longer than a insert.
3. The allocation of nodes to the new segments aims at the random node load balancing among the clients and /or peers. However, the splitting algorithm also allocates the same nodes to the successive segments of different scalable tables of the same client. The policy aims at faster query execution, as the queries tend to address the tables of the same client.
4. The concurrent execution of the split and of the scalable queries should be serializable. A concurrent scalable query to the tuples in an overflowing segment, should either access them before any migrate, or only when the split is over.

We now show how SD-SQL Server achieves these goals. The creation of a new segment for a scalable table *T* occurs when an insert overflows the capacity of one of its segments, declared in local *SD.Size* for *T.* At present, all the segments of a scalable table have the same capacity, denoted *b* below, and defined in the *sd_create_table* command. The overflow may consist of arbitrarily many tuples, brought by a single *sd_insert* command with the *SELECT* expression (unlike in a record-at-the-time operations, e.g., as in a B-tree). A single *sd_insert* may further overflow several segments. More precisely, we may distinguish the cases of a (*single segment*) *tuple insert* split, of a *single segment bulk insert* split and of a *multi-segment* (*insert*) split. The bulk inserts correspond to the *sd_insert* with the *SELECT* expression.

In every *sd_insert* case, the *AFTER* trigger at every segment getting tuples tests for overflow, [8]. The positive result leads to the split of the segment, according to the following *segment partitioning scheme*. We first discuss the basic case of the partition key attribute being the (single-attribute) primary key. We show next the case of the multi-attribute key, where the partition key may thus present duplicates. We recall that the partition key under SQL Server must be a (single) key attribute. In every case, the scheme adds $N \geq 1$ segments to the table, with N as follows.

Let P be the (overflowing) set of all the tuples in one of, or the only, overflowing segment of T, ordered in ascending order by the partition key. Each server aims at cutting its P, starting from the high-end, into successive portions $P_1 \ldots P_N$ consisting each of *INT (b/2)* tuples. Each portion goes to a different server to become a possibly half-full new segment of T. The number N is the minimal one leaving at most b tuples in the splitting segment. To fulfil goal (1) above, we thus always have $N = 1$ for a tuple insert, and the usual even partitioning (for the partition key without duplicates). A single segment bulk insert basically leads to $N \geq 1$ half-full new segments. The splitting one ends up between half-full and full. We have in both cases:

$$N = \lceil (\mathrm{Card}(P) - b) \ / \ \mathrm{INT}\ (b/2) \rceil$$

The same scheme applies to every splitting segment for a multiple bucket insert. If the partition key presents the duplicates, the result of the calculus may differ a little in practice, but arbitrarily in theory. The calculus of each P_i incorporates into it all the duplicates of the lowest key value, if there is any. The new segment may start more than half-full accordingly, even overflowing in an unlikely bad case. The presence of duplicates may in this way decrease N. It may theoretically even happen that all the partition key values amount to a single duplicate. We test this situation in which case no migration occurs. The split waits till different key values come in. The whole duplicate management is potentially subject of future work optimizing the P_i calculus.

The *AFTER* trigger only tests for overflow, to respond to goal (2). If necessary, it launches the actual splitting process as an asynchronous job called *splitter* (for performance reasons, see above). The splitter gets the segment name as the input parameter. To create the new segment(s) with their respective portions, the splitter acts as follows. It starts as a distributed transaction at the repeatable read isolation level. SQL Server uses then the shared and exclusive tuple locks according to the basic 2PL protocol. The splitter first searches for *PrimNd* of the segment to split in *Primary* meta-table. If it finds the searched tuple, SQL Server puts it under a shared lock. The splitter requests then an exclusive lock on the tuple registering the splitting segment in *RP* of the splitting table that is in the NDB at *PrimNd* node. As we show later, it gets the lock if there are no (more) scalable queries or other commands in progress involving the segment. Otherwise it would encounter at least a shared lock at the tuple. SQL Server would then block the split until the end of the concurrent operation. In unlikely cases, a deadlock may result. The overall interaction suffices to provide the serializability of every command and of a split, [14]. If the splitter does not find the tuple in *Primary*, it terminates. As it will appear, it means that a delete of the table occurred in the meantime.

From now on, there cannot be a query in progress on the splitting segment; neither can another splitter lock it. It should first lock the tuple in *RP*. The splitter safely verifies the segment size. An insert or deletion could change it in the meantime. If the segment does not overflow anymore, the splitter terminates. Next, it determines *N* as above. It finds *b* in the local *SD.Size* meta-table. Next, it attempts to find *N* NDBs without any segment of *T* as yet. It searches for such nodes through the query to NDB meta-table, requesting every NDB in the SDB which is a server or peer and not yet in *RP* for *T*. Let $M \geq 0$ be the number of NDBs found. If $M = N$, then the splitter allocates each new segment to a node. If $M > N$, then it randomly selects the nodes for the new segments. To satisfy goal (3) above, the selection is nevertheless driven, at the base of the randomness generation, by *T* creation NDB name. Any two tables created by the same client node share the same primary NDB, have their 1st secondary segments at the same (another) server as well etc… One may expect this policy to be usually beneficial for the query processing speed. At the expense however, perhaps of the uniformity of the processing and storage load among the server NDBs.

If $M < N$, it means that the SDB has not enough of NDBs to carry out the split. The splitter attempts then to extend the SDB with new server or peer NDBs. It selects (exclusively) possibly enough nodes in the meta-database which are not yet in the SDB. It uses the meta-tables *Nodes* and *SDB* in the MDB and *NDB* at the primary SDB node. If it does not succeeds the splitting halts with a message to the administrator. This one may choose to add nodes using *sd_create_node* command. Otherwise, the splitter updates the *NDB* meta-table, asks SQL Server to create the new NDBs (by issuing the *sd_create_node_database command*) and allocates these to the remaining new segment(s).

Once done with the allocation phase, the splitter creates the new segments. Each new segment should have the schema of the splitting one, including the proper name, the key and the indexes, except for the values of the *check constraint* as we discuss below. Let *S* be here the splitting segment, let *p* be *p* = INT (*b*/2), let *c* be the key, and let S_i denote the new segment at node *Ni*, destined for portion P_i. The creation of the new segments loops for *i* = 1…*N* as follows.

It starts with the SQL Server query in the form of :

```
SELECT TOP p (*) WITH TIES INTO Ni.Si FROM S ORDER BY c ASC
```

The option "with ties" takes care of the duplicates[2]. Next, the splitter finds the partition key *c* of *S* using the SQL Server system tables and alters S_i scheme accordingly. To find *c*, it joins SQL Server system tables *information_schema.Tables* and *information_schema.TABLE_CONSTRAINTS* on the *TABLE_SCHEMA*, *CONSTRAINT_SCHEMA* and *CONSTRAINT_NAME* columns. It also determines the indexes on *S* using the SQL Server stored procedure *sp_helpindex*. It creates then the same indexes on S_i using the SQL Server *create index* statements. Finally, it creates the check constraint on S_i as we describe soon. Once all this done, it registers the new

[2] Actually, it first performs the test whether the split can occur at all, as we discussed, using the similar query with count(*).

segment in the SD-SQL Server meta-tables. It inserts the tuples describing S_i into (i) *Primary* table at the new node, and (ii) *RP* table at the primary node of *T*. It also inserts the one with the $\mathrm{S_i}$ size into *Size* at the new node. As the last step, it deletes from *S* the copied tuples. It then moves to the processing of next P_i if any remains. Once the loop finishes, the splitter commits which makes SQL Server to release all the locks.

The splitter computes each *check constraint* as follows. We recall that, if defined for segment *S*, this constraint *C(S)* defines the low *l* and/or the high *h* bounds on any partition key value *c* that segment may contain. SQL Server needs for updates through a distributed partitioned view, a necessity for SD-SQL Server. Because of our duplicates management, we have: $C(S) = \{\ c :\ \ l \le c < h\ \}$. Let thus h_i be the highest key values in portion $P_{i>1}$, perhaps, undefined for P_1. Let also h_{N+1} be the highest key remaining in the splitting segment. Then the low and high bounds for new segment S_i getting P_i is $l = h_{i+1}$ and $h = h_i$ .The splitting segment keeps its *l*, if it had any, while it gets as new *h* the value $h' = h_{N+1}$, where $h' < h$. The result makes *T* always range partitioned.

4.4 Image Processing

4.4.1 Checking and Adjustment

A scalable query invokes an image of some scalable table, let it be *T*. The image can be primary or secondary, invoked directly or through a (scalable) view. SD-SQL Server produces from the scalable query, let it be *Q*, an SQL Server query *Q'* that it passes for the actual execution. *Q'* actually addresses the distributed partitioned view defining the image that is *dbo.T*. It should not use an outdated view. *Q'* would not process the missing segments and *Q* could return an incorrect result.

Before passing *Q'* to SQL Server, the client manager first checks the image correctness with respect to the actual partitioning of *T*. *RP* table at *T* primary server let to determine the latter. The manager retrieves from *Image* the presumed size of *T*, in the number of segments, let it be S_I. It is the *Size* of the (only) tuple in *Image* with *Name* = '*T*'. The client also retrieves the *PrimNd* of the tuple. It is the node of the primary NDB of *T*, unless the command *sd_drop_node_database* or *sd_drop_node* had for the effect to displace it elsewhere. In the last case, the client retrieves the *PrimNd* in the *SD.SDB* meta-table. We recall that this NDB always has locally the same name as the client NDB. They both share the SDB name, let it be *D*. Next, the manager issues the multi-database SQL Server query that counts the number of segments of *T* in *PrimNd.D.SD.RP.* Assuming that SQL Server finds the NDB, let S_A be this count. If $S_A = 0$, then the table was deleted in the meantime. The client terminates the query. Otherwise, it checks whether $S_I = S_A$. If so, the image is correct. Otherwise, the client adjusts the *Size* value to S_A. It also requests the node names of *T* segments in *PrimNd.D.SD.RP.* Using these names, it forms the segment names as already discussed. Finally, the client replaces the existing *dbo.T* with the one involving all newly found segments.

The view *dbo.T* should remain the correct image until the scalable query finishes exploring *T*. This implies that no split modifies partitioning of *T* since the client requested the segment node names in *PrimNd.D.SD.RP*, until *Q* finishes manipulating *T*. Giving our splitting scheme, this means in practice that no split starts in the meantime the deletion phase on any *T* segment. To ensure this, the manager requests from SQL Server to process every *Q* as a distributed transaction at the repeatable read isolation level. We recall that the splitter uses the same level. The counting in *PrimNd.D.SD.RP* during *Q* processing generates then a shared lock at each selected tuple. Any split of *T* in progress has to request an exclusive lock on some such tuple, registering the splitting segment. According to its 2PL protocol, SQL Server would then block any *T* split until *Q* terminates. Vice versa, *Q* in progress would not finish, or perhaps even start the S_A count and the *T* segment names retrieval until any split in progress ends. *Q* will take then the newly added *T* segments into account as well. In both cases, the query and split executions remain serializable.

Finally, we use a *lazy schema validation* option for our linked SQL Servers, [1, 10]. When starting *Q'*, SQL Server drops then the preventive checking of the schema of any remote table referred to in a partitioned view. The run-time performance obviously must improve, especially for a view referring to many tables [9]. The potential drawback is a run-time error generated by a discrepancy between the compiled query based on the view, *dbo.T* in our case, and some alterations of schema *T* by SD-SQL Server user since, requiring *Q* recompilation on the fly.

Example. Consider query *Q* to *SkyServer* peer NDB at the *Ceria1*:

sd_select ' from PhotoObj'*

Suppose that *PhotoObj* is here a scalable table created locally, and with the local primary segment, as typically for a scalable table created at a peer. Hence, *Q'* should address *dbo.PhotoObj* view and is here:

*SELECT * FROM dbo.PhotoObj*

Consider that *Ceria1* manager processing *Q* finds *Size* = 1 in the tuple with of *Name* = *'PhotoObj'* retrieved from its *Image* table. The client finds also *Ceria1* in the *PrimeNd* of the tuple. Suppose further that *PhotoObj* has in fact also two secondary segments at *Dell1 and Dell2.* The counting of the tuples with *Table* = '*PhotoObj'* and *CreatNd = 'Ceria1' in Ceria1.SkyServer.SD.RP* reports then S_A = 3. Once SQL Server retrieves the count, it would put on hold any attempt to change *T* partitioning till *Q* ends. The image of *PhotoObj* in *dbo.PhotoObj* turns out thus not correct. The manager should update it. It thus retrieves from *Ceria1.SkyServer.SD.RP* the *SgmNd* values in the previously counted tuples. It gets {*Dell1*, *Dell2, Ceria1*}. It generates the actual segment names as '*_Dell1_PhotoObj*' etc. It recreates *dbo.PhotoObj* view and updates *Size* to 3 in the manipulated tuple in its *Image* table. It may now safely pass *Q'* to SQL Server.

4.4.2 Binding

A *scalable* query consists of a query command to an SD-SQL Server client (peer) followed by a (scalable) *query expression.* We recall that these commands are *sd_select*, *sd_insert*, *sd_update* and *sd_delete*. Every scalable query, unlike a static one, starts with an *image binding* phase that determines every image on which a table or a view name in a query depends. The client verifies every image before it passes to SQL Server any query to the scalable table behind the image. We now present the processing of scalable queries under SD-SQL Server. We only discuss image binding and refer to more documentation for each command to [14].

The client (manager) parses every *FROM* clause in the query expression, including every sub-query, for the table or view names it contains. The table name can be that of a scalable one, but then is that of its primary image. It may also be that of a secondary image. Finally, it can be that of a static (base) table. A view name may be that of a scalable view or of a static view. Every reference has to be resolved. Every image found has to be verified and perhaps adjusted before SD-SQL Server lets SQL Server to use it, as already discussed.

The client searches the table and view names in *FROM* clauses, using the SQL Server x*p_sscanf* function, and some related processing. This function reads data from the string into the argument locations given by each format argument. We use it to return all the objects in the *FROM* clause. The list of objects is returned as it appears in the clause *FROM*, i.e. with the ‘,’ character. Next, SD-SQL Server parses the list of the objects and takes every object name alone by separating it from it *FROM* clause list. For every name found, let it be *X*, assumed a proper name, the client manager proceeds as follows:

It searches for *X* within *Name* attribute of its *Image* table. If it finds the tuple with *X*, then it puts *X* aside into *check_image* list, unless it is already there.

Otherwise, the manager explores with *T* the *sysobjects* and s*ysdepends* tables of SQL Server. Table *sysobjects* provides for each object name its type (*V* = view, *T* = base table…) and internal *Id*, among other data. Table s*ysdepends* provides for each view, given its *Id*, its local (direct) dependants. These can be tables or views themselves. A multi-database base view does not have direct remote dependants in *sysobjects*. That is why we at present do not allow scalable multi-database views. The client searches, recursively if necessary, for any dependants of *X* that is a view that has a table as dependant in *sysobjects* or has no dependant listed there. The former may be an image with a local segment. The latter may be an image with remote segments only. It then searches *Image* again for *X*. If it finds it, then it attempts to add it to *check_image*.

Once all the images have been determined, i.e., there is no *FROM* clause in the query remaining for the analysis, the client verifies each of them, as usual. The verification follows the order on the image names. The rationale is to avoid the (rare) deadlock, when two outdated images could be concurrently processed in opposite order by two queries to the same manager. The adjustment generates indeed an exclusive lock on the tuple in *Image*. After the end of the image binding phase, the client continues with the specific processing of each command that we present later.

With respect to the concurrent command processing, the image binding phase results for SQL Server in a repeatable read level distributed transaction with shared locks or exclusive locks on the tuples of the bound images in *Image* tables and with the shared locks on all the related tuples in various *RP* tables. The image binding for one query may thus block another binding the same image that happened to be outdated. A shared lock on *RP* tuple may block a concurrent split as already discussed. We’ll show progressively that all this behaviour contributed to the serializability of the whole scalable command processing under SD-SQL Server.

5 Performance Analysis

To validate the SD-SQL Server architecture, we evaluated its scalability and efficiency over some Skyserver DB data [2]. Our hardware consisted of 1.8 GHz P4 PCs with either 785 MB or 1 GB of RAM, linked by a 1 Gbs Ethernet. We used the SQL Profiler to take measurements. We measured split times and query response times under various conditions. The split time was from about 2.5 s for a 2-split of 1000-tuple *PhotoObj* segment, up to 150 seconds for a 5-split of a 160 000 tuple segment. Presence of indexes naturally increased this time, up to almost 200 s for the 5-split of a 160 000 tuple segment with three indexes.

The query measures included the overhead of the image checking alone, of image adjustment and of image binding for various queries, [14]. Here, we discuss two queries:

(Q1) *sd_select 'COUNT (*) FROM PhotoObj'*

(Q2) s*d_select 'top 10000 x.objid from photoobj x, photoobj y where x.obj=y.obj and x.objid>y.objid*

Query (Q1) represents the rather fast queries, query (Q2) the complex ones, because of its double join and larger result set. The measures of (Q1) concern the execution time under various conditions, Fig. 2. The table had two segments of various sizes. We measured (Q1) at SD-SQL Server peer, where the *Image* and *RP* tables are at the same node, and at a client where they are at different nodes. We executed (Q1) with image checking (IC) only, and with the image adjustment (IA). We finally compared these times to those of executing (Q1) as an SQL Server, i.e., without even IC. As the curves show, IC overhead appeared always negligible. (Q1) always executed then under 300 msec. In contrast the IA overhead is relatively costly. On a peer node it is about 0.5s, leading to the response time of 0.8sec. On a client node, it increases to about 1s, leading to (Q1) response time of 1.5s. The difference is obviously due to the remote accesses to *RP* table. Notice that IA time is constant, as one could expect. We used the LSV option, but the results were about the same without it.

The measures of (Q2) representing basically longer to process typical queries than (Q1), involved *PhotoObj* with almost 160 K tuples spreading over two, three or four segments. The query execution time with IC only (or even without, directly by SQL Server) is now between 10 – 12 s. The IA overhead is about or little over 1 s. It grows a little since SQL Server ALTER VIEW operation has more remote segments to access for the check constraint update (even if it remains in fact the same, which indicates a clear path for some optimizing of this operation under SQL Server). IA overhead becomes relatively negligible, about 10 % of the query cost. We recall that IA should be in any case a rare operation.

Finally, we have measured again (Q1) on our *PhotoObj* scalable table as it grows under inserts. It had successively 2, 3, 4 and 5 segments, generated each by a 2-split. The query counted at every segment. The segment capacity was 30K tuples. We aimed at the comparison of the response time for an SD-SQL Server user and for the one of SQL Server. We supposed that the latter (i) does not enters the manual repartitioning hassle, or, alternatively, (ii) enters it by 2-splitting manually any time the table gets new 30K tuples, i.e., at the same time when SD-SQL Server would trigger its split. Case (i) corresponds to the same comfort as that of an SD-SQL Server

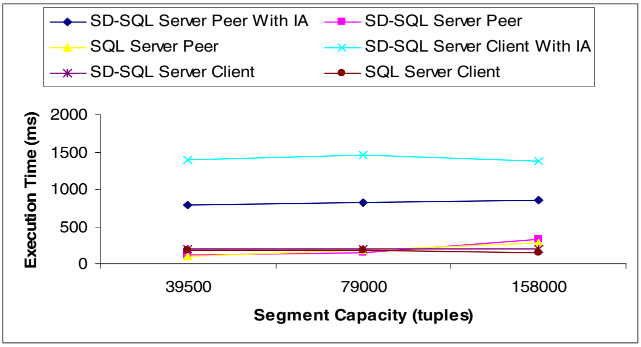


Fig. 2. Query (Q1) execution performance

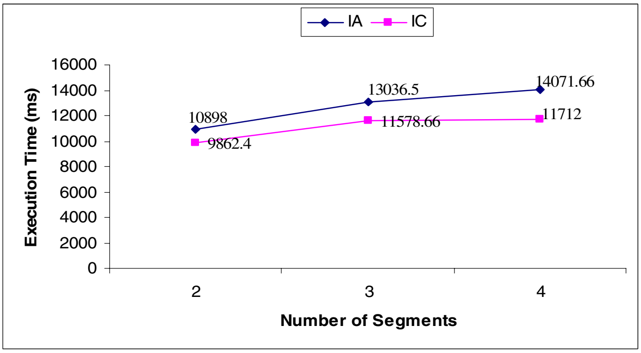


Fig. 3. Query (Q2) with image checking only (IC) and with image adjustment (IA)

user. The obvious price to pay for an SQL Server user is the scalability, i.e., the worst deterioration of the response time for a growing table. In both cases (i) and (ii) we studied the SQL Server query corresponding to (Q1) for a static table. For SD-SQL Server, we measured (Q1) with and without the LSV option.

The figure displays the result. The curve named “SQL Server Centr.” shows the case (i), i.e., of the centralized *PhotoObj*. The curve “SQL Server Distr.” reflects the manual reorganizing (ii). The curve shows the minimum that SD-SQL Server could reach, i.e., if it had zero overhead. The two other curves correspond to SD-SQL Server.

We can see that SD-SQL Server processing time is always quite close to that of (ii) by SQL Server. Our query-processing overhead appears only about 5%. We can also see that for the same comfort of use, i.e., with respect to case (i), SD-SQL Server without LZV speeds up the execution by almost 30 %, e.g., about 100 msec for the largest table measured. With LZV the time decreases there to 220 msec. It improves thus by almost 50 %. This factor characterizes most of the other sizes as well. All these results prove the immediate utility of our system.

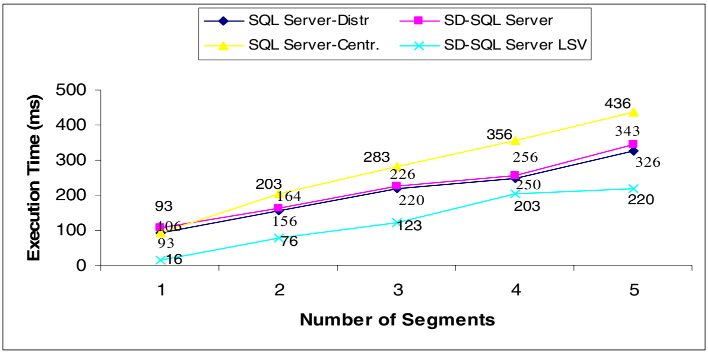


Fig. 4. Query (Q1) execution on SQL Server and SD-SQL Server (Client/Peer)

Notice further that in theory SD-SQL Server execution time could remain constant and close to that of a query to a single segment of about 30 K tuples. This is 93 ms in our case. The timing observed practice grows in contrast, already for the SQL Server. The result seems to indicate that the parallel processing of the aggregate functions by SQL Server has still room for improvement. This would further increase the superiority of SD-SQL Server for the same user's comfort.

6 Related Works

Parallel and distributed database partitioning has been studied for many years, [13]. It naturally triggered the work on the reorganizing of the partitioning, with notable results as early as in 1996, [12]. The common goal was global reorganization, unlike for our system.

The editors of [12] contributed themselves with two on-line reorganization methods, called respectively *new-space* and *in-place* reorganization. The former method created a new disk structure, and switches the processing to it. The latter approach balanced the data among existing disk pages as long as there was room for the data. Among the other contributors to [12], one concerned a command named '*Move Partition Boundary*' for Tandem Non Stop SQL/MP. The command aimed on on-line changes to the adjacent database partitions. The new boundary should decrease the load of any nearly full partition, by assigning some tuples into a less loaded one. The command was intended as a manual operation. We could not ascertain whether it was ever implemented.

A more recent proposal of efficient global reorganizing strategy is in [11]. One proposes there an automatic advisor, balancing the overall database load through the periodic reorganizing. The advisor is intended as a DB2 offline utility. Another attempt, in [4], the most recent one to our knowledge, describes yet another sophisticated reorganizing technique, based on database clustering. Called AutoClust, the technique mines for closed sets, then groups the records according to the resulting attribute clusters. AutoClust processing should start when the average query response time drops below a user defined threshold. We do not know whether AutoClust was put into practice.

With respect to the partitioning algorithms used in other major DBMSs, parallel DB2 uses (static) hash partitioning. Oracle offers both hash and range partitioning, but over the shared disk multiprocessor architecture only. Only SQL Server offers the updatable distributed partitioned views. This was the major rationale for our choice, since scalable tables have to be updatable. How the scalable tables may be created at other systems remains thus an open research problem.

7 Conclusion

The proposed syntax and semantics of SD-SQL Server commands make the use of scalable tables about as simple as that of the static ones. It lets the user/application to easily take advantage of the new capabilities of our system. Through the scalable distributed partitioning, they should allow for much larger tables or for a faster response time of complex queries, or for both.

The current design of our interface is geared towards a “proof of concept” prototype. It is naturally simpler than a full-scale system. Further work should expand it. Among the challenges at the processing level, notice that there is no user account management for the scalable tables at present. Concurrent query processing could be perhaps made faster during splitting. We tried to limit the use of exclusive locks to as little as necessary for correctness, but there is perhaps still a better way. Our performance analysis should be expanded, uncovering perhaps further directions for our current processing optimization. Next, while SD-SQL Server acts at present as an application of SQL Server, the scalable table management could alternatively enter the SQL Server core code. Obviously we could not do it, but the owner of this DBS can. Our design could apply almost as is to other DBSs, once they offer the updatable distributed partitioned (union-all) views. Next, we did not address the issue of the reliability of the scalable tables. More generally, there is a security issue for the scalable tables, as the tuples migrate to places unknown to their owners.

Acknowledgments. We thank J.Gray (Microsoft BARC) for the original SkyServer database and for advising this work from its start. G. Graefe (Microsoft) provided the helpful information on SQL Server linked servers’ capabilities. MS Research partly funded this work, relaying the support of CEE project ICONS. Current support comes from CEE Project EGov.

References

1. Ben-Gan, I., and Moreau, T. Advanced Transact SQL for SQL Server 2000. Apress Editors, 2000
2. Gray, J. & al. Data Mining of SDDS SkyServer Database. WDAS 2002, Paris, Carleton Scientific (publ.)
3. Gray, J. The Cost of Messages. Proceeding of Principles Of Distributed Systems, Toronto, Canada, 1989
4. Guinepain, S & Gruenwald, L. Research Issues in Automatic Database Clustering. ACM-SIGMOD, March 2005
5. Lejeune, H. Technical Comparison of Oracle vs. SQL Server 2000: Focus on Performance, December 2003

6. Litwin, W., Neimat, M.-A., Schneider, D. LH*: A Scalable Distributed Data Structure. ACM-TODS, Dec. 1996
7. Litwin, W., Neimat, M.-A., Schneider, D. Linear Hashing for Distributed Files. ACM-SIGMOD International Conference on Management of Data, 1993
8. Litwin, W., Rich, T. and Schwarz, Th. Architecture for a scalable Distributed DBSs application to SQL Server 2000. 2nd Intl. Workshop on Cooperative Internet Computing (CIC 2002), August 2002, Hong Kong
9. Litwin, W & Sahri, S. Implementing SD-SQL Server: a Scalable Distributed Database System. Intl. Workshop on Distributed Data and Structures, WDAS 2004, Lausanne, Carleton Scientific (publ.), to app
10. Microsoft SQL Server 2000: SQL Server Books Online
11. Rao, J., Zhang, C., Lohman, G. and Megiddo, N. Automating Physical Database Design inParallel Database, ACM SIGMOD '2002 June 4-6, USA
12. Salzberg, B & Lomet, D. Special Issue on Online Reorganization, Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 1996
13. Özsu, T & Valduriez, P. Principles of Distributed Database Systems, 2nd edition, Prentice Hall, 1999.
14. Litwin, W., Sahri, S., Schwarz, Th. SD-SQL Server: a Scalable Distributed Database System. CERIA Research Report 2005-12-13, December 2005.
15. Litwin, W., Sahri, S., Schwarz, Th. Architecture and Interface of Scalable Distributed Database System SD-SQL Server. The Intl. Ass. of Science and Technology for Development Conf. on Databases and Applications, IASTED-DBA 2006, to appear.

Using UML's Sequence Diagrams for Representing Execution Models Associated to Triggers*

Harith T. Al-Jumaily, César de Pablo, Dolores Cuadra, and Paloma Martínez

Computer Science Department, Universidad Carlos III de Madrid
{haljumai, cdepablo, dcuadra, pmf}@inf.uc3m.es

Abstract. Using active rules or triggers to verify integrity constraints is a serious and complex problem because these mechanisms have behaviour that could be difficult to predict in a complex database. The situation is even worse as there are few tools available for developing and verifying them. We believe that automatic support for trigger development and verification would help database developers to adopt triggers in the database design process. Therefore, in this work we suggest a visualization add-in tool that represents and verifies triggers execution by using UML's sequence diagrams. This tool is added in RATIONAL ROSE and it simulates the execution sequence of a set of triggers when a DML operation is produced. This tool uses the SQL standard to express the triggers semantics and execution.

1 Introduction

The passive behaviour of traditional databases often causes semantic loss in database systems. Users and applications always have the responsibility to protect these semantics. For this reason, traditional databases have been improved by adopting active behaviour. An active behaviour is a complex operation that is activated in an autonomous way to perform predefined actions. Usually, this behaviour is known as triggers or ECA rules. An ECA rule consists of three components; event, condition, and action. The execution model of ECA rules follows a sequence of steps: event detection, condition test, and action execution. An event in relational databases is a DML (Data Manipulation Language) statement such as (INSERT, DELETE, and UPDATE). Once a trigger is activated and its condition is evaluated to true, the predefined actions are automatically executed.

Incorporating active rules enhances the functionality of database systems and provides flexible alternatives to implement many database features, such as to enforce integrity constraints [1]. Because of execution models of triggers, an active database is more complicated than a passive one. For that reason we believe that automatic support for triggers development could help to adopt active rules by database designers and developers. The validation of active rules/triggers execution is the major problem that makes the application development a difficult task. As rules can act in a way that leads to conflict and undesirable problems, the developer needs

* This work is part of the project *"Software Process Management Platform: modelling, reuse and measurement"*. TIN2004/07083.

D. Bell and J. Hong (Eds.): BNCOD 2006, LNCS 4042, pp. 36 – 46, 2006.

additional effort to control this behaviour. The objective of this validation is to guarantee the successful execution of the triggers; that means to avoid non-termination state in their execution.

In commercial CASE tools which support triggers development, we have detected that developing triggers and plugging them into a given model is an insufficient task because of the behaviour of such triggers is invisible to the developers.

Therefore, in this work we suggest a visualization tool to represent and verify triggers execution by using UML's sequence diagrams. This tool has three contributions. First, we use the SQL standard [5] to express triggers semantics and execution. Second, we use the UML sequence diagram to display interactions between triggers. And finally, we use a commercial CASE tool (Rational Rose) to add-in our approach. These contributions make our approach quite useful, practical and intuitive to manage triggers using the triggering graph (TG) for checking non termination state. TG is one of the most important tools in active rules to check the termination execution for a set of rules and we adopt it for our approach.

The rest of this work is organised as follows. In section (2) the semantics of triggers execution is explained according to the SQL standard. In section (3) works related to rules behaviour analyzer tools or visualization tools are presented. In section (4) we will explain our visualization tool design. Finally, in section (5) some conclusions and future works are exposed.

2 Triggers Execution in SQl3

This section addresses common components according to the definition specified in the SQL2003 standard which makes revisions to all parts of SQL99 and adds new features [4]. A SQL standard trigger is a named event-condition-action rule that is activated by a database state transition. Every trigger is associated with a table and is activated whenever that table is modified. Once a trigger is activated and its condition evaluated to true, the trigger's action is performed. When we talk about the semantics of triggers execution in the SQL standard, we consider the Knowledge Model and the Execution Model.

A knowledge model supports the description of the active functionality; it is considered to have three principal components; an event, a condition, and an action [1]. The SQL3 syntax of triggers is shown in Fig.1.

In database systems, the triggers execution model specifies how a set of triggers is treated and executed at runtime. Although triggers are available in most DBMS, unfortunately their execution models change from one DBMS to another. Despite this fact, a common execution strategy is shared among systems according to the two main requirements for the SQL triggers execution. These requirements are [1]; (1) the execution model must ensure consistency in the database, and (2) all Before-triggers and After-triggers must be execute before or after the associated table will be modified, respectively. Before-triggers are especially useful for maintaining a constraint at the time that the data changes, while After-triggers are useful for invoking functions and stored procedures which execute modification operations inside and outside the database.

```
CREATE TRIGGER <trigger name>
[BEFORE | AFTER]
[INSERT | DELETE | UPDATE [OF <trigger column name>]]
ON <table name>
[REFERENCING <Old | New [Row | Table]>]
[FOR EACH {ROW | STATEMENT}]
[WHEN < condition >]
BEGIN ATOMIC
{< SQL procedure statement…… >}...
END;
```

Fig. 1. The SQL standard triggers syntax

The SQL standard allows the definition of multiple triggers associated to the same table, same event, and same activation time. Multiple triggers can simultaneously be selected for the execution. When multiple triggers are activated at the same time, priorities are used to resolve triggers executions. A trigger with the highest priority is executed first [5].

3 Related Work

As many works have been done in the area of static analysis [17] [18], we use in our analyzer the concept of Triggering Graph (TG) to detect non-termination states. TG is a straightforward graph where each node T_i corresponds to a rule and a direct arc between T_1 and T_2 is the event which belongs to T_1‘s action and causes the activation of T_2. A cycle is produced in TG when a rule T_i may trigger itself or when T_i triggers the same initial subset. In the figure (2), the subset of active rules S={T_1, T_2, T_3} is a cycle when the rule T_1 is fired again by the event e_3. The termination analysis itself focuses on identifying and eliminating arcs that could introduce cycles into the TG [19]. Redefining the rule T_3 and reconstructing again the graph TG is a good solution to verify the termination state of the subset S.

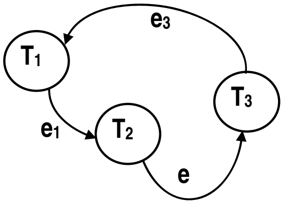


Fig. 2. The TG of the rules execution

Rules behaviour analyzer tools or visualization tools have been received strong interest from active database community where various works have been mentioned in the literature on using these tools in the verification of triggers execution. Arachne [10] is one of such tools; it is used in the context of object oriented (OO) active database systems. It accepts as input a set of Chimera active rules, and it uses triggering graph analysis to detect non-termination behaviour at compile-time. Active rules terminate if the triggering graph is acyclic. Once a cycle is detected, the user is responsible for assuring the termination. VITAL [11] is another set of development tools; it includes a tool for static analysis. This tool uses the triggering graph for detecting cycles in a set of active rules. TriGS [12] is a graphical development tool of active OO database application, it has been specifically designed for Trigger system for GemSione. DEAR [13] tool has been implemented on an active OO database system. It mainly focuses on a display form to rules interaction and sends information to users to help them to detect errors.

On the other hand, multiple efforts have been devoted to face database modelling problems. One of these problems is the automatization of database design process using CASE tools. Frequently, these tools do not completely support all phases of analysis and design methodology of databases. Triggers development is supported by some of these CASE tools such as Rational Rose [14], ERwin [15], Designer2000 [16]. These tools provide editors to allow users define different types of triggers. Furthermore, ERwin allows users generate triggers which enforce integrity constraints. The drawback of these CASE tools is that the termination of triggers execution is not guaranteed. Until now, there is no way to allow users of these tools to verify the developed triggers without need to execute them in a real database.

4 Visualization Tool Design

For explaining our approach, let us consider the example shown in figure 3 (a). It is a simple database schema using UML class diagram. It has three persistent classes (Student, Professor, and Department). The mapping of each class and association into Rational Rose Data model [20] [21] is shown in figure 3(b), and the data model transformation into relational model is shown in figure 3(a).

The main objective of our tool is to display triggers execution scenarios and to send messages to users for indicating whether these scenarios terminate or it is necessary the users' intervention to resolve a non termination execution. In this proposal, we show triggers and integrity constraints interaction in a display form as well as the non termination problem.

On the other hand, the UML's sequence diagram is used to show the interactions between objects and events in a sequential order according to the time. It is a two-dimension diagram, the vertical dimension is the time axis, and the horizontal dimension shows objects roles in their interactions.

In the context of triggers execution, it is very helpful to employ a tool to show the behaviour and the interactions of triggers that belong to a model. Therefore, we will use the sequence diagram elements to interpret the execution of triggers associated to

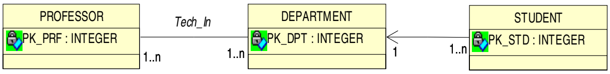


Fig. 3(a). Class diagram

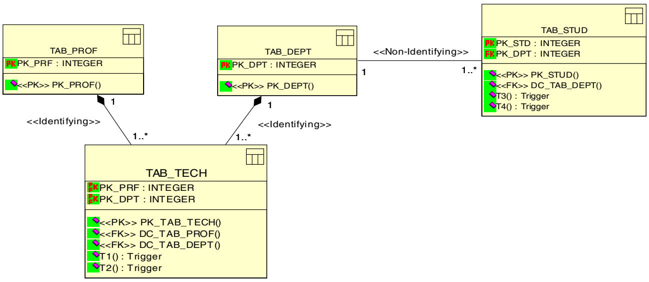


Fig. 3(b). Transformation of (a) into Rational Rose Data model

```
Create Table TAB_DEPT (PK_DPT Primary Key …);
Create Table TAB_STUD (PK_STD Primary Key, PK_DPT,
Constraint DC_TAB_DEPT References TAB_DEPT(PK_DPT)
On Delete Cascade);
Create Table TAB_PROF (PK_PRF Primary Key …);
Create Table TAB_TECH (PK_PRF, PK_DPT,
Constraint DC_TAB_PROF FOREIGN KEY (PK_PRF)
References TAB_PROF(PK_PRF) On Delete Cascade,
Constraint DC_TAB_DEPT FOREIGN KEY (PK_DPT)
References TAB_DEPT(PK_DPT) On Delete Cascade);
```

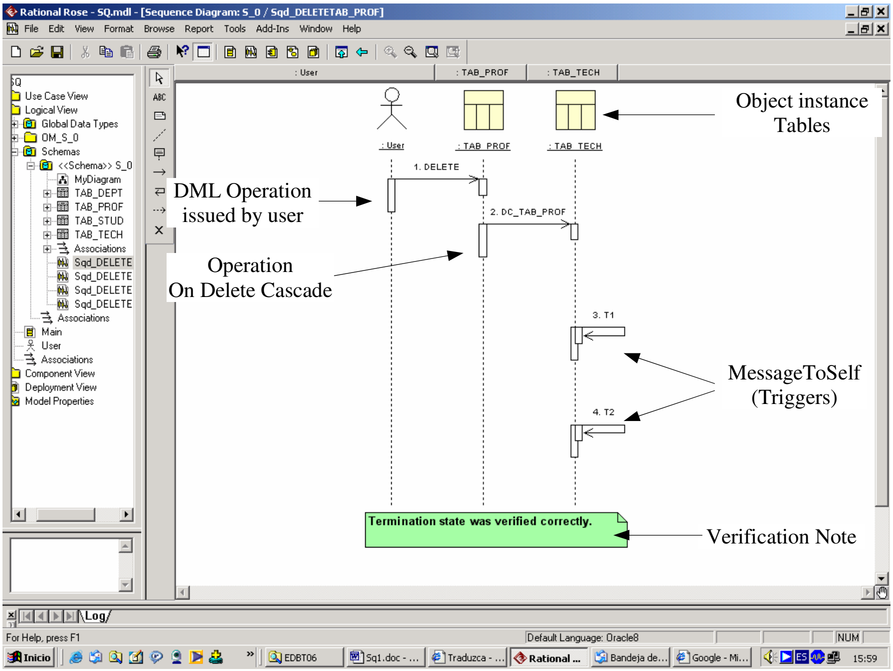
Fig. 3(c). Transformation of (b) into relational model

a relational schema. We use Rational Rose CASE tool to implement our approach because it is able to easily add-in software tools. It can be accessed from the Tools menu.

4.1 Used UML Notation

In this section, we will explain how we use the UML notation [22] to represent triggers execution and how we apply sequence diagrams to detect the non termination problem. The figure (4) shows an example of a sequence diagram.

- **Scenario Diagram:** A scenario is an instance of a use case that describes the sequential occurrences of events during the system execution. Sequence diagrams



Termination state was verified correctly.

Fig. 4. A Scenario Diagram (Rational Rose)

allow users to create a display form of a scenario. In our approach, we create a scenario diagram for each event that may be generated on a table and the sequence of events and operations that follow after that event. Therefore, for each object table in the model, we need to create three scenario diagrams, one for each DML statement (INSERT, DELETE, and UPDATE).

- **Tables:** Tables are represented in Rational Rose as a stereotype of an object instance. The scenario diagram contains one or more object instances which have behaviour to be shown in the diagram. A table has three basic behaviours relevant for static analysis of the termination which are the three DML operations (INSERT, UPDATE, and DELETE). An object instance has a lifeline which represents the existence of the object over a period of time.
- **Message:** Messages in a sequence diagram are methods or operations which are used to illustrate the object behaviour. A message is the communication carried between two objects to define the interaction between them. A message is represented in the sequence diagram by using the message icon connecting two lifelines together. The message icons appear as solid arrows with a sequence number and a message label. The first message always starts at the top of the diagram and others messages follow it. When theSender=theReceiver, this means that the object theSender is sending a message to itself, MessageToSelf. Each message is associated with an integer number that shows the relative position of the message in the diagram. For example, if theSequence=3, the message is the third message in the diagram.

- **Note:** We use notes to warn users about the results of the verification. Our tool represents two types of notes to the users. The first is "*Termination state was correctly verified*" which is sent when the execution of a given scenario is correctly terminated. The second note is "*Non termination state was detected. Please, solve the problem and try again*". This note is sent when the verification of the scenario detects a non termination state in the execution of triggers.

4.2 Applying Sequence Diagrams

In general, triggers which are associated to a table are activated when that table is modified. In this context, when a trigger is fired it must examine the associated table and all other tables that can be modified by it. When an activated trigger examines its associated table, this is exactly like when an object sends a message to itself theSender=theReceiver. Therefore, a trigger instance is represented in sequence diagrams as MessageToSelf (figure 4). The trigger name is included into the message icon. BEFORE-triggers and AFTER-triggers are represented by using the same notation MessageToSelf. Trigger conditions are not considered because we use the static analysis approach to detect non termination state.

On the other hand, we used the notation Message for the other operations related to the behaviour. Such operations are shown below:

- The first operation that starts the scenario diagram. This message represents the operation which is sent from the user to a given object to start the scenario.
- DML statements (INSERT, DELETE, and UPDATE) included in a trigger's action and modify other object table theSender≠theReceiver.
- Referential constraint actions, CASCADE, SET DEFULT, and SET NULL are considered. We represent these actions in the sequence diagrams as messages from the parent object to the child object. The name and the type of the operation are indicated on the message icon.

The message icon used to represent these operations is a solid horizontal arrow with a sequence number and a message label.

The most important aspects that distinguish the SQL standard trigger execution model from others models such as, (Ariel [6], HiPac [7], and Starburst [8]) are the interactions between the triggers and the referential constraint actions [9]. In relational databases, the tables are represented by sets of rows and connections between tables are represented by defining foreign keys. Referential integrity constraints are predicates on a database state that must be evaluated, if these restrictions are violated the database is in an inconsistent state. In order to maintain the referential integrity of the database, the SQL standard uses actions such as NO ACTION, RESTRICT, CASCADE, SET DEFAULT, and SET NULL. In this work, we consider the last three actions because they produce interactions among triggers.

We will present in this section two scenarios to illustrate the usefulness of our tool.

Scenario 1

Let us consider that the table TAB_TECH has two triggers (figure 3(b)). The descriptions of these triggers are shown as follows:

```
CREATE TRIGGER T1              CREATE TRIGGER T2
AFTER DELETE ON TAB_TECH       AFTER DELETE ON TAB_TECH
WHEN C1                        WHEN C2
BEGIN ATOMIC                   BEGIN ATOMIC
X:=1;                          Y:=X;
END;                           END;
```

The scenario 1 (figure 4) starts when the user **Actor** carries out the operation (1: DELETE) in the table TAB_PROF. This table does not have any associated trigger, but when this operation is carried out, the referential action On Delete Cascade (2:DC_TAB_PROF) in TAB_TECH is executed; then the two triggers (3: T1) and (4: T2) are executed as well. As figure (4) shows, when the execution of T2 is finished the transaction is ended, so termination state is reached. The execution sequence of this scenario is shown below:

1: When DELETE FROM TAB_PROF is carried out ⇒
2: The DC_TAB_PROF is executed (section 2.3) ⇒
3: T1 (X=1) is executed ⇒
4: T2 (Y=X) is executed ⇒ END. "*Termination state was correctly verified*"

Scenario 2
The new scenario illustrates the non termination state (figure 5). We will redefine the body of the trigger T2 incorporating in its action a delete operation from TAB_PROF. As is shown below:

```
CREATE TRIGGER T2
AFTER DELETE ON TAB_TECH
WHEN C2
BEGIN ATOMIC
DELETE FROM TAB_PROF WHERE ……..;
END;
```

Now, if we examine this scenario, the operations sequence is similar to the previous scenario until the execution reaches the message (4: T2). In this time, the new statement incorporated into T2 is carried out after its execution. This operation (5: DELETE) generates the referential action execution (6:DC_TAB_PROF). As consequence, the last trigger operation (7: T1) is fired again. When a trigger is fired twice in the same scenario this means that the non termination state is detected. Therefore, the scenario is stopped and a message is sent to the developer which must resolve the problem and repeat the scenario again. The execution sequence of this scenario is shown below:

1: When DELETE FROM TAB_PROF is carried out ⇒

2: The DC_TAB_PROF is executed (section 2.3) ⇒

3: T1 (X=1) is executed ⇒

4: T2 is executed ⇒

5: DELETE FROM TAB_PROF is executed ⇒

6: Again 2 ⇒

7: Again 3 ⇒ ; STOP; *"Non termination state was detected. Please, solve the problem and try again"*

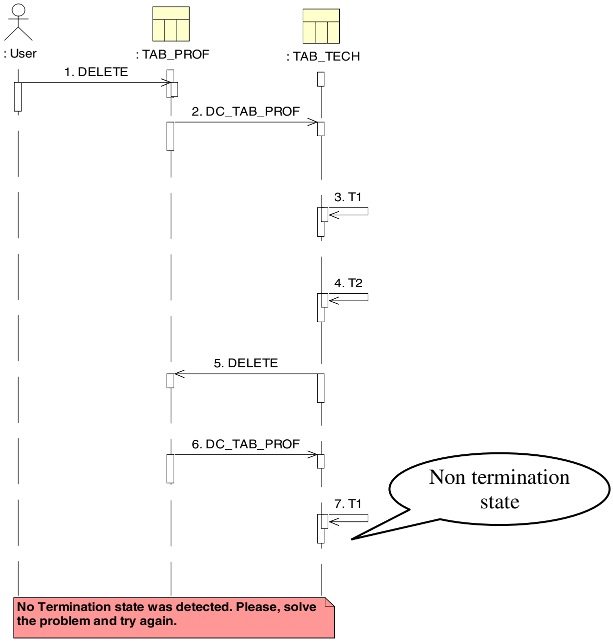


Fig. 3. Scenario 2, non termination state

5 Conclusions

Active rules/triggers systems are exposed in many studies and some challenges and issues are addressed to control the execution of these systems. One of these challenges is to encourage commercial CASE tools will cover all analysis phases with extended conceptual models.

Using triggers means additional effort in database development because the triggers execution model adds more complexity. We use UML's sequence diagrams to represent the triggers execution flow in order to verify the triggers behaviour and to guarantee the correct execution in Rational Rose Tool. Our principal objective in this work is to motivate database designers to use triggers for completing semantic specifications gathered in a conceptual schema through a CASE tool which shows triggers execution associated to a relational schema in an intuitive way.

As future work, we will extend our approach to include the confluence problem of triggers execution, and we will aggregate this tool into our toolbox in order to get to transform integrity constraints of a given schema into triggers. Furthermore, we are going to design some experiments to validate our tool and the efficiency use according to our contributions: to make easy the complex problem of triggers implementation and to check triggers execution.

References

1. Paton W. N., "Active Rules in Database Systems", Springer-Verlag, New York, 1998.
2. Norman W. P., Diaz O., "Active Database Systems", ACM Computing Surveys, Vol.31, No.1, 1999.
3. Ceri S., Cochrane R. J., Widom J., "Practical Applications of Triggers and Constraints: Successes and Lingering Issues". Proc. of the 26th VLDB Conf., Cairo, Egypt, 2000.
4. A. Eisenberg, J. Melton, K. Kulkarni, J. Michels, F. Zemke, "SQL:2003 has been published", ACM SIGMOD Record, Volume 33 , Issue 1, March 2004.
5. Melton J., Simon A. R.. "SQL: 1999 Understanding Relational Language Components", Morgan Kaufmann Publishers, 2002.
6. Hanson E. N., "The Design and Implementations of Ariel Active Database Rule System", IEEE Transactions on Knowledge and Data Engineering, Vol. 8, No.1, February 1996.
7. Dayal U., Buchmann A. P., Chakravarthy S., "The HiPAC Project" in Active database systems: triggers and rules for advanced database processing, Widom J., Ceri S., Eds. San Francisco, CA.: Morgan Kaufmann Publishers, 1996, pp. 177-205.
8. Widom J., Cochrane R. J., Lindsay B. G. "Implementing set-oriented production rules as an extension to Starburst". Proc. 7^{th} International Conference on VLDB, September 1991.
9. Kulkarni K., Mattos N., Cochrane R., "Active Database Features in SQL3", Active Rules in Database Systems", Springer-Verlag, New York, 1998. pp 197-218.
10. Ceri S., Fraternalli, P., "Designing database applications with objects and rules:the IDEA Methodology". Addsion-Wesley".1997.
11. E. Benazet, H. Guehl, and M. Bouzeghoub. "VITAL a visual tool for analysis of rules behaviour in active databases", Proc of the 2nd Int. Workshop on Rules in Database Systems. Pages 182-196, Greece 1995.
12. 12. G. Kappel, G. Kramler, W. Retschitzegger. "TriGS Debugger A Tool for Debugging Active Database Behavior", Proceedings of the 12th International Conference on Database and Expert Systems Applications, Springer-Verlag London, UK , 2001
13. O. Díaz, A. Jaime, N. Paton. "DEAR a DEbugger for Active Rules in an object-oriented context". In M. Williams, N. Paton. Rules In Database Systems. Pages 180-193, LNCS Springer Verlag 1993.
14. Rational Web site http://www.rational.com/support/documentation/
15. AllFusion® ERwin® Data Modeler site http://www3.ca.com/Solutions/
16. ORACLE Web site http://www.oracle.com/technology/products/
17. Alexander A., Jennifer W., "Behavior of Database Production Rules: Termination, Confluence, and Observable Determinism", Proc. ACM-SIGMOD Conf. 1992.

18. Paton N., Díaz O., "Active Database Systems", ACM Computing Surveys, Vol.31, No.1, 1999.
19. Hickey T., “Constraint-Based Termination Analysis for Cyclic Active Database Rules”. Proc. DOOD'2000: 6th. International Conference on Rules and Objects in Databases, Springer LNAI vol. 1861, July 2000, pp. 1121-1136.
20. Vadaparty, Kumar, “ODBMS - Bridging the Gap Between Objects and Tables: Object and Data Models”, volume 12 - issue 2, 1999.
21. Timo Salo, Justin hill, “Mapping Objects to Relational Databases”, Journal of Object Oriented Programming, volume 13 - issue 1, 2000.
22. UML 2 Sequence Diagram Overview http://www.agilemodeling.com/artifacts/sequenceDiagram.htm

An Experimental Consideration of the Use of the Transrelational^{TM} Model for Data Warehousing*

Victor Gonzalez-Castro[1], Lachlan M. MacKinnon[2], and David H. Marwick[1]

[1] School of Mathematical and Computer Sciences, Heriot-Watt University,
Edinburgh, EH14 4AS, Scotland
{victor, dhm}@macs.hw.ac.uk
[2] School of Computing and Creative Technologies, University of Abertay Dundee,
Dundee, DD1 1HG, Scotland
l.mackinnon@abertay.ac.uk

Abstract. In recent years there has been a growing interest in the research community in the utilisation of alternative data models that abandon the relational record storage and manipulation structure. The authors have already reported experimental considerations of the behavior of Relational, Binary Relational and Associative models within the context of Data Warehousing, to address issues of storage efficiency and combinatorial explosion through data repetition. In this paper we present an implementation of the Transrelational^{TM} model, based on the public domain definition provided by C.J. Date, which we believe to be the first reported instantiation of the model. Following the presentation of the implementation, we also present the results of performance tests utilising a set of metrics for Data Warehouse environments, which are compared against a traditional N-ary Relational implementation. The experiment is based on the standard and widely-accepted TPC-H data set.

1 Introduction

The Transrelational^{TM} model was defined by S. Tarin and patented in the United States of America [14], and it has been promoted to the Database community by C.J. Date through a series of seminars and in the latest edition of his widely-adopted textbook [3]. However, as far as we can determine there is no implementation available for either commercial or research use. Therefore, in order to carry out our experimental consideration, we have utilised the general description made by Date [3] of the Transrelational^{TM} model and its behavior to implement the essential algorithms that make up the model. Since Date [3] has provided the only public domain documentation of the model, which we shall henceforward refer to as TR following his nomenclature, we shall make reference extensively to his work in describing our experiment.

Our experimental consideration of the TR model follows on from research which we have already reported considering the performance of Relational, Binary Relational and Associative models in the context of Data Warehousing [4][5][8]. We

[*] The Transrelational model is based on the Tarin Transform Method and is the intellectual property of Required Technologies Inc.

D. Bell and J. Hong (Eds.): BNCOD 2006, LNCS 4042, pp. 47 – 58, 2006.

are interested in the use of alternative data models that can solve the problems of the inefficiency of storage caused by models based on rows, which include repetitions at field level no matter if they use normalized (snow-flake) or non-normalized (star) schemas, as well as the database explosion phenomenon [7] that occurs in Relational Data Warehouses.

According to Date [3:954], the TR model has an implicit data compression mechanism by using *condensed columns* which eliminates the duplicate values at a column level. This is very appealing in Data Warehouse environments where it is common to have many repetitive values at column level, so we wanted to measure and benchmark this characteristic of the model.

2 The Transrelational Model^{TM}

As already indicated the only public domain description of the Transrelational^{TM} model is provided by Date [3]. In this section we introduce some of the basic definitions of the model in order to establish a baseline for our experimental consideration, all quotes and references to Date are attributable to [3] and where necessary page number references are supplied. As a starting point Date states that, *"The Transrelational^{TM} (TR) is not intended as a replacement for the Relational model"*, from which we can infer that it can be seen as an alternative route to implement the relational model and thus to build a Relational DBMS.

From the early days of data processing systems, through the development of relational databases and up to the present day, data has predominantly been conceived as Records of *n* number of fields or attributes. This approach has been called an *N-ary Storage Model* (NSM) [2] or in Date's nomenclature *Direct Image Systems* (DIS). Within this approach data is seen (and stored) following a horizontal approach (rows).

Alternatively, there is also a vertical approach (columns) to store and process data, and this has its origins in Copeland's seminal paper *"A Decomposition Storage Model"* (DSM) [2]. This has recently been used as the basis for the creation of some novel database architectures and instantiations, such as MonetDB [1], [6], SybaseIQ [11], [12] and C-store [10]. TR differs from both vertical and horizontal approaches, but is closer to a vertical approach since, in the *Field Values Table* (FVT), each column stores distinct values, and most of the processing can be done at column level. This is analysed in more detail in section 3.

2.1 Model Data Structures

To illustrate the characteristics of the model, we utilise the examples developed by Date [3]. The TR model consists basically of two storage structures: the **Field Values Table (FVT),** where each column contains the values from the corresponding field of the input file rearranged in an ascending sort order **Fig. 1(b);** and the **Record Reconstruction Table (RRT) Fig. 1(c),** which keeps a set of values derived from the original input file that can be thought of as pointers to enable the original record to be rebuilt when necessary using the **ZigZag algorithm** [3:948].

Record Sequence	S#	SNAME	STATUS	CITY
1	S4	Clark	20	London
2	S5	Adams	30	Athens
3	S2	Jones	10	Paris
4	S1	Smith	20	London
5	S3	Blake	30	Paris

	S#	SNAME	STATUS	CITY
1	S1	Adams	10	Athens
2	S2	Blake	20	London
3	S3	Clark	20	London
4	S4	Jones	30	Paris
5	S5	Smith	30	Paris

	S#	SNAME	STATUS	CITY
1	5	4	4	5
2	4	5	2	4
3	2	2	3	1
4	3	1	1	2
5	1	3	5	3

Fig. 1. (a) A Suppliers relation, (b) Field Values Table, (c) Record Reconst. Table

To understand how both tables are used when rebuilding a record, utilising the ZigZag algorithm, we provide Date's description:

> *"Step 1:* Go to cell [1, 1] of the Field Values Table and fetch the value stored there: namely, the supplier number S1. That value is the first field value (that is. the S# field value) within a certain supplier record in the suppliers file.
>
> *Step 2:* Go to the same cell (that is, cell [1, 1]) of the Record Reconstruction Table and fetch the value stored there: namely, the row number 5. That row number is interpreted to mean that the next field value (which is to say, the second or SNAME value) within the supplier record whose S# field value is S1 is to be found in the SNAME position of the fifth row of the Field Values Table -in other words, in cell (5,2) of the Field Values Table. Go to that cell and fetch the value stored there (supplier name Smith).
>
> *Step 3:* Go to the corresponding Record Reconstruction Table cell [5, 2] and fetch the row number stored there (3). The next (third or STATUS) field value within the supplier record we are reconstructing is in the STATUS position in the third row of the Field Values Table-in other words, in cell [3,3]. Go to that cell and fetch the value stored there (status 20).
>
> *Step 4:* Go to the corresponding Record Reconstruction Table cell [3, 3] and fetch the value stored there (which is 3 again). The next (fourth or CITY) field value within the supplier record we are reconstructing is in the CITY position in the third row of the Field Values Table-in other words, in cell [3,4]. Go to that cell and fetch the value stored there (city name London).
>
> *Step 5:* Go to the corresponding Record Reconstruction Table cell [3, 4] and fetch the value stored there (1). Now, the "next" field value within the supplier record we are reconstructing looks like it ought to be the fifth such value; however, supplier records have only four fields, so that "fifth" wraps around to become the first. Thus, the "next" (first or S#) field value within the supplier record we are reconstructing is in the S# position in the first row of the Field Values Table-in other words, in cell [1,1]. But that is where we came in, and the process stops."

To this point, the model provides no potential database size reduction because all values are kept and additional data is held in the Record Reconstruction Table, we refer to this as TR Version 1. The desired reduction is achieved when the **Condensed columns** are introduced. As can be observed in Fig. 1(b) a considerable amount of redundant data is stored, this is also true in traditional N-ary systems (see Fig. 1(a)). The Column-Condensing process aims to eliminate such redundancy by keeping unique values at column level; we refer to this as TR Version 2.1. This process should

	S#	SNAME	STATUS	CITY
1	S1	Adams	10 [1:1]	Athens [1:1]
2	S2	Blake	20 [2:3]	London [2:3]
3	S3	Clark	30 [4:5]	Paris [4:5]
4	S4	Jones		
5	S5	Smith		

Fig. 2. Condensed Field Values Table with row ranges

be applied selectively, since attempting to condense columns already consisting of unique values does not make sense. However, as each value can be used in many "records" it is necessary to keep **Row Ranges** (numbers in squared brackets in Fig. 2) to avoid losing information on how to reconstruct the record, we refer to this as TR Version 2.3. The resulting FVT with condensed columns and row ranges is presented in **Fig. 2.**

The Column-Condensing process destroys the unary relationship between the cells of the FVT and the cells in the RRT, but the ZigZag algorithm is easily adaptable as stated in [3:956].

"Consider cell [i, j] of the Record Reconstruction Table. Instead of going to cell [i,j] of the Field Values Table, go to cell [i',j] of that table where cell [i',j] is that unique cell within column j of that table that contains a row range that includes row i."

3 Implementation

We have implemented algorithms to create the Field Values Table, the Record Reconstruction Table and the Zigzag algorithm to rebuild the records. Some variations and improvements have been made during implementation and we will describe those in the following subsections.

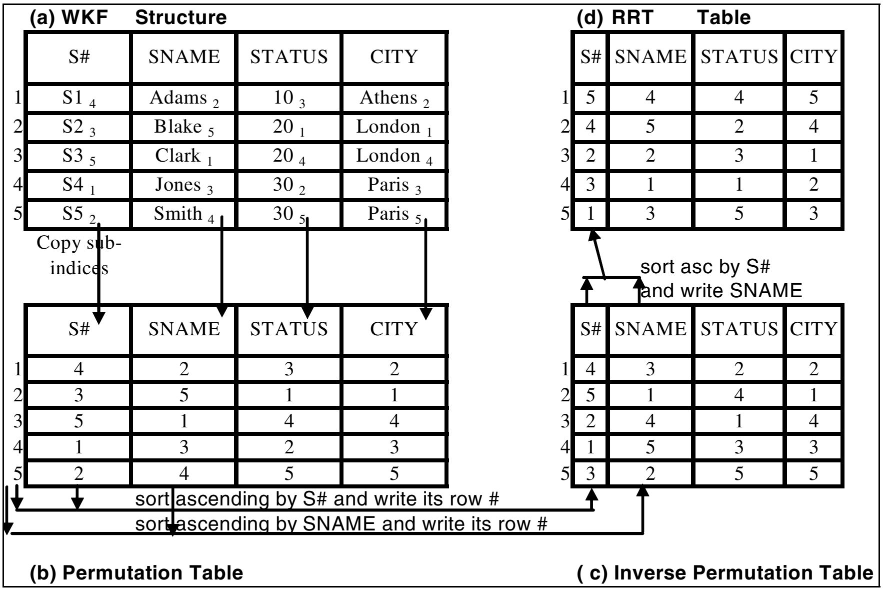
This implementation was focused on the initial bulk load of the Data Warehouse and it retained the limitations identified by Date [3:943] where updates are discarded and the Database is Read Only. Inherently, Data Warehouse environments are more suited to these assumptions (with batch updates during off line hours and read only during operation hours) than transactional systems. Consequently, we would argue that the benchmarking of the TR model for Data Warehouse environments is both relevant and important.

An extraction tool was written in order to generate the flat files that will be processed by the TR model algorithms. One important point that was introduced during the extraction process is that each record is pre-appended with its record number, to provide additional support during the Transformation and Loading stages.

3.1 The Field Values Table

Data within each column is rearranged into ascending sort order. All operations are made in bulk and parallelising as much as possible, extensive use of the sort, parallelisation and synchronisation mechanisms offered by the operating system has been made. The sorting process of each column is made in parallel as each column is independent. Improvements were introduced in order to prepare the creation of the RRT and minimise reprocessing. The first step is to create a structure (we call it Working File WKF and it enhances the algorithms described by Date) where each column is in ascending order and maintains the original record number as sub-index Fig. 3(a). From this structure (WKF) the Field Values Table with condensed columns (Fig. 2) is derived by choosing unique values in ascending order and recording the corresponding row ranges. The other structure derived from the WKF structure is the Permutation Table [3:951] Fig. 3(b), which is required to build the Record Reconstruction Table.

The Column-Condensing process is selective as recommended by Date [3:953]. In our implementation we establish that columns with unique values and those where the reduction would be lower than the established condensation threshold are not condensed. The condensation threshold is not part of Date's algorithms and we define it as the ratio between the original number of values (including repetitions) and the number of unique values. The condensation threshold is 30% and this was set after testing different threshold values and considering the balance between the disk space required to keep row ranges versus the disk space required to keep the complete column with relatively few repetitive values. The processing time was also considered to set the threshold. This approach was taken in order to maintain a balance between the theoretical model and the pragmatic implementation.



(a) WKF Structure

	S#	SNAME	STATUS	CITY
1	S1 $_4$	Adams $_2$	10 $_3$	Athens $_2$
2	S2 $_3$	Blake $_5$	20 $_1$	London $_1$
3	S3 $_5$	Clark $_1$	20 $_4$	London $_4$
4	S4 $_1$	Jones $_3$	30 $_2$	Paris $_3$
5	S5 $_2$	Smith $_4$	30 $_5$	Paris $_5$

(d) RRT Table

	S#	SNAME	STATUS	CITY
1	5	4	4	5
2	4	5	2	4
3	2	2	3	1
4	3	1	1	2
5	1	3	5	3

(b) Permutation Table

	S#	SNAME	STATUS	CITY
1	4	2	3	2
2	3	5	1	1
3	5	1	4	4
4	1	3	2	3
5	2	4	5	5

(c) Inverse Permutation Table

	S#	SNAME	STATUS	CITY
1	4	3	2	2
2	5	1	4	1
3	2	4	1	4
4	1	5	3	3
5	3	2	5	5

Copy sub-indices
sort asc by S# and write SNAME
sort ascending by S# and write its row #
sort ascending by SNAME and write its row #

Fig. 3. Proposed alternative generation algorithm to build the Record Reconstruction Table

3.2 The Record Reconstruction Table

A permutation is defined by Date [3:951] as the ordering of the records by a particular column. For example the permutation corresponding to ascending S# ordering is 4,3,5,1,2 (refer to Fig. 1(a)). According to this definition, the *Permutation Table* is computed directly from the original input file, but instead of doing it in this way we derived it from the WKF structure by using the already ascending ordered values but taking the sub-indices, see Fig. 3(a) and 3(b). Thus the columns of the Permutation Table can be computed in parallel. The first column of the Permutation Table keeps an ascending sequence from 1 to the number of records in the table (i.e. the row number within the Permutation Table).

Date also defined the *Inverse Permutation as, "That unique permutation that if applied to the original sequence 4,3,5,1,2, will produce the sequence 1,2,3,4,5".* It is computed column by column on the permutation table by applying the previous rule to obtain the ascending sequence 1,2,3,4,5.

We found that is more efficient to compute the Inverse Permutation Table from the existing Permutation Table by taking each column together with its corresponding row number and then sorting ascending by the corresponding column values and writing the row number in the Inverse Permutation Table rather than the column values (in this example S#), see Fig 3(b) and 3(c). The resulting Inverse Permutation Table is exactly the same as that described by Date, and its columns can be computed in parallel.

Finally the Record Reconstruction Table can be built from the Inverse Permutation Table. Date's algorithm [3:952] is as follows:

"Go to the cell [i,1] of the Inverse Permutation Table. Let that cell contain the value r; also the next cell to the right, cell [i,2], contain the value r'. Go to the rth row of the Record Reconstruction Table and place the value r' in cell [r,1]."

Following the algorithm for i=1,2,3,4,5 will produce the entire S# column of the Record Reconstruction Table. The other columns are computed in a similar way.

This algorithm processes one row at a time which is inefficient. We propose the following algorithm to compute the Record Reconstruction Table:

From the Inverse Permutation Table use column i and column i+1, sort in ascending order by i-th column values and write the value held in i+1 column to the corresponding i-column in the RRT. This applies to all columns except the last one which must use the n-th column and the 1st column. See Fig. 3(c) and (d).

Our algorithm enables bulk processing and each column is processed in parallel as there is no dependency between columns.

3.3 Implemented Versions

Four versions of the TR model were implemented in order to gather information regarding its behavior with different characteristics. Each version, its characteristics, and the implications for the ZigZag algorithm [3:948] are listed in **Table 1.**

Table 1. TR model versions

Version 1	Field Values Table (FVT) keeps repetitive values. Both the Record Reconstruction Table (RRT) and the ZigZag Algorithm are as stated in [3].
Version 2.1	All columns in all tables are condensed in their corresponding FVT tables and the remaining unique values keep row ranges. The ZigZag Algorithm is enhanced to be aware of row ranges. The RRT Table remains unchanged.
Version 2.2	Only the Fact Table is considered to condense its columns. The ZigZag algorithm needs to be improved to detect the Fact Table and be aware that row ranges only exist in the Fact Table but not in any other table. RRT remains unchanged.
Version 2.3	Selective column condensation in the FVT is applied if the established threshold is reached; this is applied to all tables as proposed by Date. The ZigZag algorithm needs to be aware of condensed and uncondensed columns, and those which are condensed have row ranges. RRT Table remains unchanged.

4 Benchmarking Environment

In order to benchmark the TR model and its implementation in Data Warehouse environments the standard and well accepted TPC-H [13] data set was chosen. TPC-H has the characteristics of real life data warehouses where a big Fact table exists with complementary tables around this table (no matter if it is a Star or Snow-flake schema). The authors are very experienced in the use of this data set for benchmarking Data Warehouses considering different data models [4],[5],[8] to highlight their specific characteristics. The tests were executed with two database sizes, called scale factors (SF=100 MB and SF=1GB).

The machine used to evaluate the defined metrics has 1 CPU Pentium IV @ 1.60GHz, 512 MB in RAM, Cache size 256 KB, Bus speed 100MHz and Operating System Fedora 2 version 2.4.9-12. The relational instantiation used is Oracle Version 9.0 with its corresponding SQL*Plus and SQL*Loader utilities.

SHQL version 1.3 [9] is used to provide a SQL interface. It was used to execute the necessary DDL statements. The implemented algorithms made use of the initial structures generated by SHQL to build and manipulate the required tables (FVTs and RRTs).

5 Experimental Results and Analysis

The results presented in this section follow the flow of the Extraction Transformation and Loading Process. As mentioned before, a tool was made to extract data and generate the input flat files to be loaded in both Relational and TR instantiations. The differences between input flat file sizes are small but the TR input files are slightly bigger because of the extra column required to keep the row number that will be used for further processing. Resulting flat file sizes are presented in **Table 2.**

Table 2. Extracted file sizes to be used as input

Scale Factor (SF)	Relational (MB)	TR (MB)
100 MB	102.81	103.77
1GB	1,049.60	1067.46

The next step is to Transform and Load the input files into the instantiations for both models. As stated before four different versions of TR were implemented (Table 1). The results for these versions are presented in **Table 3.** (SF=100MB) and **Table 4.** (SF=1GB).

Table 3. DBMS Object size in MB with SF=100MB

TPC-H Table Name	Relational	**TR V1** (with repetitive values)	**TR V2.1** (everything condensed)	**TR V 2.2** (only Fact Table condensed)	**TR V2.3** (selective condensation)
Region	0.0625	0.0005	0.0005	0.0005	0.0005
Nation	0.0625	0.0024	0.0027	0.0024	0.0023
Supplier	0.1875	0.1600	0.1900	0.1600	0.1600
Customer	3.00	3.04	3.41	3.04	2.79
Part	3.00	3.41	2.34	3.41	2.12
PartSupplier	13.00	14.19	13.90	14.19	13.05
Orders	19.00	25.63	21.91	25.63	18.56
Lineitem	88.00	137.07	86.09	86.09	86.90
TOTAL	**126.31**	**183.50**	**127.84**	**132.52**	**123.58**

Table 4. DBMS Object size in MB with SF=1GB

TPC-H Table Name	Relational	**TR V1** (with repetitive values)	**TR V2.1** (everything condensed)	**TR V 2.2** (only Fact Table condensed)	**TR V2.3** (selective condensation)
Region	0.0625	0.0005	0.0005	0.0005	0.0005
Nation	0.0625	0.0024	0.0027	0.0024	0.0023
Supplier	2.0	1.75	2.07	1.75	1.68
Customer	27.0	32.00	36.23	32.00	29.3
Part	30.0	36.37	24.74	36.37	21.52
PartSupplier	128.0	149.00	137.69	149.00	130.93
Orders	192.0	274.96	235.32	274.96	200.89
Lineitem	848.0	1489.71	925.73	925.73	962.22
TOTAL	**1,227.1**	**1,983.78**	**1,361.81**	**1,419.80**	**1,346.54**

TR Version 1 is of limited value because it keeps all repetitive values and adds more information within the RRT table; as a result it increases the N-ary Relational Instantiation DB size by a factor of around 50%.

From Tables 3 and 4 with version 2.1 (where all columns are condensed); in medium sized tables (Supplier and Customer) the effect of condensing resulted in bigger table sizes than the corresponding sizes without being condensed, this is as a result of keeping the row ranges; for the bigger tables (Lineitem and Orders) the Column-Condensing process is beneficial. Considering that the Lineitem Table (Fact Table) is the biggest table, Version 2.2 was introduced. The benefit of this version is that only the Fact Table is passed through the Column-Condensing process in order to reduce the CPU processing time, but the downside is that the final DB size is bigger than Version 2.1. Finally in version 2.3 all tables are processed by the Column-Condensing process but in a selective fashion where if the estimated condensing level reaches the established threshold then the column is condensed, otherwise there is no benefit in investing CPU processing time which will not achieve significant column size reductions. According to the results obtained, version 2.3 is the one that offers a better balance between processing time and disk space consumption but requires a complex ZigZag algorithm to rebuild records when necessary. The ZigZag algorithm needs to be intelligent enough to identify condensed columns and uncondensed columns and make use of row ranges when rebuilding the original record.

These analyses have been focused on the FVT since, as identified by Date [3:954] the FVT will offer compression benefits. However, our experimental results show that, even when condensing repetitive values, the final database size is bigger or, at best, only slightly smaller than the traditional N-ary Relational instantiation.

Further analyses based on Version 2.3, show that the FVT behaves in keeping with Date's description, but the problem is with the RRT. While the RRT at first sight only holds integers, after millions of rows (i.e. Lineitem=6 million rows for SF=1GB), these integers are of more considerable size and occupy most of the final database space. These results are presented in **Table 5.** (SF=100MB) and **Table 6.** (SF=1GB).

As in any Data Warehouse environment the Fact table (in this case LineItem) occupies most of the space, with SF=1GB this table occupies 962MB of 1,346MB of the total DB size. Importantly, however, the corresponding RRT for Lineitem required 760MB of those 962MB. In general RRT structures are 65% of the total DB space while FVT structures are the remaining 35%. From these results it is clear that further work is necessary to tackle the RRT structures, and particularly the RRT for Fact Tables, to enable the TR model to achieve the benefits predicted.

Another key finding of the experiment is that with the bigger scale factor, the Version 2.3 (selective condensation) has better results than any other version (see Table 4), including Version 2.1 where every column is condensed, but remains very close (additional 10%) to the traditional N-ary Relational implementation.

Finally, another aspect to be considered is the time to Transform and Load the input files into the DBMS. In this aspect the TR instantiation required more time than the N-ary Relational instantiation. The Transformation and Loading time was not linear with respect to DB size, as presented in **Table 7.** with around 4 times more with SF=100MB and 10 times more with SF=1GB.

Table 5. Transrelational^{TM} RRTs and FVTs with SF=100MB

TPC-H Table Name	Relational (MB)	TR V2.3 (MB)	RRT (MB)	FVT (MB)	% RRT vs total	%FVT vs total
Region	0.0625	0.0005	0.0001	0.0004	11%	89%
Nation	0.0625	0.0023	0.0003	0.0020	13%	87%
Supplier	0.1875	0.1600	0.0300	0.1300	19%	81%
Customer	3.00	2.79	0.68	2.11	24%	76%
Part	3.00	2.12	1.04	1.08	49%	51%
PartSupplier	13.00	13.05	2.68	10.37	21%	79%
Orders	19.00	18.56	8.95	9.61	48%	52%
Lineitem	88.00	86.90	66.36	20.54	76%	24%
TOTAL	126.31	123.58	**79.74**	**43.84**	**65%**	**35%**

Table 6. Transrelational^{TM} RRTs and FVTs with SF=1GB

TPC-H Table Name	Relational (MB)	TR V2.3 (MB)	RRT (MB)	FVT (MB)	% RRT vs total	%FVT vs total
Region	0.0625	0.0005	0.00005	0.00045	10%	90%
Nation	0.06	0.0023	0.00030	0.00200	13%	87%
Supplier	2.0	1.68	0.37	1.31	22%	78%
Customer	27.0	29.3	8.06	21.24	28%	72%
Part	30.0	21.52	12.29	9.23	57%	43%
PartSupplier	128.0	130.93	31.41	99.52	24%	76%
Orders	192.0	200.89	51.69	149.20	26%	74%
Lineitem	848.0	962.22	760.34	201.88	79%	21%
TOTAL	1,227.1	1,346.54	**864.16**	**482.38**	**64%**	**36%**

Table 7. Transformation and Loading Times

	Scale Factor (SF)	Transformation & Loading time
Relational	100MB	2.4 minutes
Transrelational^{TM}	100MB	10.2 minutes
Relational	1GB	19.9 minutes
Transrelational^{TM}	1GB	191.6 minutes

6 Conclusions and Future Work

The TR model as described by Date is very appealing for Data Warehouse environments, but after analysis of our results it really does not offer the tangible benefits that we were looking for. It may reduce the inefficiency of storing repetitive values at column level that exists in traditional N-ary Relational implementations, but the expected reduction in Data Warehouse size was not achieved. This has to be set against the results achieved for other alternative models [4][5][8], especially Binary Relational. However we have been able to produce a novel public domain instantiation of the TR model described by Date, and we have identified and implemented improvements to the algorithms of that model. Further research needs to be done on the RRT structure to minimise its size and thus reduce the final DB size, but at the same time we envisage that this will increase the processing time. Date [3:956] identifies another feature that could be used in TR (which we might call Version 3) which uses *Merged Columns*. However, we have identified that the existing drawbacks of the RRT should be the next problem to be tackled, and we would argue that this needs to be undertaken before introducing more complexity to the algorithms for the marginal benefits in terms of the final DB size that might be achieved in Version 3.

Based on our research results [4][5][8] and the current state of the TR model, we would argue that it does not represent the model of choice for future Data Warehouse environments.

Acknowledgments. The present work has been possible due the financial support of Consejo Nacional de Ciencia y Tecnología México (CONACYT) and Secretaría de Educación Pública (SEP) México. The authors gratefully acknowledge that support.

References

1. Boncz, Peter. Monet: A next generation DBMS Kernel for query intensive applications. PhD Thesis. Amsterdam, 2002.
2. Copeland, George P. Khoshafian, Setrag N. A Decomposition Storage Model. In Proceedings of the ACM SIGMOD Int'l. Conf. On Management of Data, pp 268-279, May 1985.
3. Date, C.J. An introduction to Database Systems. Appendix A. The Transrelational Model, Eighth Edition. Addison Wesley. 2004. USA. ISBN: 0-321-18956-6. pp.941-966
4. Gonzalez-Castro, Victor. MacKinnon, Lachlan. A Survey "Off the Record" - Using Alternative Data Models to increase Data Density in Data Warehouse Environments. Proceedings BNCOD Volume 2. Edinburgh, Scotland 2004.
5. Gonzalez-Castro, Victor. MacKinnon, Lachlan. Data Density of Alternative Data Models and its Benefits in Data Warehousing Environments. British National Conference on Data Bases, BNCOD 22 Proceedings Volume 2. pp 21-24. Sunderland, England U.K. 2005.
6. MonetDB. ©1994-2004 by CWI. http://monetdb.cwi.nl
7. Pendse, Nigel. Database explosion. http://www.olapreport.com Updated Aug, 2003.
8. Petratos, P and Michalopoulos D. (editors) Gonzalez-Castro V. and Mackinnon L.(authors). Using Alternative Data Models in the Context of Data Warehousing. 2005 International Conference in Computer Science and Information Systems. Athens Institute of Education and Research, ATINER. Greece. ISBN: 960-88672-3-1. pp 83-100. 2005.
9. SHQL. http://backpan.cpan.org/modules/dbperl/scripts/shql/

10. Stonebraker, Mike, et.al. C-Store: A column Oriented DBMS. Proceedings of the 31st VLDB conference, Trondheim, Norway, 2005. pp. 553-564.
11. SybaseIQ web site. www.sybase.com/products/informationmanagement/sybaseiq
12. Sybase Inc. Migrating from Sybase Adaptive Server Enterprise to SybaseIQ white paper. USA 2005.
13. TPC Benchmark H (Decision Support) Standard Specification Revision 2.1.0. 2002.
14. U.S. Patent and Trademark Office: Value-Instance-connectivity Computer-Implemented Database. U.S. Patent No. 6,009,432 (December 28, 1999).

Reducing Sub-transaction Aborts and Blocking Time Within Atomic Commit Protocols

Stefan Böttcher[1], Le Gruenwald[2,*], and Sebastian Obermeier[1]

[1] University of Paderborn, Computer Science
Fürstenallee 11, 33102 Paderborn, Germany
{stb, so}@uni-paderborn.de
[2] The University of Oklahoma, School of Computer Science
200 Telgar street, Room 116 EL, Norman, OK 73019-3032, USA
ggruenwald@ou.edu

Abstract. Composed Web service transactions executed in distributed networks often require an atomic execution. Guaranteeing atomicity in mobile networks involves a lot more challenges than in fixed-wired networks. These challenges mostly concern network failures, e.g. network partitioning and node disconnection, each of which involves the risk of infinite blocking and can lead to a high number of aborts.

In this paper, we introduce an extension to existing atomic commit protocols, which decreases the time during which a resource manager that is involved in a web-service is blocked. In addition, our proposal reduces the number of sub-transaction aborts that arise due to message loss or due to conflicting concurrent transactions by distinguishing reusable and repeatable sub-transactions from aborting sub-transactions.

1 Introduction

The use of Web service transactions within fixed wired network structures is supported by multiple specification languages, e.g. BPEL4WS [1] or BPML [2]. Especially when Web service transactions are composed of several sub-transactions, it is often crucial that either all or none of the sub-transactions are executed. Atomic commit protocols are a standard technique to meet this requirement, i.e. for guaranteeing an atomic execution of nested transactions. However, in the context of Web services, sub-transactions may be dynamically invoked and therefore are not always known in advance. Approaches like the "WS-Atomic-Transaction" standard ([3]) suggest using a modified 2-Phase-Commit-protocol (2PC, [4]), where each dynamically invoked sub-transaction registers at the coordinator by itself.

However, if parts of a Web service transaction should be processed within a mobile ad-hoc environment where mobile participants are suspect to disconnects and network partitioning may occur, the use of 2PC may lead to a long blocking

[*] This material is based upon work supported by (while serving at) the National Science Foundation (NSF) and the NSF Grant No. IIS-0312746.

D. Bell and J. Hong (Eds.): BNCOD 2006, LNCS 4042, pp. 59–72, 2006.

time of mobile participants. Even if we assume the coordinator to be stable, a database that disconnects while executing a transaction blocks the data that the transaction has accessed until it gets the commit decision.[1]

In addition, a high number of disconnects and reconnects may lead to an unnecessarily high number of aborts since timeouts for reconnections are hard to estimate. Imagine a coordinator that waits for the last missing vote for commit. If the coordinator waits too long, concurrent transactions that access conflicting data cannot be processed. If the coordinator does not wait and aborts the transaction due to the missing vote, the database may reconnect just at this moment and the transaction was superfluously executed and aborted. Since estimations about message delivery times and transaction execution times within dynamic mobile ad-hoc networks are often significantly different to the observed times, the coordinator will very likely estimate wrong timeouts.

Solutions like timeout-based approaches ([5]) optimistically suggest to commit a transaction and apply compensation transactions to undo in the case of abort. However, since committed transactions can trigger other operations, we cannot assume that compensation for committed transactions in mobile networks, where network partitioning makes nodes unreachable but still operational, is always possible. Therefore, we focus on a transaction model, within which atomicity is guaranteed for distributed, non-compensatable transactions.

In this paper, we show how the suspend state can be used within 2PC to not only reduce the blocking times of databases, but also to reduce the number of aborts and reuse existing results as far as possible. Section 2 describes details of our assumed transaction model and introduces necessary requirements for guaranteeing atomic commit in a mobile environment. In Section 3, we propose a solution, which consists of the following key ideas: a non-blocking state for transactions that are ready to vote for commit; a flexible reaction to concurrency failures by distinguishing failures that require a transaction abort, failures that only require the repetition of a sub-transaction, and failures that allow the reuse of a sub-transaction; and finally, we show how a tree data structure that represents the execution status of all active sub-transactions can be used by the coordinator to efficiently coordinate the transaction.

2 Problem Description

This section describes our assumed transaction model and identifies the additional requirements that arise when using mobile devices within a transaction. Finally, we describe the goal, i.e., to reduce both blocking and the number of transaction aborts.

2.1 Transaction Model

Our transaction model is based on the Web Services Transactions specifications. However, since we focus on the atomicity property, we can rely on a much sim-

[1] Even validation-based synchronisation shows a blocking behavior, cf. Section 2.4.

pler transaction model, e.g., we do not need a certain Web service modeling or composition language like BPEL4WS [1] or BPML [2]. Therefore, we have designed our transaction model to consist only of the following objects, that are "application", "transaction procedure", "Web service", and "sub-transaction". In the following, we explain how we understand these terms as well as their relationship to each other.

An *application AP* can consist of one or more *transaction procedures.* A transaction procedure is a Web service that must be executed in an atomic fashion. Transaction procedures and Web services are implemented using local code, database instructions, and (zero or more) calls to other remote Web services. Since a Web service invocation can depend on conditions and parameters, different executions of the same Web service may result in different local executions and different calls to other Web services.

An *execution* of a transaction procedure is called a global transaction T. We assume that an application AP is interested in the result of T, i.e., whether the execution of a global transaction T has been committed or aborted. In case of commit, AP may be further interested in the return values of T's parameters.

The relationship between transactions, Web services, and sub-transactions is recursively defined as follows: We allow each transaction or sub-transaction T to dynamically invoke additional Web services offered by physically different nodes. We call the execution of such Web services invoked by the transaction T or by a sub-transaction T_i the sub-transactions $T_{si} \dots T_{sj}$ of T or of T_i, respectively.

Whenever during the execution of the global transaction T, a child or descendant sub-transaction T_s of T, or T itself, invokes the sub-transactions $T_1 \dots T_n$, the atomicity property of T requires that either all transactions $T, T_1 \dots T_n$ commit, or all of these transactions abort.

Since we allow dynamic Web service invocations, we assume that each Web service only knows which Web services it calls directly, but not which Web services are invoked by the called Web services. This means that at the end of its execution, each transaction T_i knows which sub-transactions $T_{is_1} \dots T_{is_j}$ it has called directly, but T_i, in general, will not know which sub-transactions have been called by $T_{is_1} \dots T_{is_j}$. Moreover, we assume that usually a transaction T_i does not know how long its invoked sub-transactions $T_{is_1} \dots T_{is_j}$ will run.

We assume that during the execution of a sub-transaction, a database enters the following phases: the *read-phase*, the *coordinated commit decision phase*, and, in case of successful commit, the *write-phase.* While executing the read-phase, each sub-transaction carries out write operations only on its private transaction storage. Whenever the coordinator decides to commit a transaction, each database enters the write phase. During this phase, the private transaction storage is transferred to the durable database storage, such that the changes done throughout the read-phase become visible to other transactions after completion of the write-phase.

In the mobile architecture for which our protocol is designed, Web services must be invoked by messages instead of synchronous calls for the following reason. We want to avoid that a Web service T_i that synchronously calls a sub-transaction T_j cannot complete its read phase and cannot vote for commit before T_j sends its return value. For this reason, we allow invoked sub-transactions only to return values indirectly by asynchronously invoking corresponding receiving Web services and not synchronously by return statements.[2]

Since (sub-)transactions describe general services that do not exclusively concern databases, we also call the node executing a sub-transaction *resource manager (RM).*

The following example shows the necessity of an atomic execution of transactions within our transactional model:

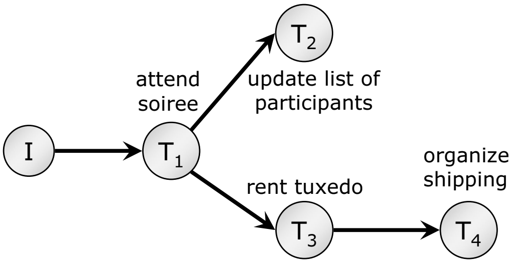


Fig. 1. The initiator I of a Web service transaction T and its sub-transactions $T_1 \ldots T_4$

Example 1. Assume a conference participant decides to attend the conference's soiree and invokes the corresponding Web Service. As shown in Figure 1, the corresponding global transaction T, started by the Initiator I, invokes the Web Service for attending the soiree. The corresponding sub-transaction T_1 then updates the participant list by invoking sub-transaction T_2 and detects that the participant has not brought a tuxedo. Therefore, a tuxedo must be rented and T_1 calls a sub-transaction T_3. To organize the shipping of the tux to the participant's hotel, T_3 calls the Web service T_4 of a shipping company. It is obvious that no booking component is allowed to fail: if the participant list is already full, if there are no tuxedos available, or if the shipping company cannot deliver the tuxedo to the participant on time, the participant cannot attend the soiree. Therefore, all sub-transactions are required to be performed in an atomic fashion.

This example shows one characteristic of our Web service transactional model: The Initiator and the Web services do not know every sub-transaction that is generated during transaction processing.

Our model differs from other models that use nested transactions (e.g. [6], [7], [3]) in some aspects including, but not limited, to the following: Since compensation of a sub-transaction may not be possible if a mobile network is partitioned, we consider each sub-transaction to be non-compensatable. Therefore, no sub-transaction is allowed to commit independently of the others or before the commit coordinator guarantees – by sending the commit message – that all sub-transactions will be committed.

[2] However, if a synchronous call within T_i to T_j is needed, e.g. because T_i needs return values from T_j, it is possible to achieve this behavior by splitting T_i into T_{i1} and T_{i2} as follows: T_{i1} includes T_i's code up to and including an asynchronous invocation of its sub-transaction T_j; and T_{i2} contains the remaining code of T_i. After T_j has completed its read phase, T_j performs an asynchronous call to T_{i2} which can contain return values computed by T_j that shall be further processed by T_{i2}.

Different from CORBA OTS ([7]), we assume that we cannot always identify a hierarchy of commit decisions, where aborted sub-transactions can be compensated by executing other sub-transactions.

A Web service T may be programmed by using control structures, e.g. `if <Condition> then <T1> else <T2>`. This means that resource managers executing a Web service T may dynamically invoke other sub-transactions $T_1 \ldots T_j$.

We assume a message-orientated communication. This means that a Web service does not explicitly return values but may pass parameters to other invoked Web services which perform further operations based on these results.

2.2 Requirements

Besides the main requirement to design an atomic commit protocol for guaranteeing the atomic execution of non-compensatable Web service transactions including sub-transactions in mobile networks, we identified the following additional requirements especially for mobile network protocols.

A resource manager failure or disappearance may occur at any time. This, however, must not have blocking effects on other resource managers.

A general problem when guaranteeing atomicity for transactions that are non-compensatable is that participating resource managers are blocked during protocol execution. The time that a distributed sub-transaction remains in a state within which it waits for a commit decision can be much longer in dynamic mobile environments than it is in fixed-wired networks, since link failures and node failures occur significantly more frequently. Furthermore, each sub-transaction T_{si} of a transaction T running on a resource manager RM that resides in a blocking state also involves the risk of the infinite blocking of other sub-transactions of T and sub-transactions of other transactions running on RM. Therefore, we need efficient mechanisms to reduce the time that a resource manager is in a blocking state and to unblock sub-transactions if other resource managers do not respond.

Within mobile networks, message delay or message loss is no exception. In case that the vote message of a resource manager is lost or delayed, traditional protocols like 2PC either wait until the missing vote arrives and block all participating resource managers in the meantime, or they abort the transaction which thereafter can be repeated as a whole. Nevertheless, a lost vote differs from an explicit vote for abort. On one hand, a general abort may not be necessary for many sub-transactions, especially if there is no other concurrent transaction that tries to get a lock on the same data that the sub-transactions are accessing. On the other hand, if there are concurrent transactions that try to access data in a conflicting way, an abort is necessary for processing these concurrent transactions. Therefore, a requirement is to abort as few transactions as necessary.

There are situations, especially when network partitioning occurs, where blocking is proven to be unavoidable until the network is reconnected again ([8]). Of course, our solution cannot avoid this blocking, but it should reduce the chance that resource managers are blocked by minimizing the time period during which a failure could have a blocking effect on the resource managers.

Our contribution should be an extension to existing atomic commit protocols, such that a concrete protocol can be chosen depending on the applications' needs.

Our protocol extension should use previous results to the greatest possible extent, so that an unnecessary repetition of a sub-transaction can be avoided in as many cases as possible.

It should be possible that the user can abort a transaction as long as the transaction has not been globally committed.

2.3 Further Assumptions

Our atomic commit protocol extension is based on the following assumptions:

In case of resource manager failure, the atomicity property requires the following: Whenever a resource manager RM_i is unreachable after the commit decision on a transaction T was reached, i.e. RM_i has failed or is separated from the network for an indefinitely long time, it is not a violation of the atomicity constraint if RM_i has not executed its sub-transaction T_i of T. However, if the resource manager RM_i recovers and returns to the network, RM_i must immediately execute or abort T_i, depending on whether the commit decision on T was commit or abort, before further participating in the network.

We assume that at least some (sub-)transactions are non-compensatable. We claim that this is a realistic assumption for mobile environments. Even in our simple example given above (Example 1), some sub-transactions (e.g. T_3 and T_4) can be considered to be non-compensatable since many of today's rental or shipping companies demand expensive cancellation fees. We cannot tolerate a commit protocol which must repeatedly change or cancel contracts.

An aborted (sub-) transaction cannot be compensated by the invocation of a different sub-transaction; in contrast to Corba OTS ([7]), in which a hierarchy of commit decisions allows this kind of compensation.

The stability of the coordinator process itself is not a topic of this proposal. There are many contributions that handle coordinator failures, e.g., by running special termination protocols or by increasing coordinator availability with multiple coordinators (e.g [9], [10], or [11]). We only propose an improvement which is compatible to each of these commit protocols. The concrete protocol can be chosen depending on the desired coordinator stability. Therefore, we do not discuss coordinator failure in this contribution.

2.4 Blocking Behavior of Locking and Validation

For synchronization of concurrent transactions in fixed-wired networks, validation is usually considered to be a scheduling technique that avoids blocking. However, we argue that even validation-based synchronization shows a blocking behavior if used in combination with an atomic commit protocol. More precisely, in case of link failures or node failures, locking and validation are equivalent regarding their blocking behavior in the following sense. Assume a sub-transaction T_i, reading the tuples $t(R, T_i)$ and writing the tuples $t(W, T_i)$, has voted for commit and is waiting for a global commit decision.

Two-phase locking would not allow any sub-transaction T_k with $t(W, T_i) \cap (t(W, T_k) \cup t(R, T_k)) \neq \emptyset$ to get the required locks and would therefore block T_k and prevent the completion of T_k´s read phase.

Validation (e.g. [12]) would allow any later sub-transaction T_k with $t(W, T_i) \cap (t(W, T_k) \cup t(R, T_k)) \neq \emptyset$ to enter the read phase. However, since the tuple sets $t(W, T_i)$ and $(t(W, T_k) \cup t(R, T_k))$ are not disjoint, validation aborts T_k during its validation phase and thereby prevents T_k to enter its write phase.

This means that both techniques show a similar behavior when dealing with atomic commit decisions for mobile networks: A transaction T_i that has voted for commit, is waiting for a global commit decision, and has accessed the tuples $t(W, T_i) \cup t(R, T_i)$ prevents other sub-transactions T_k to gain access to the same data. This means that T_i prevents T_k from being committed while T_i only waits for the commit decision.

To reduce both, the blocking time and the probability of an abort, our solution distinguishes between two states within which a transaction waits for the global commit decision: Besides a blocking state "wait for global commit", we introduce the non-blocking "suspend state".

3 Solution

In the following, we describe the solution to the requirements, i.e., how to guarantee atomicity for Web service transactions and how to reduce the time of blocking and the number of transaction aborts compared to standard protocols like 2PC or 3PC. To reduce the time of blocking to a minimum extent, we distinguish between two states where a transaction is waiting for a commit: a blocking state (defined in Section 3.1) and a new non-blocking suspend state (introduced in Section 3.2). In Section 3.3, we explain how the suspend state can be used to reduce the number of transaction aborts and how previous results can be reused to a maximum extent. Finally, Section 3.5 explains by means of a "commit tree" how the coordinator learns of all sub-transactions that are dynamically invoked.

3.1 The Wait for Global-Commit State

We define the wait for global-commit state for the sub-transaction T_i reading the tuples $t(R, T_i)$ and writing the tuples $t(W, T_i)$ in the following way:

Definition 1. *The wait for global-commit state of T_i is a state in which a resource manager waits for a final decision of a commit coordinator on T_i to commit and blocks the tuples $t(W, T_i) \cup t(R, T_i)$. If another transaction T_k is executed while T_i is in the wait for global-commit state, T_k is not allowed to write on the tuples $t(R, T_i)$, and it is not allowed to read or to write on the tuples $t(W, T_i)$. The concurrent transaction T_k must wait until T_i is back in the suspend state or is committed or aborted and T_i has unlocked $t(R, T_i)$ and $t(W, T_i)$.*

3.2 The Non-blocking Suspend State

While waiting for the transaction's commit decision, protocols like 2PC or 3PC remain in a wait state and block the client that has voted for commit ([13], [4]). To reduce the duration of this blocking, our protocol extension suggests an additional *suspend state*, such that a transaction waiting for the global transaction decision can be in either of two states: in the non-blocking suspend state or in the blocking wait state.

For each sub-transaction T_i, let $t(R, T_i)$ denote the data read by T_i and $t(W, T_i)$ denote the data written by T_i. Then, we define the suspend state for the sub-transaction T_i in the following way:

Definition 2. *The suspend state of T_i is a state in which the resource manager RM executing T_i waits for a decision of the commit coordinator on T_i, but does not block the tuples $t(W, T_i) \cup (R, T_i)$.*

If another transaction T_k is executed while T_i is suspended, RM checks whether

$$t(W, T_i) \cap (t(R, T_k) \cup t(W, T_k)) \neq \emptyset \ \texttt{or} \ t(R, T_i) \cap t(W, T_k) \neq \emptyset$$

If this is the case, there is a conflict between T_i and T_k, and therefore, RM locally aborts T_i and can either abort the global transaction T or try a repeated execution of the sub-transaction T_i.

3.3 Using the Suspend State to Reduce the Number of Aborts

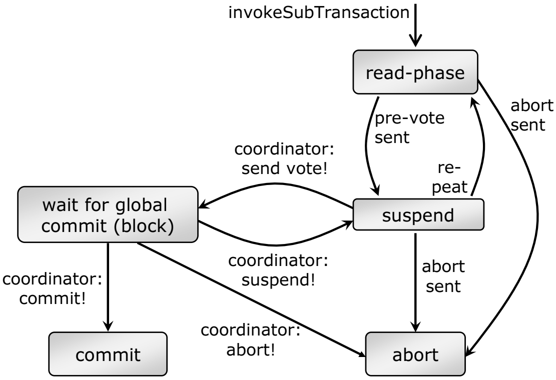


Fig. 2. Automaton showing the states and the received messages of a resource manager

Figure 2 shows an automaton that demonstrates all possible state transitions of a resource manager. Each resource manager enters the suspend state after having executed its read-phase and having sent a pre-vote message on the successful completion of the sub-transaction to the commit coordinator. The commit coordinator considers this pre-vote as a vote that does not bind the resource manager to a commit decision; instead, it only shows the resource manager's successful completion of the read-phase. Note that in comparison to 3PC, a resource manager is not bound to its pre-vote, but may still decide to abort the transaction as long as it is in suspend state. Similar to 2PC and 3PC, whenever a resource manager decides to abort a sub-transaction, it informs the coordinator, which then sends abort messages to the global transaction and all its other sub-transactions.

When the coordinator has received the pre-votes of all resource managers, i.e., all resource managers have pre-voted for commit, the coordinator demands the resource managers to vote on the commit status of the transaction. This vote, in contrast to the pre-vote, is binding and the resource managers proceed to a blocking state where they wait for the global commit decision, i.e., the resource managers are not allowed to abort the transaction themselves while in blocking "wait for global-commit" state. The second collection of votes is needed because a resource manager may come to the decision to abort the sub-transaction while waiting in the suspend state. The coordinator collects these binding vote decisions on the sub-transactions and returns "abort" to the sub-transactions, if at least one RM's vote on its sub-transaction was "abort", and it returns "commit" to the sub-transactions, if it received the binding votes of all sub-transactions and all sub-transactions voted for commit. If the coordinator determines that vote messages are missing after a certain time, it can advise the RMs to put their sub-transactions back to the non-blocking suspend state instead of committing or aborting the sub-transaction. Note that this reduces the number of aborts compared to time-out-based 2PC, in which the coordinator decides to abort a transaction if votes do not arrive before time-out.

After a resource manager has successfully finished its read-phase, it sends the pre-vote message to the coordinator and proceeds to suspend. The resource manager waits in this non-blocking state until one of the following events occur:

- the coordinator demands the vote on the sub-transaction or
- the coordinator aborts the sub-transaction or
- a concurrent transaction causes an abort or a repetition of the sub-transaction due to access conflicts on the tuples accessed by the sub-transaction.

After sending its vote to the coordinator, each resource manager proceeds to the "wait for global-commit state". Since it is now bound to its votes, concurrent transactions are blocked until the coordinator either decides on commit or abort or, after a timeout caused by missing votes, advises the sub-transaction to go into the suspend state again.

In case that the user or the application program wants to abort the transaction, the initiator sends an abort message to the coordinator, which is allowed to abort the transaction anytime before the commit decision is reached.

The benefit of the suspend state lies in the reduction of blocking to only those cases where all pre-votes are present at the coordinator and all are commit. Even then, blocking only occurs for one message exchange cycle, i.e., while the coordinator asks for and retrieves the binding votes. Our protocol definition implies that all resource managers are able to give their vote on the transaction immediately. If, however, not all resource managers respond immediately, the coordinator may again advise the resource managers to go into suspend state while waiting for the missing votes.

2PC, in contrast, blocks a resource manager from the time when it finishes its read-phase, i.e., when it is ready to vote for commit, until the time when it receives the commit decision. The more the duration of the read-phases varies for different sub-transactions belonging to the same global transaction, the more the suspend state reduces the blocking time compared to 2PC.

In addition, in our protocol resource managers may fail or disappear without having a blocking effect on other sub-transactions, while in 2PC, in case of a resource manager failure before reaching a commit decision, the participants are blocked until the transaction is aborted due to timeout. Since timeouts must be sufficiently long to allow the execution of all sub-transactions in 2PC, the blocking time can be significantly long. However, if final votes are missing or delayed in our protocol extension, the coordinator can advise the sub-transactions to go into the non-blocking suspend state after a very short timeout, and sub-transactions do not have to stay in a blocking state until all resource managers have reconnected and voted.

3.4 Abort and Repetition of Sub-transactions

The second key idea of the suspend state, besides reducing the time of blocking, is to reduce the number of transaction aborts. For this purpose, we distinguish three different cases that arise due to an abort of a sub-transaction T_i.

1. T_i cannot be restarted because T_i is not committable or shall not be repeated anymore.
2. T_i is restarted and invokes the same sub-transactions $T_{si} \dots T_{sj}$ with the same values for the actual parameters $P_{si} \dots P_{sj}$ during the repeated execution of the read-phase. This case includes the situation where both executions of the sub-transaction do not invoke any other sub-transaction.
3. T_i is restarted, i.e. executed as T_i', and T_i' invokes different sub-transaction calls during execution of its read phase. In this case, the sub-transactions called by T_i and T_i' or their parameter values differ.

Only the first case requires the global transaction to be aborted, whereas the other types of sub-transaction abort can be solved by a repetition of the aborted sub-transaction T_i, which may involve calls to other sub-transactions $T_{si} \dots T_{sj}$ as well. Now, we discuss each of these three cases in more detail.

T_i cannot be Restarted. There are different reasons why a restart of an aborted transaction does not make sense:

- the transaction abort is caused by the user,
- the commit coordinator requires T_i to abort because another sub-transaction belonging to the same global transaction T required T to abort,
- the abort is caused by the execution of T_i on a resource manager RM itself, i.e., the result of running T_i on RM is that T_i cannot be committed.

In each of these cases, T_i and the global transaction T must be aborted. Note that although we have a hierarchy of invoked sub-transactions, we do not have a hierarchy of commit decisions: If a leaf node is not able to repeat the aborted sub-transaction with a chance of commit, it must vote for abort and the complete transaction T must be aborted and started again.

Restart with Identical Sub-transaction Calls and Parameters. The restart of a sub-transaction T_i is useful if T_i is in suspend state and a concurrent transaction C_k accesses at least one tuple accessed by T_i in a conflicting access mode. The resulting abort of T_i does not necessarily mean that the whole transaction T, of which T_i is a sub-transaction, must be aborted and restarted. Instead, we monitor which sub-transactions T_{si} are called by T_i with which parameters P_{si}[3] for each transaction T_i. Whenever T_i has called T_{si} with parameters P_{si}, and T_i', i.e. the restarted version of T_i, needs to call a sub-transaction T_{si}' with exactly the same parameters P_{si}', i.e. $P_{si}' = P_{si}$, we have the following optimization opportunity: We can omit a new invocation of T_{si} for the following reason. If T_{si} is still in suspend state, the result of a new execution of T_{si} will be the same. If, however, the tuples accessed by T_{si} change and T_{si} leaves the suspend state, the resource manager that executes T_{si} detects this conflict and starts T_{si}' as the repeated execution of T_{si}. The same argument applies to all sub-transactions that T_{si} has called. Since a sub-transaction does not directly return values, but calls receiving Web services instead, T_i' is not affected by the time when T_{si}' is executed.

Therefore, the repeated invocation of a previously called sub-transaction T_{si} can be omitted in the execution of T_i' if the invocation parameters P_{si} have not changed. In this case, we call T_{si} a *re-used sub-transaction.*

After T_i' has executed its read-phase, the resource manager again proceeds to the suspend state. However, if the coordinator message to give a vote on the transaction arrives while RM still executes T_i', RM can inform the coordinator to unblock other waiting resource managers until the repetition is completed.

To summarize, if all sub-transaction calls are re-used or if no sub-transaction was invoked, a renewed sending of the pre-vote is not necessary and the resource manager can continue as in the first execution.

Restart with Different Sub-transaction Calls and Parameters. If we allow repetition, a problem may arise if the invoked sub-transactions $T_{si} \dots T_{sj}$ differ in a repeated execution of a sub-transaction T_i. Therefore, a resource manager not only remembers invoked sub-transactions, but also the invocation parameters $P_{si} \dots P_{sk}$.

If T_i is restarted as T_i' and needs to invoke the sub-transactions $T_{si}' \dots T_{sj}'$, it checks whether it can re-use the sub-transaction calls $T_{si} \dots T_{sj}$ of T_i. If this is not the case or if some calls are different, those sub-transactions $T_{si}' \dots T_{sj}'$ that do not find re-usable sub-transactions must be executed again. Furthermore, a sub-transaction T_{si} which is no longer needed for the execution of T_i' can be locally aborted. A local abort of T_{si} means that T_{si} and all of its sub-transactions are aborted, but the global transaction T and all other sub-transactions not being a descendant of T_{si} are not aborted. The advantage is that the global transaction T and all other sub-transactions of T do not have to be repeated.

Furthermore, the sub-transaction T_{si}' now belongs to T and the coordinator must be informed to wait for the pre-vote of this newly invoked sub-transaction

[3] The parameters include the name of the transaction procedure or web-service and all its actual parameters.

T'_{si} instead of waiting for T_{si}. This information is passed from T_i to the coordinator by a renewed sending of the pre-vote with updated parameters to the coordinator at the end of the sub-transaction execution of T'_i.

In addition, a resource manager that is going to restart a sub-transaction T_i can inform the coordinator about this restart, so that a possible abort vote of a descendant sub-transaction T_{si} will not cause an immediate global abort. Instead, the coordinator can wait for the pre-vote of T'_i, i.e. the restarted version of T_i, to check whether T_{si} is still needed at all.

3.5 The Coordinator's Commit Tree

The coordinator's *commit tree* is a data structure that allows the coordinator to determine which votes are missing for a commit decision. The tree structure is used to represent dependencies between sub-transactions.

To ensure that the commit tree gets knowledge of all invoked sub-transactions belonging to T, we require that each pre-vote message, sent by a sub-transaction T_i to the coordinator, not only contains the commit vote, but also informs the coordinator about all sub-transactions $T_{s1} \dots T_{sk}$ that are called by T_i. The coordinator then creates the nodes $T_{s1} \dots T_{sk}$ and adds these nodes as child nodes of the commit tree node containing T_i. Since the coordinator needs the pre-votes of all commit tree nodes, it must also wait for the pre-votes of $T_{s1} \dots T_{sk}$.

When a resource manager has repeated the execution of a sub-transaction T_i as T'_i according to Section 3.4, the resource manager sends a renewed pre-vote to the coordinator, which includes the invoked sub-transactions T'_{si} and the IDs of the re-used sub-transactions T_{si}^{reused}. The coordinator then replaces the subtree with root node T_i with T'_i. Each sub-transaction $T_j \in T_{si}$ that is not needed anymore, i.e. $T_j \notin T_{si}^{\text{reused}}$, is locally aborted and deleted from the commit tree. The re-used sub-transactions T_{si}^{reused} and the new sub-transactions T'_{si} are appended as child nodes to T'_{si}.

4 Related Work

To distinguish contributions in the field of atomicity and distributed transactions, we can use two main criteria: first, whether flat or nested transactions are supported and secondly, whether transactions and sub-transactions are regarded as compensatable or non-compensatable. Our contribution is based on a transactional model that assumes sub-transactions to be non-compensatable and allows nested transaction calls.

Our contribution differs from the Web service transaction model of [3] in several aspects. For example, [3] uses a "completion protocol" for registering resource managers at the coordinator, but does not propose a non-blocking state – like our suspend state – to unblock transaction participants while waiting for other participants' votes. In comparison, our suspend-state may even be entered repeatedly during the protocol's execution. However, since the suspend state can be used as a protocol extension, it can also be combined with [3].

Besides the Web service orchestration model, there are other contributions that set up transactional models to allow the invocation of sub-transactions, e.g. [6] or [14]. Common with these transaction models, we have a global transaction and sub-transactions that are created during transaction execution and cannot be foreseen. The main differences to these nested transactional models is that we consider all sub-transactions to be non-compensatable.

The use of a suspend state is also proposed by [15]. However, these approaches are intended for use within an environment with a fixed-wired network and several mobile cells, where disconnections are detectable and therefore transactions are considered to be compensatable.

Our work is based on [16], but goes beyond this in several aspects, e.g., we distinguish three different types of transaction aborts and provide technologies for reusing and locally restarting sub-transactions. In addition, we use the suspend state not only to treat network failures, but also in the failure-free case in order to further reduce the blocking time during failure-free transaction execution.

Corba OTS ([7]) uses the term "suspend" for a concept which differs from our model because our suspend state is a non-blocking state for a mobile environment. Regarding the transactional model, Corba OTS uses a hierarchy of commit decisions, where an abort of a sub-transaction can be compensated by other sub-transactions. However, in the presence of non-compensability, this implies waiting for the commit decision of all descendant nodes. In a mobile environment where node failures are likely, we neither propose to wait nor to block the participants until the commit decision has reached all participants.

5 Summary and Conclusion

We have presented two key ideas for guaranteeing atomicity for web-service transactions in a mobile context that also reduce the time of blocking and the number of aborts. The first idea is to use the suspend state for a transaction that has finished its read phase while the coordinator waits for the votes of other sub-transactions of the same global transaction. Being in suspend state, a sub-transaction can still be aborted by the resource manager if the resource manager decides to grant the resources used by this sub-transaction to other concurrent transactions to prevent them from blocking.

Secondly, for reducing the number of aborts in case of conflicts or missing votes, we identify those aborted sub-transactions that are repeatable or reusable, instead of only aborting and restarting all sub-transactions of the global transaction. Additionally, we have introduced the commit tree as a data structure that can be used to implement the coordinator's management of transaction atomicity for a dynamically changing set of sub-transactions.

We have embedded our atomic commit protocol in a web-service transactional model, the characteristics of which is that sub-transactions must not be known in advance. We have furthermore presented all key solutions as an extension to 2PC. Note however, that our contributions are applicable to a much broader set of protocols. For example, the extension of an atomic commit protocol by a non-blocking suspend state is not limited to 2PC, but appears to be compatible with a variety of other atomic commit protocols, e.g. [9], [10], or [11].

Finally, our protocol extension can nicely be combined with various concurrency control strategies including validation and locking. Although a proof of serializability for any schedule of concurrent transactions is beyond the scope of this paper, we have evidence that serializability can be guaranteed, and we plan to report about this on a forthcoming paper.

References

1. Curbera, F., Goland, Y., Klein, J., Leymann, F., et al.: Business Process Execution Language for Web Services, V1.0. Technical report, BEA, IBM, Microsoft (2002)
2. Arkin, A., et al.: Business process modeling language, bpmi.org. (Technical report)
3. Cabrera, L.F., Copeland, G., Feingold, M., et al.: Web Services Transactions specifications – Web Services Atomic Transaction. http://www-128.ibm.com/developerworks/library/specification/ws-tx/ (2005)
4. Gray, J.: Notes on data base operating systems. In Flynn, M.J., Gray, J., Jones, A.K., et al., eds.: Advanced Course: Operating Systems. Volume 60 of Lecture Notes in Computer Science., Springer (1978) 393–481
5. Kumar, V., Prabhu, N., Dunham, M.H., Seydim, A.Y.: Tcot-a timeout-based mobile transaction commitment protocol. IEEE Trans. Com. **51** (2002) 1212–1218
6. Dunham, M.H., Helal, A., Balakrishnan, S.: A mobile transaction model that captures both the data and movement behavior. Mobile Networks and Applications **2** (1997) 149–162
7. Object Management Group: Trans. Service Spec. 1.4. http://www.omg.org (03)
8. Skeen, D., Stonebraker, M.: A formal model of crash recovery in a distributed system. In: Berkeley Workshop. (1981) 129–142
9. Reddy, P.K., Kitsuregawa, M.: Reducing the blocking in two-phase commit with backup sites. Inf. Process. Lett. **86** (2003) 39–47
10. Gray, J., Lamport, L.: Consensus on transaction commit. Microsoft Research – Technical Report 2003 (MSR-TR-2003-96) **cs.DC/0408036** (2004)
11. Böse, J.H., Böttcher, S., Gruenwald, L., Obermeier, S., Schweppe, H., Steenweg, T.: An integrated commit protocol for mobile network databases. In: 9th International Database Engineering & Application Symposium IDEAS, Montreal, Canada (2005)
12. Kung, H.T., Robinson, J.T.: On optimistic methods for concurrency control. ACM Trans. Database Syst. **6** (1981) 213–226
13. Skeen, D.: Nonblocking commit protocols. In Lien, Y.E., ed.: Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data, Ann Arbor, Michigan, ACM Press (1981) 133–142
14. Pitoura, E., Bhargava, B.K.: Maintaining consistency of data in mobile distributed environments. In: Intl. Conf. on Distributed Computing Systems. (1995) 404–413
15. Dirckze, R.A., Gruenwald, L.: A toggle transact. management technique for mobile multidatabases. In: CIKM '98, New York, USA, ACM Press (1998) 371–377
16. Böttcher, S., Gruenwald, L., Obermeier, S.: An Atomic Web-Service Transaction Protocol for Mobile Environments. In: Proceedings of the 2nd EDBT Workshop on Pervasive Information Management, Munich, Germany (2006)

Query Translation for Distributed Heterogeneous Structured and Semi-structured Databases

Fahad M. Al-Wasil, N.J. Fiddian, and W.A. Gray

School of Computer Science
Cardiff University
Queen's Buildings
5 The Parade, Roath
Cardiff CF24 3AA
Wales, UK
{Wasil, N.J.Fiddian, W.A.Gray}@cs.cardiff.ac.uk

Abstract. The main purpose of building data integration systems is to facilitate access to a multitude of data sources. A data integration system must contain a module that uses source descriptions in order to reformulate user queries which are posed in terms of the composite global schema, into sub-queries that refer directly to the schemas of the component data sources. In this paper we propose a method for this user query translation task to target distributed heterogeneous structured data residing in relational databases and semi-structured data held in well-formed XML documents (XML documents which have no referenced DTD or XML schema) produced by Internet applications or human-coded. These XML documents can be XML files on local hard drives or remote documents on Web servers. Our method is based on mappings between the master (composite) view and the participating data source schema structures that are defined in a generated XML Metadata Knowledge Base (XMKB).

1 Introduction

Users and application programs in a wide variety of businesses today are increasingly requiring the integration of multiple distributed autonomous heterogeneous data sources [1, 2]. The continuing growth and widespread popularity of the Internet mean that the collection of useful data sources available for public access is rapidly increasing both in number and size. Furthermore, the value of these data sources would in many cases be greatly enhanced if the data they contain could be combined, "queried" in a uniform manner (i.e. using a single query language and interface), and subsequently returned in a machine-readable form. For the foreseeable future, much data will continue to be stored in relational database systems because of the reliability, scalability, tools and performance associated with these systems [3, 4]. However, due to the impact of the Web, there is an explosion in complementary data availability: this data can be automatically generated by Web-based applications and Web services or can be human-coded [5]. Such data is called semi-structured data (ssd) due to its varying degree of structure. In the domain of semi-structured data, the eXtensible Markup Language (XML) is a major data representation as well as data exchange format. XML is a W3C specification [6] that allows creation and

D. Bell and J. Hong (Eds.): BNCOD 2006, LNCS 4042, pp. 73 – 85, 2006.

transformation of a semi-structured document conforming to the XML syntax rules and having no referenced DTD or XML schema. Such a document has metadata buried inside the data and is called a well-formed XML document. The metadata content of XML documents enables automated processing, generation, transformation and consumption of semi-structured information by applications. Much interesting and useful data can be published as a well-formed XML document by Web-based applications and Web services or by human-coding.

Hence, building a data integration system that provides unified access to semantically and structurally diverse data sources is highly desirable to link structured data residing in relational databases and semi-structured data held in well-formed XML documents produced by Internet applications or human-coded [7, 8]. These XML documents can be XML files on local hard drives or remote documents on Web servers. The data integration system has to find structural transformations and semantic mappings that result in correct merging of the data and allow users to query the so-called mediated schema [9]. This linking is a challenging problem since the pre-existing databases concerned are typically autonomous and located on heterogeneous hardware and software platforms. In this context, it is necessary to resolve several conflicts caused by the heterogeneity of the data sources with respect to data model, schema or schema concepts. Consequently, mappings between entities from different sources representing the same real-world objects have to be defined. The main difficulty is that the related data from different sources may be represented in different formats and in incompatible ways. For instance, the bibliographical databases of different publishers may use different formats for authors' or editors' names (e.g. full name or separated first and last names), or different units for prices (e.g. dollars, pounds or euros). Moreover, the same expression may have a different meaning, or the same meaning may be specified by different expressions. This implies that syntactical data and metadata alone cannot provide sufficient semantics for all potential integration purposes. As a result, the data integration process is often very labour-intensive and demands more computing expertise than most application users have. Therefore, semi-automated approaches seem the most promising way forward, where mediation engineers are given an easy tool to describe mappings between the integrated (integrated and master are used interchangeably in this paper) view and local schemas, to produce a uniform view over all the participating local data sources [10].

XML is becoming the standard format to exchange information over the Internet. The advantages of XML as an exchange model - such as rich expressiveness, clear notation and extensibility - make it an excellent candidate to be a data model for the integrated schema. As the importance of XML has increased, a series of standards has grown up around it, many of which were defined by the World Wide Web Consortium (W3C). For example, the XML Schema language provides a notation for defining new types of XML elements and XML documents. XML with its self-describing hierarchical structure and the language XML Schema provide the flexibility and expressive power needed to accommodate distributed and heterogeneous data. At the conceptual level, they can be visualized as trees or hierarchical graphs.

In [11] we proposed and described a System to Integrate Structured and Semi-structured Databases (SISSD) through a mediation layer. Such a layer is intended to combine and query distributed heterogeneous structured data residing in relational databases with semi-structured data held in well-formed XML documents (that conform to the XML syntax rules but have no referenced DTD or XML schema) produced by Internet applications. We investigated how to establish and evolve an XML Metadata Knowledge Base (XMKB) incrementally to assist the Query Processor in mediating between user queries posed over the master view and the underlying distributed heterogeneous data sources. The XMKB is built in a bottom-up fashion by extracting and merging incrementally the metadata of the data sources. It contains and maintains data source information (names, types and locations), meta-information about relationships of paths among data sources, and function names for handling semantic and structural discrepancies. The associated SISSD system automatically creates a GUI tool for meta-users (who do the metadata integration) to describe mappings between the master view and local data sources by assigning index numbers and specifying conversion function names. From these mappings the SISSD produces the corresponding XML Metadata Knowledge Base (XMKB), which is capable of supporting the generation of queries to local data sources from user queries posed over the master view. The GUI tool parses the master view to generate an index number for each element and parses local schema structures to generate a path for each element. Mappings assign indices to match local elements with corresponding master elements and to names of conversion functions, which can be built-in or user-defined functions. The XMKB is derived based on the mappings by combination over index numbers.

We have proposed a generic mechanism to compute index numbers for the master view elements. By applying this mechanism, a unique index number is generated for each element in an XML document whatever the nesting complexity of the document. We have also described several mapping cases between master view and local schema structure elements (e.g. One-to-One, One-to-Many and Many-to-One) and how to resolve structural and semantic conflicts that may occur between elements.

This system is flexible: users can assemble any virtual master view they want from the same set of data sources depending on their interest. It also preserves local autonomy of the local data sources, thus these data sources can be handled without rebuilding or modification. The SISSD uses the local-as-view approach to map between the master view and the local schema structures. This approach is well-suited to supporting a dynamic environment, where data sources can be added to or removed from the system without the need to restructure the master view. The XML Metadata Knowledge Base (XMKB) is evolved and modified incrementally when data sources are added to or removed from the system, without the need to regenerate it from scratch.

This paper concentrates on the problem of querying a multiplicity of distributed heterogeneous structured data residing in relational databases and semi-structured data held in well-formed XML documents. The important aspect is to develop a technique to seamlessly translate user queries over the master view into sub-queries - called local queries - fitting each participating data source, by exploiting the mapping information stored in the XMKB.

User queries are formulated in XQuery (a powerful universal query language for XML) FLWR (short for For-Let-Where-Return) expressions and processed according to the XMKB, by generating an executable (sub-) query for each relevant local data source.

The rest of this paper is organized as follows. The next section presents related work. The SISSD architecture and the internal architecture of its Query Processor (QP) are described in section 3. Section 4 presents the structure, content and organization of knowledge in the XMKB. Section 5 summarises the query translation process in algorithmic form. Finally, we present conclusions in section 6.

2 Related Work

Data integration has received significant attention since the early days of databases. In recent years, there have been several projects focusing on heterogeneous information integration. Most of them are based on a common mediator architecture [12] such as Garlic [13], the Information Manifold [14], Disco [15], Tsimmis [16], Yat [17], Mix [18], MedMaker [19] and Agora [20]. The goal of such systems is to provide a uniform user interface to query integrated views over heterogeneous data sources. A user query is formulated in terms of the integrated view; to execute the query, the system translates it into sub-queries expressed in terms of the local schemas, sends the sub-queries to the local data sources, retrieves the results, and combines them into the final result provided to the user. Data integration systems can be classified according to the way the schemas of the local data sources are related to the global, unified schema. A first approach is to define the global schema as a view over the local schemas: such an approach is called global-as-view (GAV). The opposite approach, known as local-as-view (LAV), consists of defining the local sources as views over the global schema [21].

Now consider query processing. In the GAV approach, translating a query over the global schema into queries against the local schemas is a simple process of view unfolding. In the case of LAV, the query over the global schema needs to be reformulated in terms of the local data source schemas; this process is traditionally known as "rewriting queries using views" and is a known hard problem [22].

Projects like Garlic, Disco, Tsimmis, Mix, MedMaker and Yat all adopt the GAV approach, and therefore do not compare directly to our system since we use the LAV approach. Projects like the Information Manifold and Agora are integration systems with a LAV architecture; however, in the Information Manifold the local and global schemas are relational, while the Agora system supports querying and integrating data sources of diverse formats, including XML and relational sources under an XML global schema, but assumes explicit schemas for XML data sources.

SilkRoute [23] and XPERANTO [4, 24] focus on exporting relational databases under an XML interface. Since the mapping is done from tuples to XML, these projects adopt the GAV approach; also, they can only integrate relational data sources. By contrast, our integration approach can handle diverse data sources (XML and relational), not just relational. Also SISSD follows the Information Manifold and Agora systems by adopting the LAV approach.

The LAV approach provides a more flexible environment to meet users' evolving and changing information requirements across the disparate data sources available over the global information infrastructure (Internet). It is better suited and scalable for

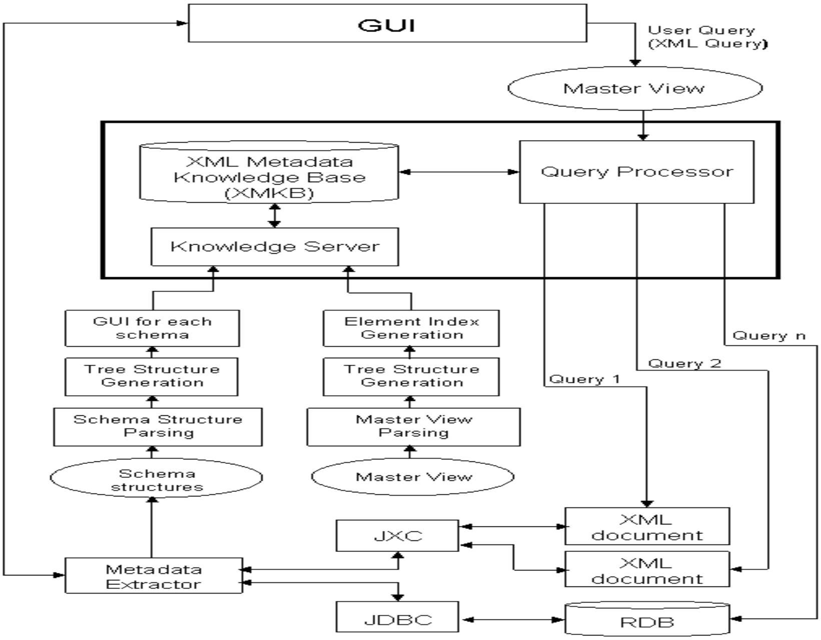


Fig. 1. The SISSD Architecture

integrating a large number of autonomous read-only data sources accessible over communication networks. Furthermore the LAV approach provides a flexible environment able to accommodate the continual change and update of data source schemas, especially suitable for XML documents on Web servers since these remote documents are not static and are often subject to frequent update.

3 The SISSD Architecture and Components

In this section, we present an overview of the SISSD architecture and summarize the functions of the main components. The architecture we adopt is depicted in Figure 1. Its main components are the Metadata Extractor (MDE), the Knowledge Server (KS) and the Query Processor (QP).

3.1 Metadata Extractor (MDE)

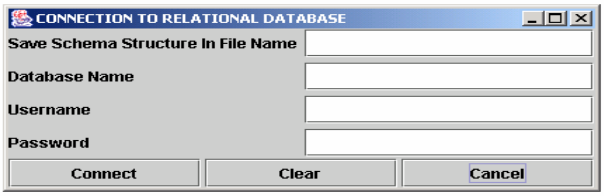
The MDE needs to deal with heterogeneity at the hardware, software and data model levels without violating the local autonomy of the data sources. It interacts with the data sources via JDBC (Java Database Connectivity) if the data source is a relational database or via JXC (Java XML Connectivity) if the data source is an XML document. The MDE extracts the metadata of all data sources and builds a schema structure in XML form for each data source.

We developed JXC using a JDOM (Java Document Object Model) interface to detect and extract the schema structure of a well-formed XML document (that conforms to the XML syntax rules but has no referenced DTD or XML schema), where the metadata are buried inside the data.

3.1.1 Schema Structures

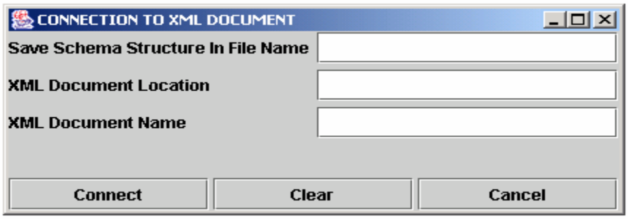
Typically, the heterogeneous data sources use different data models to store their data (e.g. relational model and XML model). This type of heterogeneity is referred to as syntactic heterogeneity. The solution commonly adopted to overcome syntactic heterogeneity is to use a common data model and to map all schemas to this common model. The advantages of XML as an exchange model make it a good candidate to be the common data model and for supporting the integrated data model. The metadata extracts generated on top of the data sources by using this data model are referred to as schema structures. We define a simple XML Data Source Definition Language (XDSDL) for describing and defining the relevant identifying information and the data structure of a data source. The XDSDL is represented in XML and is composed of two parts. The first part provides a description of the data source name, location and type (relational database or XML document). The second part provides a definition and description of the data source structure and content. The emphasis is on making these descriptions readable by automated processors such as parsers and other XML-based tools. This language can be used for describing the structure and content of relational databases and well-formed XML documents which have no referenced DTD or XML schema.

For relational databases the MDE employs JDBC to access the DB without making any changes to it. The MDE accepts the information necessary to establish a connection to a DB to retrieve the metadata of its schema and uses the XDSDL to build the target schema structure for that DB, together with necessary information such as the DB location (URL), where to save the schema structure, the User ID and Password.


CONNECTION TO RELATIONAL DATABASE
Save Schema Structure In File Name
Database Name
Username
Password
Connect
Clear
Cancel

It opens a connection to that DB through a JDBC driver. Opening this connection enables SQL queries to be issued to and results to be retrieved from the DB. Once the connection is established, the MDE retrieves the names of all the tables defined in the accessed DB schema and then uses the XDSDL to define these tables as elements in the target schema structure. Furthermore, for each table the MDE extracts and analyses the attribute names, then defines these attributes as child elements for that table element in the target schema structure using the XDSDL.

For XML documents the MDE employs JXC to access the document without making any changes to it. The MDE accepts the information necessary to establish a connection to a well-formed XML document to retrieve the metadata of its schema where the metadata are buried inside the data. It then uses the XDSDL to build the target schema structure for that XML document, together with necessary information such as the document location (URL), where to save the schema structure, and the document name.


CONNECTION TO XML DOCUMENT
Save Schema Structure In File Name
XML Document Location
XML Document Name
Connect
Clear
Cancel

It opens a connection to that XML document through a JDOM interface. Once the connection is established, the JXC automatically tracks the structure of the XML document, viz. each element found in the document, which elements are child elements and the order of child elements. The JXC reads the XML document and detects the start tag for the elements. For each start tag, the JXC checks if this element has child elements or not: if it has then this element is defined as a complex element in the target schema structure using the XDSDL, otherwise it is defined as a simple element by the MDE. The defined elements in the target schema structure take the same name as the start tags.

3.2 Knowledge Server (KS)

The Knowledge Server (KS) is the central component of the SISSD. Its function is to establish, evolve and maintain the XML Metadata Knowledge Base (XMKB), which holds information about the data sources and provides the necessary functionality for its role in assisting the Query Processor (QP). The KS creates a GUI tool for meta-users to do metadata integration by building the XML Metadata Knowledge Base (XMKB) that comprises information about data structures and semantics. This information can then be used by the Query Processor (QP) to automatically rewrite a user query over the master view into sub-queries called local queries, fitting each local data source, and to integrate the results.

3.3 Query Processor (QP)

The Query Processor (QP) is responsible for receiving a user query (master query) over a master view to process it and return the query result to the user. The master view provides the user with the elements on which the query can be based. The QP gives flexibility to the user to choose the master view that he/she wants to pose his/her query over and then automatically selects the appropriate XMKB that will be used to process any query posed over this master view. The query language that our QP supports is XQuery FLWR expressions. XQuery is the standard XML query language being developed by the W3C [25]. It is derived from Quilt, an earlier XML query language designed by Jonathan Robie together with IBM's Don Chamberlin, co-inventor of SQL, and Daniela Florescu, a well-known database researcher. XQuery is

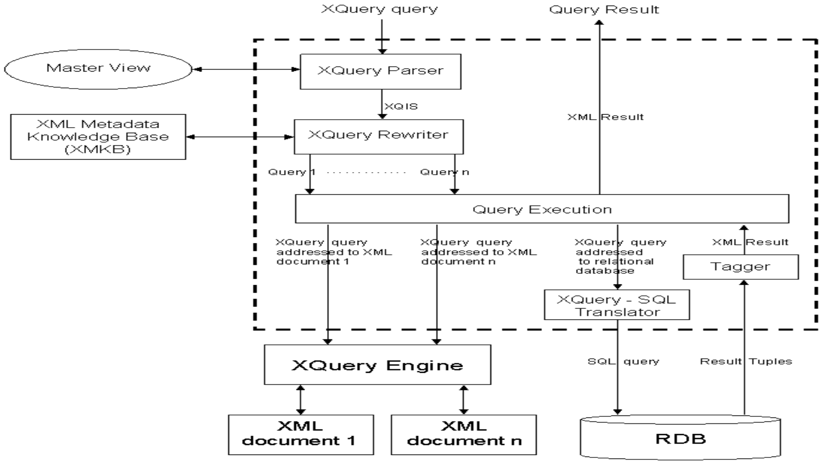


Fig. 2. Internal Architecture of the Query Processor

designed to be a language in which queries are concise and easily understood. It is also flexible enough to query a broad spectrum of XML information sources, including both databases and documents. It can be used to query XML data that has no schema at all, or that is governed by a W3C standard XML Schema or by a Document Type Definition (DTD).

XQuery is centered on the notion of expression; starting from constants and variables, expressions can be nested and combined using arithmetic, logical and list operators, navigation primitives, function calls, higher order operators like sort, conditional expressions, element constructors, etc. For navigating in a document, XQuery uses path expressions, whose syntax is borrowed from the abbreviated syntax of XPath. The evaluation of a path expression on an XML document returns a list of information items, whose order is dictated by the order of elements within the document (also called document order).

A powerful feature of XQuery is the use of FLWR expressions (For-Let-Where-Return). The *for-let* clause makes variables iterate over the result of an expression or binds variables to arbitrary expressions, the *where* clause allows specification of restrictions on the variables, and the *return* clause can construct new XML elements as output of the query. In general, an XQuery query consists of an optional list of namespace definitions, followed by a list of function definitions, followed by a single query expression.

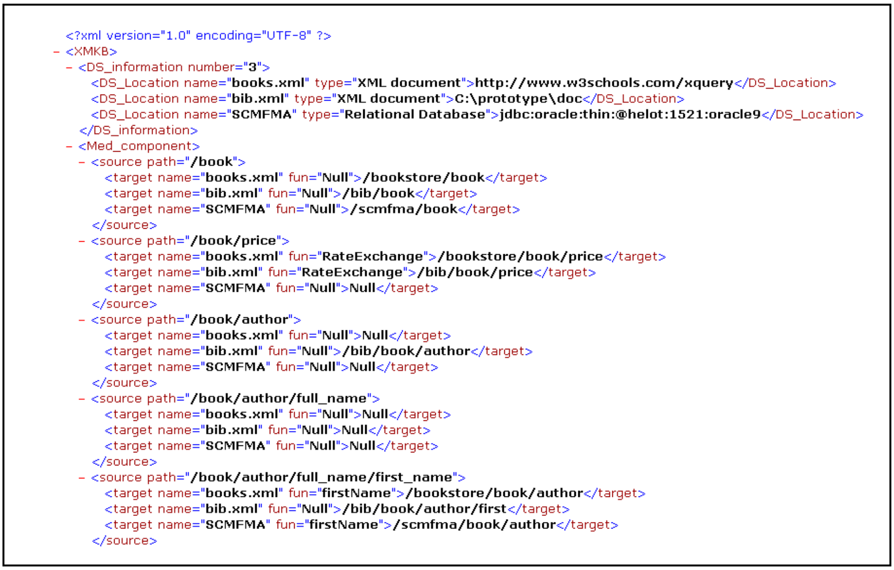
Supporting FLWR expressions for querying a master view makes it easy to translate the sub-queries directed at relational databases into SQL queries since syntactically, FLWR expressions look similar to SQL select statements and have similar capabilities, only they use path expressions instead of table and column names.

The internal architecture of the Query Processor (QP) is shown in Figure 2. It consists of five components: XQuery Parser, XQuery Rewriter, Query Execution, XQuery-SQL Translator, and Tagger. The core of the QP and the primary focus of this paper is the XQuery Rewriter. This component rewrites the user query posed over the master view into sub-queries which fit each local data source, by using the mapping information stored in the XMKB. The main role played by each of the components in Figure 2 is described below.

- **XQuery Parser:** parses a given XQuery FLWR expression in order to check it for syntactic correctness and ensure that the query is valid and conforms to the relevant master view. Also the parser analyses the query to generate an XQuery Internal Structure (XQIS) which contains the XML paths, variables, conditions and tags present in the query, then passes it to the XQuery Rewriter.
- **XQuery Rewriter:** Takes the XQIS representation of a query, consults the XMKB to obtain the local paths corresponding to the master paths and function names for handling semantic and structural discrepancies, then produces semantically equivalent XQuery queries to fit each local data source. That is, wherever there is a correspondence between the paths in the master view and local schema structures concerned (otherwise the local data source is ignored).
- **Query Execution:** Receives the rewritten XQuery queries, consults the XMKB to determine each data source's location and type (relational database or XML document), then sends each local query to its corresponding query engine, to execute the query and return the results.
- **XQuery-SQL Translator:** Translates an XQuery query addressed to a relational database into the SQL query needed to locate the result, then hands the query over to the relational database engine to execute it and return the result in tabular format through the Tagger.
- **Tagger:** Adds the appropriate XML tags to the tabular SQL query result to produce structured XML documents for return to the user.

4 The Structure of the XMKB

The XML Metadata Knowledge Base (XMKB) is an XML document composed of two parts. The first part contains information about data source names, types and locations. The second part contains meta-information about relationships of paths among data sources, and function names for handling semantic and structural discrepancies. The XMKB structure is shown in Figure 3. The *<DS_information>* element here contains data source names, types and locations. The *<DS_information>* element has one attribute called number which holds the number of data sources participating in the integration system (3 in the example shown). Also the *<DS_information>* element has child elements called *<DS_Location>*. Each *<DS_Location>* element contains the data source name, its type (relational database or XML document) as an attribute value and the location of the data source as an element value. This information is used by the Query Processor to specify the type of generated sub-query (SQL if the data source type is relational database, or XQuery if it is XML document) and the data source location that the system should submit the generated sub-query to.



```
  <?xml version="1.0" encoding="UTF-8" ?>
- <XMKB>
  - <DS_information number="3">
      <DS_Location name="books.xml" type="XML document">http://www.w3schools.com/xquery</DS_Location>
      <DS_Location name="bib.xml" type="XML document">C:\prototype\doc</DS_Location>
      <DS_Location name="SCMFMA" type="Relational Database">jdbc:oracle:thin:@helot:1521:oracle9</DS_Location>
    </DS_information>
  - <Med_component>
    - <source path="/book">
        <target name="books.xml" fun="Null">/bookstore/book</target>
        <target name="bib.xml" fun="Null">/bib/book</target>
        <target name="SCMFMA" fun="Null">/scmfma/book</target>
      </source>
    - <source path="/book/price">
        <target name="books.xml" fun="RateExchange">/bookstore/book/price</target>
        <target name="bib.xml" fun="RateExchange">/bib/book/price</target>
        <target name="SCMFMA" fun="Null">Null</target>
      </source>
    - <source path="/book/author">
        <target name="books.xml" fun="Null">Null</target>
        <target name="bib.xml" fun="Null">/bib/book/author</target>
        <target name="SCMFMA" fun="Null">Null</target>
      </source>
    - <source path="/book/author/full_name">
        <target name="books.xml" fun="Null">Null</target>
        <target name="bib.xml" fun="Null">Null</target>
        <target name="SCMFMA" fun="Null">Null</target>
      </source>
    - <source path="/book/author/full_name/first_name">
        <target name="books.xml" fun="firstName">/bookstore/book/author</target>
        <target name="bib.xml" fun="Null">/bib/book/author/first</target>
        <target name="SCMFMA" fun="firstName">/scmfma/book/author</target>
      </source>
```

Fig. 3. A sample XMKB document

The <*Med_component*> element in Figure 3 contains the path mappings between the master view elements and the local data source elements, and the function names for handling semantic and structural discrepancies. The master view element paths are called <*source*> elements, while corresponding element paths in local data sources are called <*target*> elements. The <*source*> elements in the XMKB document have one attribute called path which contains the path of the master view elements. These <*source*> elements have child elements called <*target*> which contain the corresponding paths for the master view element paths in each local data source, or null if there is no corresponding path. The <*target*> elements in the XMKB document have two attributes. The first one is called *name* and contains the name of the local data source, while the second is called *fun* and contains the function name that is needed to resolve semantic and structural discrepancies between the master view element and the local data source element concerned, or null if there is no discrepancy.

5 The Query Translation Process

From the foregoing descriptions of the SISSD Query Processor (QP) component architecture (section 3.3) and the XML Metadata Knowledge Base (XMKB) organization and contents (section 4), we are now in a position to summarise the query translation (rewriting) process carried out at the heart of our system by the QP module. We do so in algorithmic form as follows, c.f. Figure 2 earlier. The algorithm is both conceptually simple and generally applicable. We have successfully implemented and tested it on a variety of relational and XML data source integration examples in our prototype SISSD system.

Algorithm. Master query translation process

Input: Master View, Master XQuery query *q*, and XMKB

Output: local sub-queries *q1*, *q2…*, *qn*

```
Step1: parse q;
Step2: get global paths g1, g2…., gn from Master View;
Step3: read XMKB;
Step4: identify the number of local data sources participating in the integration
       system, their locations and types;
Step5: for each data source Si do
           for each global path ge in q do
               if the corresponding local path le not null then
                   get le;
                   if the function name fe not null then
                       get fe;
                   end if
               else
                   no query generated for this local data source Si ;
               end if
           end for
           replace g1 by l1 with f1, g2 by l2 with f2 ..., gn by ln with fn, in qi;
           if data source type is relational database then
               convert qi XQuery into SQL;
       end for
Step6: execute the generated local query qi by sending it to the corresponding local
     data source engine, and return the result, with XML tags added to SQL tables.
```

6 Conclusions

In this paper, we have described an approach for querying a multiplicity of distributed heterogeneous structured data residing in relational databases and semi-structured data held in well-formed XML documents (XML documents which have no referenced DTD or XML schema) produced by Internet applications or human-coded. These XML documents can be XML files on local hard drives or remote documents on Web servers. Our method is based on mappings between the master view and the participating data source schema structures that are defined in a generated XML Metadata Knowledge Base (XMKB). The basic idea is that a query posed to the integrated system, called a master query, is automatically rewritten into sub-queries called local queries which fit each local data source, using the information stored in the XMKB. This task is accomplished by the Query Processor module. Such an approach produces a system capable of querying across a set of heterogeneous distributed structured and semi-structured data sources. We have developed a prototype system to demonstrate that the ideas explored in the paper are sound and practical, also clearly convenient from a user standpoint.

As a result, our system can easily incorporate a large number of relational databases and XML data sources from the same domain. However, most of the existing data integration systems concerned with XML documents are interested in documents that use DTD (Document Type Definition) or XML Schema language for describing the schemas of the participating heterogeneous XML data sources. We have investigated and used XML documents which have no referenced DTD or XML schema, rather the schema metadata are buried inside the document data. This paper has shown that querying a set of distributed heterogeneous structured and semi-structured data sources of this form and in this way is possible; its relevance in the Internet/Web context is readily apparent.

In addition to this, our Query Processor (QP) has been implemented using Java, JDOM API, and the JavaCC compiler. It accepts FLWR expressions as an XML query language, this is a subset of XQuery which supports the basic requirements of our approach, particularly the uniform querying of heterogeneous distributed structured (relational database) and semi-structured (well-formed XML document) data sources.

References

1. Hu, G. and H. Fernandes, Integration and querying of distributed databases, in Proceedings of the IEEE International Conference on Information Reuse and Integration (IRI 2003), October 27-29, 2003: Las Vegas, NV, USA. p. 167-174.
2. Segev, A. and A. Chatterjee, *Data manipulation in heterogeneous databases.* Sigmod Record, December 1991. **20(4)**: p. 64-68.
3. Funderburk, J.E., et al., *XTABLES: Bridging Relational Technology and XML.* IBM Systems Journal, 2002. **41(4)**: p. 616-641.
4. Shanmugasundaram, J., et al., Efficiently Publishing Relational Data as XML Documents, in Proceedings of the 26th International Conference on Very Large Databases (VLDB2000), September 2000: Cairo, Egypt. p. 65-76.
5. Lehti, P. and P. Fankhauser, XML data integration with OWL: Experiences & challenges, in Proceedings of the International Symposium on Applications and the Internet (SAINT 2004), 2004: Tokyo, Japan. p. 160-170.
6. World Wide Web Consortium, *http://www.w3.org/TR/2004/REC-xml-20040204/.* Extensible Markup Language (XML) 1.0 W3C Recommendation, third edition, February 2004.
7. Gardarin, G., F. Sha, and T. Dang-Ngoc, XML-based Components for Federating Multiple Heterogeneous Data Sources, in ER '99: Proceedings of the 18th International Conference on Conceptual Modeling, 1999, Springer-Verlag. p. 506-519.
8. Lee, K., J. Min, and K. Park, A Design and Implementation of XML-Based Mediation Framework (XMF) for Integration of Internet Information Resources, in HICSS '02: Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02) - Volume 7. 2002, IEEE Computer Society. p. 202-210.
9. Kurgan, L., W. Swiercz, and K. Cios, Semantic Mapping of XML Tags using Inductive Machine Learning, in Proceedings of the International Conference on Machine Learning and Applications - ICMLA '02. 2002: Las Vegas, Nevada, USA.

10. Young-Kwang, N., G. Joseph, and W. Guilian, A Metadata Integration Assistant Generator for Heterogeneous Distributed Databases, in Proceedings of the Confederated International Conferences DOA, CoopIS and ODBASE. October 2002, LNCS 2519, Springer, p. 1332-1344.: Irvine CA.
11. Al-Wasil, F.M., W.A. Gray, and N.J. Fiddian, Establishing an XML Metadata Knowledge Base to Assist Integration of Structured and Semi-structured Databases, in ADC '2006: Proceedings of The Seventeenth Australasian Database Conference. January 16th - 19th 2006: Tasmania, Australia.
12. Wiederhold, G., *Mediators in the Architecture of Future Information System.* IEEE Computer, March 1992. **25(3)**: p. 38-49.
13. Carey, M.J., et al., *Towards heterogeneous multimedia information systems: the Garlic approach*, in *RIDE '95: Proceedings of the 5th International Workshop on Research Issues in Data Engineering-Distributed Object Management (RIDE-DOM'95).* 1995, IEEE Computer Society. p. 124-131.
14. Kirk, T., et al., The Information Manifold, in Proceedings of the AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments, p. 85-91. March, 1995.: Stanford University, Stanford, CA.
15. Tomasic, A., L. Raschid, and P. Valduriez, *Scaling access to heterogeneous data sources with DISCO.* IEEE Transactions on Knowledge and Data Engineering, 1998. **10**(5): p. 808-823.
16. Ullman, J.D., Information Integration Using Logical Views, in ICDT '97: Proceedings of the 6th International Conference on Database Theory. 1997, Springer-Verlag. p. 19-40.
17. Christophides, V., S. Cluet, and J. Simèon, On wrapping query languages and efficient XML integration, in Proceedings of ACM SIGMOD Conference on Management of Data. May 2000.: Dallas, Texas, USA.
18. Baru, C., et al., XML-based information mediation with MIX, in SIGMOD '99: Proceedings of ACM SIGMOD International Conference on Management of Data. 1999, ACM Press. p. 597-599.
19. Papakonstantinou, Y., H. Garcia-Molina, and J.D. Ullman, MedMaker: A Mediation System Based on Declarative Specifications, in ICDE '96: Proceedings of the 12th International Conference on Data Engineering. 1996, IEEE Computer Society. p. 132-141.
20. Manolescu, I., D. Florescu, and D. Kossmann, Answering XML Queries over Heterogeneous Data Sources, in Proceedings of the 27th International Conference on Very Large Data Bases (VLDB). September 2001: Rome, Italy.
21. Lenzerini, M., Data integration: a theoretical perspective, in Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. 2002: Madison, Wisconsin.
22. Levy, A., et al., Answering queries using views, in Proceedings of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems. 1995: San Jose, CA, USA.
23. Fernndez, M., W.-C. Tan, and D. Suciu, SilkRoute: Trading between Relations and XML, in Proceedings of the Ninth International World Wide Web Conference. May 15 - 19 2000: Amsterdam.
24. Shanmugasundaram, J., et al., Querying XML Views of Relational Data, in proceedings of the 27th International Conference on Very Large Data Bases (VLDB). September 2001: Rome, Italy.
25. World Wide Web Consortium, *http://www.w3.org/TR/xquery/.* XQuery 1.0: An XML Query Language, W3C Working Draft, November 2003.

Information Retrieval Evaluation with Partial Relevance Judgment

Shengli Wu and Sally McClean

School of Computing and Mathematics
University of Ulster, Northern Ireland, UK
{s.wu1, si.mcclean}@ulster.ac.uk

Abstract. Mean Average Precision has been widely used by researchers in information retrieval evaluation events such as TREC, and it is believed to be a good system measure because of its sensitivity and reliability. However, its drawbacks as regards partial relevance judgment has been largely ignored. In many cases, partial relevance judgment is probably the only reasonable solution due to the large document collections involved.

In this paper, we will address this issue through analysis and experiment. Our investigation shows that when only partial relevance judgment is available, mean average precision suffers from several drawbacks: inaccurate values, no explicit explanation, and being subject to the evaluation environment. Further, mean average precision is not superior to some other measures such as precision at a given document level for sensitivity and reliability, both of which are believed to be the major advantages of mean average precision. Our experiments also suggest that average precision over all documents would be a good measure for such a situation.

1 Introduction

Since the beginning of information retrieval research, the evaluation issue has been paid considerable attention because of its complexity. Recall (the fraction of all the relevant documents which are retrieved) and precision (the fraction of the retrieved documents which are relevant) are considered by many researchers as the two most important (but very different) aspects [6, 8, 12]. Using a single value measure for a comprehensive consideration of these two aspects is an attractive opinion [13]. Some such measures have been proposed: Borko's BK measure [12], Vickery's Q and V measures [12]. van Rijsbergen's E measure [12], the harmonic mean by Shaw, Burgin, and Howell [10], and cumulated gain by J*ä*velin and Kek*ä*l*ä*inen [5, 7], etc. However, most of them have not been used widely.

One exception to this is average precision over all relevant documents, which has been referred to as mean average precision recently. Mean average precision has been used in Text REtrieval Conferences[1] [11] since 1994 (TREC 3) and now it is widely used by many researchers to evaluate their systems, algorithms, etc.

[1] TREC is a major event for the evaluation of information retrieval. Since 1992, it has been held yearly by the National Institute of Standards and Technology of USA and USA Department of Defence.

D. Bell and J. Hong (Eds.): BNCOD 2006, LNCS 4042, pp. 86–93, 2006.

Some previous research [1, 2, 9, 14, 16] suggests that mean average precision is a good system measure for several reasons. First, it is a single value measure therefore convenient for use, especially for comparing the performances of several different information retrieval systems. Second, it is sensitive since its calculation uses the complete information of relevant documents: the total number of relevant documents in the whole document collection and the ranked positions of them in a resultant list. Third, it is reliable. The reason for this is the same as for the second point.

Compared with mean average precision, precision at a given document level is quite different and is believed to be a good user-oriented measure. First, very often users' major concern is how many relevant documents exist in the top k (say, 5 or 10) documents. Second, it is very convenient for evaluation and requires much less effort than mean average precision does. Third, its value is explicit and easy to understand, while a mean average precision value is abstract and cannot be explained explicitly.

The above conclusion on mean average precision should be true if all the relevance judgment information is available. However, this is not the case in some situations. For example, in TREC, a pooling strategy is used. For every information need statement (topic) the top 100 documents in all or some submitted runs are put in the pool. Only those documents in the pool are judged by human assessors and all the documents which are not in the pool are unjudged and assumed to be irrelevant. Therefore, many relevant documents may be missed using such a pooling strategy [18]. Results from a Web search service is another situation where complete relevance judgment is impossible. However, the harmful effect of the incompleteness of relevance judgment information on mean average precision has not been discussed.

In this paper we would like to investigate this issue through analysis and experimentation with TREC data. The analysis and experiments will reveal some drawbacks of mean average precision for incomplete relevance judgment besides the one already known – only obscure explanation available for any mean average precision value.

Furthermore, a new measure is introduced and investigated in this paper. It is average precision over all documents. It will be demonstrated in this paper that this measure has the advantages of both mean average precision and average precision at a given document level, but does not have some shortcomings of mean average precision.

2 Two Measures

Mean average precision has been used by TREC in TREC 3 and onwards [11]. Since then, mean average precision has been widely used by researchers to evaluate their information retrieval systems and algorithms. It uses the following formula:

$$map = \frac{1}{n}\sum_{i=1}^{n}\frac{i}{r_i} \tag{1}$$

where n is the total number of relevant documents in the whole collection for that information need and r_i is the ranking position of the i-th relevant document in the list. For example, suppose there are 4 relevant documents for a topic, and these relevant documents are ranked in number 1, 4, 10, and 12 in a result, then this result's mean average precision is (1/1+2/4+3/10+4/12)/4=0.525.

Precision at a given document level is not very sensitive because it does not consider the positions of the relevant documents involved. For example, a relevant document appearing in rank 1 and in rank 100 has the same effect on precision at the 100 document level.

A new measure, average precision over all documents, is introduced in this paper. It can be a better choice than precision at a given document level since it concerns with the positions of relevant documents. It uses the following formula to calculate scores:

$$ap\_all(m) = \frac{1}{m} \sum_{i=1}^{m} \frac{r(i)}{i} \qquad (2)$$

Where $r(i)$ is the number of relevant documents in the top i documents and m is the total number of documents considered. Comparing Formula 1 and Formula 2, they bear some similarities.

If a document in rank j is relevant, then its contribution to the final score is $ap\_all(j, m) = \frac{1}{m} \sum_{i=j}^{m} \frac{1}{i}$. $H(m) = \sum_{i=1}^{m} \frac{1}{i}$ is a Harmonic number [4], which has some interesting characteristics. Let us consider $ap\_all'(j, m) = ap\_all(j, m) * m = \sum_{i=j}^{m} \frac{1}{i}$, which is a tail of a Harmonic number and we have $ap\_all'(j, m) = H(m) - H(j-1)$. Actually, the measure of discounted cumulated gain proposed by Jävelin and J. Kekäläinen [5] is a "general" measure of weighting schemas. Also they suggested a weighting schema: 1 for rank 1 and 2, 1/2 for rank 3, 1/3 for rank 4,..... Average precision over all documents can be regarded as a specialised form of discounted cumulated gain. In the remainder of this paper, we will focus on average precision over all documents and will not discuss other weighting schema variations.

3 Experiments

Experimental results using TREC data are reported in this section. We hope this can help us to obtain a better understanding about these measures. Compared with previous work [2, 3, 9, 14, 16], our experiments have different goals: we would like to find out how mean average precision perform when only incomplete relevance judgment information is available, and we also would like to investigate the new measure introduced in this paper – average precision over all documents.

3.1 Experimental Setting

9 groups of results (TREC 5, 6, 7, and 8: ad hoc track; TREC 9, 2001, and 2002: web track; TREC 2003 and 2004: robust track) submitted to TREC ad hoc, web

and robust track are used in the experiments. Three measures, mean average precision, average precision over all documents, and precision at 10 document level, are used in the experiment. In order to eliminating the effect of pooling, only the top 100 documents are used for the evaluation of all the involved results.

3.2 Error Rates Using Different Measures

First we carry out an experiment to investigate the stability and sensitivity of different measures. For a given measure, we evaluate all the results in a year group and obtain the average performance of them. Then for those pairs whose performance difference is above 5%, we check if this is true for all the topics. Suppose we have two results A and B such that A's average performance is better than B's average performance by over 5% in all l topics. Then we consider these l topics one by one. We find that A is better than B by over 5% for m queries, and B is better than A by over 5% for n queries ($l \geq m + n$). In this case the error rate is $n/(m + n)$.

The result of this experiment is shown in Table 1. On average, average precision over all document levels (*ap_all*) is the best, precision at 10 document level (p10) is in the second place, while mean average precision (*map*) is the worst. The differences between these measures are not big (p10-map: 2.69%, ap_all-map: 3.86%, and ap_all-p10: 1.13%).

On the other hand, when using mean average precision, more pairs are selected than when using the two other measures. This suggests that mean average precision is more sensitive than the two others. However, the difference is not large here either (map-p10: 3.68% and map-ap_all: 3.57%). Our experimental results suggest that these three measures are close in sensitivity and stability.

A similar experiment was carried out by Buckley and Voorhees [2]. They used all results submitted to the TREC 8 query track and tested the stability of several measures over different query formats. The experimental result reported here is consistent with that of Buckley and Voorhees's [2] though the experimental settings are different. In their experiment, they considered the top 1000 documents for every result and they found that precision at 1000 document level has lower error rates than mean average precision, while precision at 10 and 30 document levels have higher error rates than mean average precision. This suggests that precision at certain document level can be as good as mean average precision if the same number of documents are used.

3.3 Correlation Among Different Measures

Our second experiment aims to investigate how similar or different these measures are. Given a group of results, we use different measures to evaluate them and rank them based on their performances. Then we compare those rankings generated by using different measures. The experimental result is shown in Table 2. Both Spearman and Kendall's tau ranking coefficients are calculated. In table 2, all Kendall's tau ranking coefficient values are lower than the corresponding Spearman coefficient values, though the difference does not affect their relative rankings in most cases.

Table 1. Error rates of using different measures (numbers in parentheses are numbers of compared pairs)

Group	map	p10	ap_all
TREC 5	0.2731(1657)	0.2771(1694)	0.2710(1631)
TREC 6	0.2559(2262)	0.2650(2317)	0.2549(2257)
TREC 7	0.2451(4707)	0.2481(4837)	0.2429(4716)
TREC 8	0.2270(7096)	0.2304(7375)	0.2255(7119)
TREC 9	0.2351(5116)	0.2421(5114)	0.2315(5028)
TREC 2001	0.2839(4147)	0.2936(4261)	0.2798(4114)
TREC 2002	0.2641(2331)	0.2739(2374)	0.2595(2343)
TREC 2003	0.3006(2315)	0.3239(2478)	0.2916(2347)
TREC 2004	0.3223(4616)	0.3184(5053)	0.3241(4724)
Average	0.2675(3805)	0.2747(3945)	0.2645(3809)

Table 2. Correlation among rankings generated using different measures (S for Spearman coefficient and K for Kendall's tau coefficient)

Group	map vs. ap_all		map vs. p10		ap_all vs. p10	
	S	K	S	K	S	K
TREC 5	0.9683	0.8656	0.9628	0.8546	0.9822	0.9060
TREC 6	0.9551	0.8342	0.9482	0.8149	0.9773	0.8954
TREC 7	0.9754	0.8759	0.9523	0.8233	0.9797	0.8942
TREC 8	0.9710	0.8697	0.9466	0.8241	0.9807	0.8934
TREC 9	0.9689	0.8579	0.9526	0.8176	0.9851	0.9011
TREC 2001	0.9701	0.8621	0.9302	0.7934	0.9685	0.8565
TREC 2002	0.9243	0.7835	0.9538	0.8157	0.9036	0.7730
TREC 2003	0.9443	0.8069	0.8512	0.6830	0.8689	0.7362
TREC 2004	0.9800	0.8902	0.9460	0.8202	0.9588	0.8445
Ave.	0.9619	0.8496	0.9382	0.8052	0.9594	0.8556

The rankings generated using these three measures are strongly correlated with each other. On average the correlation is above 0.8 (Kendall's tau coefficient) or 0.9 (Spearman coefficient). In addition, the rankings generated using average precision over all documents are almost equally and very strongly correlated to the rankings generated using either of the two other measures, while the ranking correlation between precision at 10 document level and mean average precision is weaker.

3.4 Effect of Environment on Results Evaluation and Ranking

To evaluate information retrieval results using mean average precision demands much more efforts than using some other measures such as precision at the 10 or 100 document level, mainly because all relevant documents need to be identified. If complete relevance judgment is not available, then the performance of a result on mean average precision will depend on the relevant documents detected to a certain

Table 3. Correlation of rankings using full sets of results and rankings using partial sets of results (the numbers in parentheses indicate the performance difference of the same result in different environments)

Group	20%	40%	60%	80%
TREC 5	0.9606 (18.15%)	0.9724 (9.67%)	0.9832 (6.18%)	0.9867 (3.47%)
TREC 6	0.9582 (16.00%)	0.9793 (8.40%)	0.9905 (4.92%)	0.9941 (2.78%)
TREC 7	0.9768 (15.21%)	0.9870 (7.22%)	0.9930 (4.20%)	0.9963 (1.96%)
TREC 8	0.9690 (11.49%)	0.9833 (6.39%)	0.9922 (2.95%)	0.9968 (1.49%)
TREC 9	0.9714 (9.47%)	0.9934 (2.75%)	0.9944 (1.10%)	0.9970 (0.67%)
TREC 2001	0.9602 (15.40%)	0.9738 (7.54%)	0.9852 (3.86%)	0.9900 (2.02%)
TREC 2002	0.9604 (16.84%)	0.9810 (7.39%)	0.9856 (3.80%)	0.9879 (2.36%)
TREC 2003	0.9562 (16.39%)	0.9574 (12.10%)	0.9600 (10.10%)	0.9664 (8.95%)
TREC 2004	0.9740 (12.82%)	0.9797 (9.24%)	0.9862 (8.25%)	0.9849 (7.27%)
Average	0.9652 (14.64%)	0.9786 (8.73%)	0.9856 (5.04%)	0.9889 (3.44%)

degree. In TREC, only the documents in the pool are assessed and the pool comprises the top 100 documents from all or some of the submitted results. Therefore, a result's performance on mean average precision is affected by the other submitted results, and we refer to this phenomenon as the effect of environment.

We carry out an experiment to investigate this effect. For every year group, we evaluate and rank them as well by mean average precision. Then we randomly select a subset (20%, 40%, 60%, and 80%) of all the systems and assume these are all the results submitted, then we follow the TREC routine to generate a pool, and evaluate these systems by mean average precision and rank these results. We compare the ranking obtained from the subset of all the results and the one obtained from all the results to see if there is any ranking exchange for any two results appearing in both cases. Kendall's tau coefficient is calculated for them. Table 3 shows the experimental result. Each data point in Table 3 is the average of 10 runs.

In Table 3, the ranking correlation coefficient values are close to 1 all the time. this means that the relative rankings of a group of results do not change much when some new results are included. Though it can be regarded as a good news, it is not good enough. Since no ranking position exchanging at all is a norm with other measures such as precision at 10 or 100 document level and average precision over all documents.

On the other hand, considerable difference exists for the performance of the same result when the environment changes. When 20% of all results are considered, the difference is over 10% compared with the environment in which all the results are involved.

4 Conclusions

In this paper we have discussed three information retrieval evaluation measures, which are average precision over all relevant documents (mean average precision), precision at a given document level, and average precision over all documents, under the condition of incomplete relevance judgment.

Though it has been believed that average precision over all relevant documents is a good measure, our investigation shows that it suffers from several drawbacks when only partial relevance judgment is available. First, the correct mean average precision value can never be calculated. Hence complete relevance judgment is required for a correct calculation of average precision over all relevant documents. Second, when a pair of results take part in an information retrieval evaluation event such as TREC, their relative ranking positions may reverse if other results involved are different at each time. Though the possibility for such a contradiction is very small, there is no guarantee that it does not happen. Besides, a mean average precision value is difficult to explain, and to calculate average precision values demands great effort. These are two drawbacks of mean average precision even with complete relevance judgment.

Then what about these measures' stability and sensitivity? Our experiment suggests that mean average precision's stability and sensitivity is not superior to the two other measures: average precision over all documents and precision at a given document level, if we use the same (or similar) number of documents for the calculation of these measures. This observation is consistent with previous research [2, 9, 16]. Buckley and Voorhees in [2] find that precision at 1000 documents is more stable than mean average precision, and mean average precision is more stable than precision at 10 documents. The last point is also echoed in [9, 16].

We argue that mean average precision is not a very good measure when relevance judgment is severely incomplete. Although in theory mean average precision has some advantages, its use within TREC evaluation methodology has led to the anomalies discussed above. The difficulties are inevitable in modern IR contexts such as retrieval over the Web. Meanwhile, precision at a given document level and especially average precision over all documents are good measures in such situations. Average precision over all documents has been introduced in this paper and it is more reasonable than precision at a given document level since it distinguishes relevant documents' position. In addition, the similarity between mean average precision and average precision over all documents is more than that between mean average precision and precision at a given document level. Therefore, we consider that average precision over all documents would be a good measure for information retrieval evaluation events such as TREC as well as for researchers to evaluate information retrieval systems and algorithms when the document collection is too big for a complete relevance judgment.

References

1. J. A. Aslam, E. Yilmaz, and V. Pavlu. The maximum entropy method for analysing retrieval measures. In *Proceedings of ACM SIGIR'2005*, pages 27–34, Salvador, Brazil, August 2005.
2. C. Buckley and E. M. Voorhees. Evaluating evaluation measure stability. In *Proceedings of ACM SIGIR'2000*, pages 33–40, Athens, Greece, July 2000.
3. C. Buckley and E. M. Voorhees. Retrieval evaluation with incomplete information. In *Proceedings of ACM SIGIR'2004*, pages 25–32, Sheffield, United Kingdom, July 2004.

4. R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete mathematics*. Addison-wesley publishing company, 1989.
5. K. Jävelin and J. Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems*, 20(4):442–446, October 2002.
6. Y. Kagolovsky and J. R. Moehr. Current status of the evaluation of information retrieval. *Journal of Medical Systems*, 27(5):409–424, October 2003.
7. J. Kekäläinen. Binary and graded relevance in IR evaluations – comparison of the effects on ranking of IR systems. *Information Processing & Management*, 41(5):1019–1033, September 2005.
8. S. E. Robertson and M. M. Hancock-Beaulieu. On the evaluation of IR systems. *Information Processing & Management*, 28(4):457–466, July-August 1992.
9. M. Sanderson and J. Zobel. Information retrieval system evaluation: Effort, sensitivity, and reliability. In *Proceedings of ACM SIGIR'2005*, pages 162–169, Salvador, Brazil, August 2005.
10. W. M. Shaw, R. Burgin, and P. Howell. Performance standards and evaluations in IR test collections: Cluster-based retrieval models. *Information Processing & Management*, 33(1):1–14, January 1997.
11. TREC. http://trec.nist.gov/.
12. C. J. van Rijsbergen. *Information Retrieval*. Butterworths, 1979.
13. V. G. Voiskunskii. Evaluation of search results: A new approach. *Journal of the American Society for Information Science*, 48(2):133–142, February 1997.
14. E. M. Voorhees. Variations in relevance judgments and the measurement of retrieval effectiveness. In *Proceedings of ACM SIGIR'1998*, pages 315–323, Melbourne, Australia, August 1998.
15. E. M. Voorhees. Variations in relevance judgments and the measurement of retrieval effectiveness. *Information Processing & Management*, 36(5):697–716, September 2000.
16. E. M. Voorhees and C. Buckley. The effect of topic set size on retrieval experiment error. In *Proceedings of ACM SIGIR'2002*, pages 316–323, Tampere, Finland, August 2002.
17. S. Wu and S. McClean. Modelling rank-probability of relevance relationship in resultant document list for data fusion, submitted for publication.
18. J. Zobel. How reliable are the results of large-scale information retrieval experiments. In *Proceedings of ACM SIGIR'1998*, pages 307–314, Melbourne, Australia, August 1998.

Sources of Incompleteness in Grid Publishing

Alasdair J.G. Gray[1], Werner Nutt[2], and M. Howard Williams[1]

[1] School of Mathematical and Computer Sciences,
Heriot-Watt University, Edinburgh, EH14 4AS, UK
[2] Faculty of Computer Science, Free University of Bozen,
Dominikanerplatz 3, I-39100 Bozen, Italy

Abstract. There is a wide variety of data, both static and streaming, being published on and about computing Grids. However one aspect of Grid data that has received relatively little attention thus far is that of incompleteness. With potentially many data sources available to choose from, each with its own limitations in terms of coverage of data sets and of overall reliability, there are many opportunities for inaccuracy due to incompleteness in the data. In this short paper different types of incompleteness are identified in the context of R-GMA as part of an ongoing research project aimed at finding solutions to some of these.

1 Introduction

Computational Grids have become an important tool for eScience since they allow access to large amounts of data and the ability to process that data. The data is typically the measurements collected whilst performing some experiment, with a mix of static and streaming data, with some data sources merely providing different views of parts of an overall system. This is true in the case of astronomical data, where a number of different repositories provide overlapping views of heavenly data, or for bio-informatic data sets which cover a wide range of overlapping views on life forms. When publishing data, the source will try to describe its content, but it is difficult and cumbersome to describe accurately the data that is actually available in the repository, e.g. the SuperCOSMOS Science Archive (SSA) astronomical repository [1] claims to contain measurements for the entire southern hemisphere sky in a number of wavebands, although there are patches of the sky which either have not been measured or cannot be observed as they are dominated by a stellar object. This leads to *incompleteness* in the data source with respect to its description.

R-GMA [2, 3] is a monitoring and information service that publishes static and streaming data about the status of the Grid. At its heart, R-GMA is an information integration system for streaming data. This has posed interesting new problems in describing the information published [3]. As well as incompleteness with respect to its description, there are other sources of incompleteness found in R-GMA.

R-GMA is not the only system looking at publishing streaming data on a Grid. Streaming astronomical data is being published in the StreamGlobe project [4] and publishing static and streaming environmental data is the focus of the Linked Environments for Atmospheric Discovery (LEAD) project [5]. In all of these projects there are sources of incompleteness that have not yet been addressed.

D. Bell and J. Hong (Eds.): BNCOD 2006, LNCS 4042, pp. 94 – 101, 2006.

While the topic of incompleteness has previously been studied for distributed databases [6], these are only concerned with static data sources. This paper identifies different types of incompleteness found when publishing static and streaming data on a Grid in which R-GMA is used as an example of a typical Grid publishing system. Section 2 describes how streaming data, such as monitoring data, may be published on a Grid. Section 3 discusses how incompleteness arises and how it can be dealt with in the Grid environment. Related work is presented in Section 4 and our conclusions in Section 5.

2 Publishing Monitoring Data

R-GMA is a Grid Information and Monitoring system that allows users to locate monitoring data of interest without knowledge of where it is published [2, 3]. R-GMA is a local-as-view information integration system. The architecture is shown in Fig. 1 and consists of Primary Producers (which publish monitoring data about the Grid according to some view description), Consumers (which retrieve specific monitoring data by posing a query), and a Registry (which matches Consumer requests with Producer descriptions). For scalability and robustness, the system also has Secondary Producers that pose queries to the Primary Producers and publish the combined results.

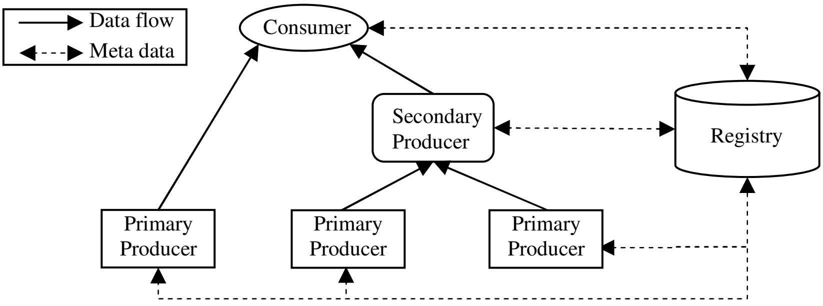


Fig. 1. The architecture of R-GMA

Primary Producers publish their data as a stream of tuples conforming to a selection view description over an agreed global schema. The Primary Producer may additionally maintain a history buffer of the stream, i.e. all the tuples that have been published during some allotted period of time, and/or a latest-state buffer which contains the most recent tuple seen for a given value of the key.

To access the data, Consumers may pose three different types of query over the agreed global schema. The first, a *continuous query* would return every new tuple that satisfies the condition of the query. The second, a *history query* would return all the tuples that have previously been published that satisfy the condition of the query and which fall within the stated time period. The third, a *latest-state query* would return the most recent tuples for the key values that satisfy the query. These last two are referred to as *one-time queries* as they are answered once, at the point in time that they are posed. The different modes of query may be combined, e.g. a history query can be combined with a continuous query to retrieve all the tuples in some stream.

Whenever a Consumer poses a query in R-GMA, it is the role of the Registry to find the relevant producers. The Consumer can then construct a query plan to retrieve the answer to their query. At present, for a continuous query the Consumer may only use Primary Producers and will contact all the Primary Producers which have a view that is relevant. For a one-time query, the Registry will return both Primary and Secondary Producers which are relevant. At present, certain complex queries such as aggregate queries involving streams from several Primary Producers, can only be answered if there is a Secondary Producer that publishes the entire table. When there is no Secondary Producer present, certain classes of queries may be answered by querying the Primary Producers.

3 Working with Incomplete Data on a Grid

This section considers the types of incompleteness that may be encountered when publishing and querying static and streaming data on a Grid.

3.1 Types of Incompleteness

Within the data published by R-GMA, four areas where incompleteness is introduced can be identified: (i) incompleteness at the data source level, (ii) incompleteness arising from the data integration process, (iii) incompleteness due to the data sources being distributed, and (iv) incompleteness with respect to a query. A more detailed description of each of these sources of incompleteness is provided below with examples drawn from the development of R-GMA. Throughout, the examples will use a simplified version of the `GlueCE` relation from the global schema of R-GMA. The `GlueCE` relation is used to monitor the number of jobs running on the various Computing Elements (CEs) on the Grid. The schema of the relation is

```
GlueCE (machineID, siteID, totalCPUs, freeCPUs,
        runningJobs, SE, [timestamp]),
```
(1)

where `machineID` is an identifier for the CE at a site, `siteID` is a unique identifier for the site where the CE is located, `totalCPUs` stores the number of CPUs that the CE has while `freeCPUs` monitors the number of CPUs that are available, `runningJobs` is the current number of jobs running on the CE, the attribute `SE` is used to identify a storage element that is linked with the CE[1], and `timestamp` records the time at which a reading was taken.

Incompleteness at the Data Source Level. A classic example of this type of incompleteness is the use of null values. Null values continue to be used in Grid applications when data is being published against a schema that no longer correctly models it. In the `GlueCE` example, a CE would put a null value in the `SE` attribute as

[1] The `SE` attribute is made up for the purposes of the following examples. In R-GMA there is a separate relation for mapping CEs to SEs that allows for a many to many mapping.

there is now a relation in the global schema that allows multiple storage elements (SEs) to be linked with multiple CEs. The `SE` attribute is kept in `GlueCE` to allow for the adoption of the new table in the schema.

Data source incompleteness also arises when the source promises to contain more data than it actually does. For example, consider the case of there being several Primary Producers for `GlueCE` where each maintains a short history of at most one hour, and the creation of a new Secondary Producer which will maintain a history for the last week. The Secondary Producer can quickly retrieve the tuples covering up to the last hour by posing a history query over the Primary Producers, but it will be unable to retrieve the rest of the tuples for the desired history period as the Primary Producers have discarded the tuples. In this case it will take one week for the Secondary Producer to be complete with respect to its description.

In R-GMA this specific example has been dealt with by the Secondary Producer gradually increasing its declared history period until it reaches the desired length. However, this type of incompleteness is common in the area of astronomical data repositories where meta-data is published expressing what areas of sky are covered and in which wavebands. With these repositories it is often the case that it does not contain all the data from the area claimed; either because no measurements have been made for some small sections, or because some areas cannot be measured due to interference from stellar objects.

Incompleteness from the Data Integration Process. In data integration systems the data in the sources is related to some global schema. It is often not possible or is very cumbersome to accurately map the content of the data sources to the global schema which results in incompleteness. In R-GMA this is due to the limited language supported. For example, a Primary Producer for `GlueCE` may only publish values for `freeCPUs` that are between 0 and `totalCPUs`. However, the view that the Primary Producer registers cannot restrict `freeCPUs` to these values so it must claim to cover a larger set of values than it will ever publish. Consider that there is a CE with a total of 5 CPUs and a Consumer query asking for at least 6 free CPUs. The Primary Producer for this CE would be returned as being relevant for the query since it is unable to restrict its view on the `freeCPUs` attribute to the range 0 to 5.

R-GMA currently copes with this type of incompleteness by treating all Primary Producers as incomplete, but non-overlapping data sources, i.e. all tuples with a specific key value may only be published from a single Primary Producer. This naturally fits the R-GMA system where Primary Producers publish data about the resources of the Grid and the result of the incompleteness on the query is contacting additional data sources that cannot answer the query. However there are other situations in which data streams could be integrated together where the sources publish the same data but with different levels of granularity or accuracy, e.g. stock tickers, and there would be a cost implication in contacting more sources.

Another source of incompleteness due to the integration process arises from the construction of query plans. Consider the history aggregate query trying to find the average number of jobs on each CE over the last 5 minutes. If there are several Primary Producers for `GlueCE` (which is likely in a Grid where each CE would have a Primary Producer publishing this information), then currently R-GMA can only answer this query if there exists a Secondary Producer which publishes the entire table and maintains a history buffer. However, since each Primary Producer is disjoint in R-GMA the correct answer can be achieved by taking the average at each Primary Producer and taking the union of the results, providing that they each maintain an adequate history buffer. The current R-GMA Registry would not be able to construct this plan and the Consumer would be unable to retrieve its answer if the Secondary Producer does not exist.

Incompleteness Due to Distributed Data Sources. A Grid is a complex distributed system with resources being supplied by multiple autonomous organisations, and with users forming inter-organisational collaborations. The distribution of the resources on the Grid can itself lead to incompleteness in the data. For example, incompleteness can arise from network failures. To try and overcome network failures, some sort of soft-state registration is often adopted. In R-GMA, each Primary and Secondary Producer maintains its entry in the Registry by sending "heartbeat" messages. However, this mechanism can fail. For example, when returning the set of relevant producers to a Consumer, it is possible that a particular producer is currently not contactable. (The network has failed since its last heartbeat message.) In this case, the system knows of a potential answer to the query but is unable to return the tuples to the Consumer.

Another important aspect of Grids is the security that needs to be imposed. The resources in a Grid are shared between several different groups of scientists, where each group has its own access rights. It is the responsibility of the resources on the Grid to uphold these access rights. Thus, when a Primary Producer claims to publish some view, a particular group of scientists may only be able to access a portion of this view due to their access restrictions. A scientist in this group may pose a query which could be answered completely from the data available on the Grid, but due to the research group's access rights the scientist will only get an answer from a subset of the data.

A further source of incompleteness due to distribution arises when one data set is derived from another. For example, in R-GMA a Secondary Producer derives its data set from the available Primary Producers. The communication between the Secondary Producer and the Primary Producers can lead to a loss of tuples which results in incompleteness in the derived source. The loss of tuples in the communication is possible because tuples are streamed from the Primary Producers to the Secondary Producer and if the Secondary Producer cannot receive tuples at the rate that they are published they will be dropped and hence lost. This is a common problem arising in stream systems when the stream arrival rate is faster than the rate the system can process the data.

The fact that some data sources on a Grid are derived from others results in inconsistencies due to propagation delays. For example, consider that there is a Primary Producer and a Secondary Producer for the `GlueCE` relation, both of which maintain a latest-state buffer and publish for the same view so that they should contain identical data[2]. Also consider that there is a slow network connection between the Primary Producer and the Secondary Producer such that a Consumer can pose a latest-state query in the time taken to propagate a newly published tuple from

[2] This is a desirable situation as it can allow the system to balance the load of answering a query across several producers.

the Primary Producer to the Secondary Producer. In this case, if the Consumer were to query the Primary Producer it would receive the newly published tuple but if instead the Consumer had queried the Secondary Producer it would receive some other tuple previously published. The Secondary Producer is no longer complete.

Incompleteness with Respect to a Query. The data available in the repositories on a Grid may not contain sufficient data to answer a query. For example, consider a Secondary Producer that maintains a history of 24 hours and a query that is interested in data from the last 48 hours. In this situation the query cannot be answered completely as the available data does not cover the entire period of the query, even though the Secondary Producer is complete with respect to its description. The Secondary Producer may still be used to return some answer to the query.

3.2 Answering Queries in the Presence of Incompleteness

It is not always immediately apparent how to answer a query in the presence of incompleteness as there can be an infinite number of possible complete extensions. In an ideal world, where data sources are complete, available all the time, where data can be described accurately, and query plans can be constructed without limitations, there is only one answer set to a query, the complete answer as there is only one possible information state. When there is a source of incompleteness this answer set cannot be derived accurately from the available data. In order to be able to return some answer to a query, and to be able to interpret that answer, different answer sets are possible based on the possible complete information states. The choice of which answer set should be returned often depends on the requirements of the user. The following discusses the semantics of these answer sets but does not discuss how these answer sets may be derived from the available data in practice.

First there is the set of *certain answers*. This set contains the tuples that are an answer to the query in all complete information states [7]. The certain answers are a subset of the answer in a complete information state and are desirable when a query answer must be entirely accurate. The resulting set can be quite small though.

The other possibility is the set of *possible answers*. This set contains the tuples that are an answer to the query in some information state. The possible answers are a superset of the answer in a complete information state and are desirable in exploratory settings where some incorrect answers can be tolerated.

To understand the differences between these two answer sets consider an astronomical query that is interested in objects that appear in the light waveband that do not have an infrared reading. Consider that we have two data sources which are complete with respect to their descriptions, one for the light waveband and one for the infrared, which partially overlap in the area of sky covered. The set of certain answers would contain tuples for the objects that are in the area of sky covered by both data sources for which there is a reading in the light band but not the infrared band. On the other hand, the set of possible answers would additionally contain those tuples in the light band for which there is no data available in the infrared band.

An answer set could contain tuples along with a query which more accurately describes the tuples returned. This more precise query is an *intensional answer*. Consider again the example of the history query with a period of 48 hours but the data source available only maintaining a history of 24 hours. In this case the Consumer could be returned the tuples covering the last 24 hours together with the more precise query covering the 24 hour period as an intensional answer. If the original query is a monotonic query then the tuples returned by the Secondary Producer with a 24 hour history forms the set of all certain answers as all complete information states are a superset of this source, otherwise the tuples are possible answers.

When tuples are allowed to include null values, then there is the possibility of returning *partial answers* [8]. These are tuples that contain only part of the information requested.

3.3 Generating Query Answers

This section considers whether the type of answer returned can be identified when there is a source of incompleteness. Again, R-GMA is used as an example.

Since R-GMA follows a local-as-view approach to data integration, an intensional answer could be returned. When a query is posed over the global schema, it is translated into a plan over the available data sources. The descriptions of the data sources in the plan can then be used to translate the plan back into a query over the global schema that describes the tuples. This would be particularly useful when there is incompleteness with respect to a query.

In R-GMA it is relatively straightforward to distinguish between when a certain answer or a possible answer is returned. This is because the views are not allowed to include projections or joins. Thus, whenever the query is monotonic the result set will contain certain answers; otherwise it will contain possible answers. More specialised techniques would be required if the views were allowed to contain semi-joins or projections.

In order to be able to inform the user of the type of answer returned, there would need to be some way of attaching meta-data to a result set. In R-GMA this is possible by attaching a warning to the result set and allows meaningful answers to be returned.

4 Related Work

The topic of incompleteness has previously been addressed in the context of incomplete relational databases [7] and in information integration systems [8] resulting in the concepts of certain, possible, and partial answers.

There has recently been a lot of attention to the publication of data on a Grid. The OGSA-DAI system [9] allows for the publication and querying of databases. The StreamGlobe system [4] allows for the publication of sensor data but has no support for static data sources. The Calder system [10] aims to provide an interface between databases published by OGSA-DAI and a data stream query processing engine. So far none of these systems have addressed the issue of incompleteness, although the problem of data sources becoming unavailable has already been highlighted in the Calder system as a source of incompleteness.

5 Conclusions

This paper has shown that incompleteness is an important characteristic of some Grid applications. To date, there has been little or no work looking at how to handle the incompleteness found in streaming applications on a Grid.

In this paper, several types of incompleteness have been identified that arise whilst publishing stream data on a Grid. Some of the types of incompleteness also arise in other Grid applications, for example astronomical data sets.

We have begun to consider what answers could be returned to queries in the presence of the types of incompleteness identified. Already now, more meaning could be given to tuples returned by R-GMA by flagging them as certain or possible answers and by adding intensional answers. We are currently investigating generating intensional answers when there is incompleteness with respect to a query where the histories also contain periods of missing information. One of the major challenges in this area lies in the development of a coherent framework that returns meaningful answers to users in the presence of incomplete data.

References

1. SuperCOSMOS Science Archive (SSA). http://surveys.roe.ac.uk/ssa/
2. A.W. Cooke, et al. The relational grid monitoring architecture: Mediating information about the grid. *Journal of Grid Computing*, 2(4):323-339, Dec 2004.
3. A. Cooke, A.J.G. Gray, and W. Nutt. Stream integration techniques for grid monitoring. *Journal on Data Semantics*, 2:136-175, 2005.
4. B. Stegmaier, R. Kuntschke, and A. Kemper. StreamGlobe: Adaptive query processing and optimization in streaming P2P environments. *In proc DMSN*, pages 88-97, Aug 2004.
5. Linked Environments for Atmospheric Discovery. http://lead.ou.edu/
6. M. Lenzerini. Data integration: A theoretical perspective. *In proc PODS*, Madison (WI, USA), pages 233-246, June 2002.
7. W. Lipski. On semantic issues connected with incomplete information databases. *In ToDS*, 4(3):262-296, Sept 1979.
8. G. Grahne, and V. Kiricenko. Partial answers in information integration systems. *In proc WIDM*, New Orleans (LA, USA), pages 98-101, Nov 2003.
9. M. Antonioletti, et al. The design and implementation of Grid database services in OGSA-DAI. *Concurrency – Practice and Experience*, 17(2-4):357-376, Feb 2005.
10. N. Vijayakumar, Y. Liu, and B. Plale. Calder: Enabling Grid access to data streams. *In proc HPDC*, Raleigh (NC, USA), pages 283-284, July 2005.

Privacy Preservation and Protection by Extending Generalized Partial Indices

Guoqiang Zhan, Zude Li, Xiaojun Ye, and Jianmin Wang

School of Software, Tsinghua University, Beijing, 100084, China
{zhan-gq03, li-zd04}@mails.tsinghua.edu.cn,
{yexj, jimwang}@tsinghua.edu.cn

Abstract. Privacy[1] violation has attracted more and more attention from the public, and privacy preservation has become a hot topic in academic communities, industries and societies. Recent research has been focused on purpose-based techniques and models with little consideration on balancing privacy enhancement and performance. We propose an efficient Privacy Aware Partial Index (PAPI) mechanism based on both the concept of purposes and the theory of partial indices. In the PAPI mechanism, all purposes are independent from each other and organized in a flatten purpose tree($\mathcal{FPT}$). Thus, security administrators can update the flatten purpose tree by adding or deleting purposes. Intended purposes are maintained in PAPI directly. Furthermore, based on the PAPI mechanism, we extend the existing query optimizer and executor to enforce the privacy policies. Finally, the experimental results demonstrate the feasibility and efficiency of the PAPI mechanism.

1 Introduction

The privacy issue has currently become a critical one. Many privacy-aware access control models [1, 11, 13] and specifications [10, 5, 18] have been proposed. Especially the most recent Purpose-Base Access Control model (PBAC) [4, 2] and Micro views [3] have been developed as feasible models and experiments have demonstrated their efficiency. The core techniques, which are used in current models, include *query modification* and *privacy labelling relational* (PLR) data models derived from MLR [14]. However, the two approaches lead to lower performance essentially: PLR increases disk IO and requires extra computing resources for relevant labels; while query modification techniques rewrite a user's queries by appending extra predicates or nested queries, which increases optimizing time, and probably leads to an inefficient executing plan.

An ideal solution to the privacy preservation problem would flexibly protect donor sensitive information without privacy violation, and would incur minimal

[1] The work is supported by both National Basic Research Program of China, Project No.2002CB312006 and National Natural Science Foundation of China, Project No.60473077.

D. Bell and J. Hong (Eds.): BNCOD 2006, LNCS 4042, pp. 102–114, 2006.

privacy enforcing overhead when processing queries. Motivated by this requirement, we propose a new technique, which avoids using both the query modification and PLR data models, to support privacy access control based on the concept of purposes and the theory of partial indices. In our mechanism, we develop a notion of Privacy Aware Partial Index (PAPI), by which privacy policies (i.e. intended purposes in PBAC) are stored and enforced efficiently. We also extend *Purpose Trees* in PBAC into flatten purpose trees with addressing restrictions on *Purpose Trees* in PBAC. Our experimental results verify the feasibility and efficiency of our model.

The rest of our paper is organized as follows. In Section 2, we summarize the recent achievements in the privacy enhancing techniques pertain to purposes. In Section 3, by extending the general partial index, we develop two notions which underlie our research work. In next section, we illustrate how to maintain PAPIs, and how to organize and manage intended purposes based on PAPI. In Section 5, based on PAPI, we extend the traditional query processing engine to provide privacy enhancement. In Section 6, we introduce how to implement our PAPI mechanism, and describe some experiments that demonstrate the efficiency and scalability of our approach. Finally, we conclude the paper and outline future work.

2 Related Work

Privacy protection is related to many different areas in secure data management. As described in Common Criteria (CC) [12], to implement a solid privacy preserving data management system, we have to support at least the following three security requirements: Access Control, Unobservability and linkability. According to the above privacy preserving requirements, privacy enhancing techniques can be classified into three categories: Privacy Aware Access Control (PAAC) [4, 2, 11], Private Information Retrieval (PIR) [8] and Privacy Information Inference Control (RIIC) [7](*k-anonymity* technique [17] belongs to this category).

Our work focuses on the PAAC technique for access control. The most recent popular techniques in this field include privacy policy specification [10, 5, 18] , purpose specification and management [1, 4, 18] and privacy polices enforcement models [4, 3]. In addition, we have used the following three core concepts in our work: partial indices [16], query optimizer [15] and executing engines [9].

The Platform for Privacy Preferences(P3P) [18] by W3C enables users to gain more control over the use of their personal information on web sites they visit. And also, APPEL [5] by W3C and EPAL [10] by IBM provide a formal way to define the privacy policies or usage preferences, but without detailed specifications to enforce the policies in an information system or product, such as DBMS.

Based on HDB [1], LeFevre etc. [11] presented a database architecture for enforcing limited disclosure expressed by privacy polices based on their proposed ten principles. They also suggested an implementation based on query modification techniques. By extending the concepts of purposes in [18], Ji-Won Byun etc. [4, 2] presented a comprehensive approach to purpose management (called purpose-based access control, PBAC for short), in which all the purposes are organized in a hierarchical way. In the PBAC model [4], they developed three basic concepts: intended purposes, access purposes and purpose compliance. Based on these concepts, they suggested an implementation for PBAC based on the query modification technique and PLR derived from MLR [14]. In [2], extended their previous work in [4] to the XML-oriented information system and Object-oriented system, and proposed a systematical model to determine the access purposes based on RBAC. In [3], Ji-Won Byun etc. go even further on the purpose-based access control (PBAC) with incorporating generalization techniques to enhance the privacy preservation.

As drawn from the above models, some open issues are listed below, which have motivated us to seek more feasible and efficient solutions to enforce privacy policies.

• **Query Modification Techniques.** Query rewriting always changes the original queries by introducing some extra predicates or the nested queries, which increase optimizing time and require extra computing time.

• **Privacy Labelling Relational Data Model (PLR).** PLR is adopted in many models, which changes the standard relational data model by adding the privacy labelling attributes [4] or choice columns [11]. This strategy will actually increase the I/O cost and involve considerable extra computation against the labels. Especially if there are many privacy-sensitive attributes in a relation, the performance will be degraded drastically.

• **Hierarchy Relationship Among Purposes.** In [4], it is assumed that all the purposes are predetermined. However, this assumption does not always hold, especially in small organizations (which are project-oriented). The organizational structures will change frequently in these project-oriented organizations, leading the purpose tree being reshaped repeatedly because of the hierarchical relationships among purposes. As a result, all intended purposes in the relation have to be re-evaluated too. In addition, a non-leaf purpose is combinational purpose, which consists of multiple nested purposes. If a donator allows her/his information accessed for a purpose, then the information can also be used for its nested purposes. So donators should have knowledge of the purpose tree or the organizational structure. Obviously, it is very inconvenient for users to protecting their sensitive data.

To address the above challenges, we have developed a privacy-aware query processing mechanism based on partial indices to provide an efficient solution to the above problems. In our model, we first adopt from [4] three concepts, namely *purpose*,*intended purpose* and *access purpose*. We then avoid using the query modification technique and discard the PLR data model. Finally, by transforming a purpose tree $\mathcal{PT}$ [4] to a flatten purpose tree ($\mathcal{FPT}$) as shown in Fig.1, all purposes in $\mathcal{FPT}$ are peers without hierarchical relationships. Thus DBA can easily change purposes and reshape $\mathcal{FPT}$ without re-evaluating all intended purposes.

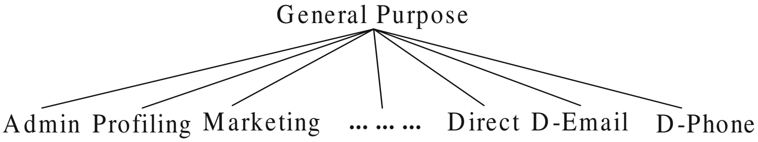


Fig. 1. Flatten Purpose Tree

Based on the concept of $\mathcal{FPT}$, relevant concepts in [4], including purpose & purpose tree, intended purpose, and access purpose, are redefined.

Definition 1. *(Purpose and Flatten Purpose Tree) A purpose describes the reasons for what data is collected or used. Purpose are organized in a flatten hierarchical structure, referred to Flatten Purpose Tree ($\mathcal{FPT}$). Actually, all the purposes are peer except the root purpose which is a virtual purpose.*

Definition 2. *(Intended Purpose) Let $\mathcal{FPT}$ be a flatten purpose tree and $\mathcal{P}$ be the set of all purposes in $\mathcal{FPT}$. An intended purpose, denoted by* IP*, is used to describe usage granted by the donators, and it is $\{AIP - PIP\}$, where $AIP \subseteq \mathbf{P}$ is a set of allowed intended purpose, and $PIP \subseteq \mathcal{P}$ is a set of prohibited purpose.*

Example 1. Suppose $AIP = \{Admin, Direct\}, PIP = \{D\_Email\}$ is defined over $\mathcal{FPT}$ given in Fig.1. Then, intended purposes (IP) is evaluated as:
$IP = AIP - PIP = \{Admin, Direct\}$.

Definition 3. *(*Access Purpose*) Let $\mathcal{FPT}$ be a flatten purpose tree. An access purpose(**AP**), is the purpose for accessing data, and it is included in $\mathcal{FPT}$.*

Definition 4. *(Access Purpose Compliance) Let $\mathcal{FPT}$ be a flatten purpose tree, Let* IP *and* AP *be intended purpose and access purpose respectively. AP is said to be compliant with IP, only if the following condition is satisfied: $AP \in IP$.*

3 Purpose Aware Partial Indices

The concept and performance of partial indices are surveyed in [16], which illustrated that partial indices would lead to a great improvement of performance in many scenarios, especially in distributed system. In traditional indexing schemes, some of the columns are not indexed. A partial indexing scheme takes this one step further, and some of the tuples are not indexing into the indexes.

In this paper, we propose a new approach to take privacy preservation techniques further, called privacy-aware query processing based on the privacy aware partial index (PAPI) which is an extension to the generalized partial index [16] and is mainly used to store intended purpose and enforce privacy policies. For complicated features in SQL (like, conjunctive or disjunctive *Selection*), we propose two kinds of PAPI to enforce the privacy policies efficiently: attribute-oriented PAPI (APAPI) and tuple-oriented PAPI (TPAPI). And TPAPI is mainly used to process non-conjunctive queries. Based on PAPI, our model can support both tuple-level and element-level privacy access control, and partial results for queries are supported,too.

Definition 5. (Attribute oriented Privacy Aware Partial Index, APAPI) *Let* $\mathbb{R} = \{A_1, A_2, \ldots, A_n\}$ *be a relation,* $\mathcal{P}$ *be the set of all purposes in* $\mathcal{FPT}$*, and* $P_k (\in \mathcal{P})$ *be a purpose. An* **APAPI(**A_i, P_k**)** *is a partial index defined on* A_i*, and index the tuple* $t_j(a_{j1}, a_{j2}, ..., a_{ji}, ..., a_{jn})$ $(t_j \in \mathbb{R})$ *if and only if* $P_k \in IP_{ji}$ *is held* (IP_{ji} is the intended purpose on a_{ji} in t_j).
An APAPI(A_i, P_k*)is defined as* $< key, CI, TID >$*, where:*

- ***key*** *is the element* a_{ji} *for* A_i *which APAPI(*A_i, P_k*) is defined on.*
- ***Compliance Indicator****(CI) is a bit string* $\{b_1 b_2 \ldots b_n\}$*,the size of CI is determined by the number of attributes in* $\mathbb{R}$*. Given a tuple* t_j*,* a_{jh} *is an element in* t_j *with the intended purpose* IP_{jh}*, then* $b_h (j = 1, \ldots, n)$ *is assigned according to the following rules:*
  - $b_h = 1$*, if and only if* $P_k \in IP_{jh}$ *is true, or there is no privacy protection requirement for* a_{jh}*;*
  - $b_h = 0$*, otherwise.*

  CI is mainly used to support the partial result of a query, which is different from PBAC which filters out the whole tuples if any of its elements violate the privacy policies. In CI, there are some reserved bits used for new appending attributes in future.
- ***TID*** *is the physical address locating a tuple uniquely and directly.*

Table 1. Personal Information Table : PI_Table

TID	Name	N_L	Gender	G_L	Age	A_L
1	Jone	Marketing	Male	Marketing	18	Analysis
2	Smith	Profiling	Male	Analysis	34	Marketing
3	Alice	Marketing	Female	Analysis	18	Marketing
4	Vincent	Third Party	Male	Marketing	29	Third Party

Example 2. Given (PI_Table) which records the personal information and will be as an example throughout this paper. An APAPI, denoted as **Name_APAPI**, is created on the attribute *Name* for the purpose *Marketing*, and the last five bits in its CI are the reserved bits. The result is shown in the Table 2.

Table 2. APAPI on PI_Table

Name	N_CI	TID
Jone	110 00000	1
Alice	001 00000	2

Table 3. TPAPI on PI_Table

N_CI	TID
110 00000	1
101 00000	3
010 00000	4

As we know, in traditional DBMS, it is difficult for an optimizer to choose an index access method to access the relation on which has disjunctive selection predicates, except few combinational indices. And in our model, the privacy policies are enforced in privacy aware query optimizer firstly by choosing optimal and suitable PAPIs to access the relevant relation. So, when there exists an disjunctive conditions on the base relation, the model probably fail to enforce the privacy policies because the optimizer can not choose APAPIs to access this relation under this scenarios. Fortunately, we introduce TPAPI, whenever we can choose the TPAPI as the access method to a relation. Because a TPAPI is independent from any attributes.

Definition 6. (Tuple oriented Privacy Aware Partial Index, TPAPI) *Let* $\mathbb{R} = \{A_1, A_2, \ldots, A_n\}$ *be a relation,* $\mathcal{P}$ *be a set of purposes in* $\mathcal{FPT}$*, and* $P_k (\in \mathcal{P})$ *be a purpose. A* **TPAPI(**P_k**)** *is a partial index to index the tuple* $t_j(a_{j1}, a_{j2}, ..., a_{ji}, ..., a_{jn})$ $(t_j \in \mathbb{R})$ *if and only if there exists any element* a_{ji} *satisfying* $P_k \in IP_{ji}(IP_{ji}$ is the intended purpose on a_{ji} in $t_j)$*,at least. A TPAPI(*P_k*)is defined as* $< CI, TID >$*, where:*

- ***Compliance Indicator****(CI) is same to the counterpart defined in APAPI*
- ***TID*** *is the physical address locating a tuple uniquely and directly.*

Example 3. The TPAPI, denoted as ***Name*_TPAPI**, is created on PI_Table for the purpose *Marketing*, and the result is shown in the Table 3.

4 PAPI Maintenance and Intended Purpose Management

This section focuses on how to collect and store user data and designated privacy policies (i.e. purposes) by donators. When users request some services, necessary data is collected by terminals, like Web Browsers, according to P3P [18]. Users provide the necessary personal information, and specify the usage type (intended purpose, short for **IP**), thus the tuple $< data, IP >$ is formed and transported into the back-end privacy aware DBMS. Then, *data* is inserted into the relation, and update the PAPIs according to *IP*.

4.1 Establishing Basic PAPIs

The objectives of PAPI include: 1) store the privacy policies **completely**; 2) improve the processing performance of privacy-oriented query (in which users designate access purposes) based on the feature of indices. And the former is a fundamental and indispensable objective. We should ensure the completeness for the privacy policies in PAPIs. It is equal to the question: how many PAPIs should be created to store the privacy policies completely, or at least?

- **PAPI Completeness.** Given $\mathcal{P}$ is the set of purposes in $\mathcal{FPT}, \mathcal{P} = \{P_1, P_2, \ldots, P_m\}$, and a relation $\mathbb{R}$ defined as: $\mathbb{R} = \{A_1, A_2, \ldots, A_n\}$.

Theorem 1. *Given* $\mathcal{P}$ *and* $\mathbb{R}$*, and considering the* APAPI *only. For an attributes* A_i*, at least* $\boldsymbol{m}$ APAPIs *have to be defined on it for m purposes respectively to ensure the completeness.*

Proof. All the possible purposes are defined in $\mathcal{FPT}$. Give an element $elem_i$ in the tuple $\mathcal{T}$ for A_i , if its intended purpose (IP_i) satisfies: $IP_i \in \mathcal{P}$, there must exist an entry for $\mathcal{T}$ in the APAPI(A_i, P_k). If there is only (m-1) APAPIs are defined, then there must exists a purpose P_h on which an APAPI (A_k, P_h) is not defined. So if there certainly exists a tuple $\mathcal{T}'$ in which $elem'_i$ for A_i can used for P_h, then $elem'_i$ will lose the privacy policy for P_h because of absenting APAPI(A_k, P_h). So, we need to define m APAPIs on each purpose, at least.

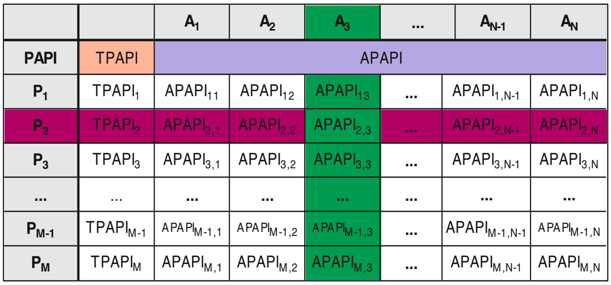
Therefore, according to the Theorem 1, for a relation $\mathbb{R}$ with n attributes, we have to create $n * m$ APAPIs totally to ensure the privacy completeness.

Theorem 2. *Given $\mathcal{P}$ and $\mathbb{R}$, and considering the TPAPI only. For $\mathbb{R}$,* **m** *TPAPIs need to be defined for m purposes respectively to ensure the completeness.*

Proof. According to the definition for TPAPI, and given a TPAPI(P_i), if a tuple $\mathcal{T}$ including any element used for P_i at least, then there must exist an entry for $\mathcal{T}$ in TPAPI(P_i). Obviously, if TPAPI(P_k) is not created, then tuple for P_k cannot be located by TPAPIs. So m TPAPIs have to be maintained for m purposes respectively.

For considering the complexity of SQL syntax and diversity of applications, we maintain APAPI and TPAPI simultaneously to facilitate different kinds of queries and applications.

So, given $\mathcal{P} = \{P_1, \ldots, P_m\}$ in $\mathcal{FPT}$, and a relation $\mathbb{R}$ with n attributes, we have to create n*m APAPIs, and m TPAPIs. These are demonstrated in a matrix (X-coordinate for attributes, Y-coordinate for purposes) in Fig.2.



		A_1	A_2	A_3	...	A_{N-1}	A_N
PAPI	TPAPI	APAPI					
P_1	TPAPI_1	APAPI_{11}	APAPI_{12}	APAPI_{13}	...	$\mathrm{APAPI}_{1,N-1}$	$\mathrm{APAPI}_{1,N}$
P_2	TPAPI_2	$\mathrm{APAPI}_{2,1}$	$\mathrm{APAPI}_{2,2}$	$\mathrm{APAPI}_{2,3}$	...	$\mathrm{APAPI}_{2,N-1}$	$\mathrm{APAPI}_{2,N}$
P_3	TPAPI_3	$\mathrm{APAPI}_{3,1}$	$\mathrm{APAPI}_{3,2}$	$\mathrm{APAPI}_{3,3}$	...	$\mathrm{APAPI}_{3,N-1}$	$\mathrm{APAPI}_{3,N}$
...	...	...	...	...	...	...	...
P_{M-1}	TPAPI_{M-1}	$\mathrm{APAPI}_{M-1,1}$	$\mathrm{APAPI}_{M-1,2}$	$\mathrm{APAPI}_{M-1,3}$	...	$\mathrm{APAPI}_{M-1,N-1}$	$\mathrm{APAPI}_{M-1,N}$
P_M	TPAPI_M	$\mathrm{APAPI}_{M,1}$	$\mathrm{APAPI}_{M,2}$	$\mathrm{APAPI}_{M,3}$	...	$\mathrm{APAPI}_{M,N-1}$	$\mathrm{APAPI}_{M,N}$

Fig. 2. PAPI matrix for all basic PAPIs

4.2 Updating PAPIs

Updating PAPIs occurs in the following three cases: 1) update the relation $\mathbb{R}$, including insertion, update, and delete; 2)update the privacy policies on data by donators; 3) update the flatten purpose tree $\mathcal{FPT}$. We discuss how to adjust the PAPIs to these changes.

Assuming that $\mathbb{R}$ contains n attributes, $\mathcal{P}$ includes m purposes, and each data item can only used for one purpose (actually, one intended purpose may contain several purpose).

- **Updating Relation.** When a relation $\mathbb{R}$ is updated, associated PAPIs are updated simultaneously, as general indices do.
- **Updating Privacy Policies on Data.** As a flexible privacy preserving model, it has to facilitate donators to modify their privacy policies efficiently and conveniently. In our model, when donators change privacy policies on an element, four PAPIs have to be updated,(including two TPAPIs and two APAPIs respectively).
- **Updating $\mathcal{FPT}$.** If security administrator has to remove a purpose, they just need to remove the relevant row in the matrix shown in Fig.2. Taking the purpose P_2 for example, if P_2 is deleted, only the second row in the matrix is deleted without any influence to the rest PAPIs. When a new purpose is appended, only a new row is appended into the matrix.

4.3 Analysis of PAPI

The analysis focuses on PAPI selectivity which is defined as the number of entries in each PAPI and its storage cost. The smaller the selectivity is,the faster it will accelerate query processing. The analysis is based on the following assumptions:

- Probability of an element used for a purpose is equal.
- Average number of intended purposes for each cell is denoted as $\boldsymbol{l}$.
- $\mathbb{R}$ contains $\boldsymbol{n}$ attributes and $\boldsymbol{c}$ tuples, and $\mathcal{P}$ includes $\boldsymbol{m}$ purposes.

Selectivity
According to the above assumptions, given APAPI(A_i, P_i) and a tuple $t_j(a_{j1}, a_{j2}, \ldots, a_{jn})$, the probability of a_{ji} used for P_i is l/m. So the probability of indexing t in APAPI(A_i, P_i) is also l/m, and there are $(l \times c)/m$ entries on average for APAPI(A_i, P_i).

As we know that the probability of a_{ji} used for P_i is l/m, while there are n elements in a tuple t, and if all the attributes are independent from each other, thus the probability of indexing t in TPAPI(P_i) is $(n \times l)/m$ according to the definition of TPAPI. So, there are $(n \times l) \times c/m$ entries for TPAPI(P_i) on average.

Storage Overhead
According to the above analysis, for an APAPI, there are $(l \times c)/m$ (c is cardinality of $\mathbb{R}$) entries. So, for $n \times m$ APAPIs, there are $(n \times m) \times (l \times c)/m$ entries in all. And also there are $(n \times l) \times c/m$ for each TPAPI, thus for m TPAPIs, there are $m \times (n \times l) \times c/m$ entries in all. So, in a privacy-aware DBMS based on PAPI mechanism, the number of all the indices entries in PAPIs (both APAPI and TPAPI) sums to $(n \times m) \times (l \times c)/m + m \times (n \times l) \times c/m = 2(n \times c) \times l$. While, in $\mathbb{R}$, there are $n * c$ data cells, so the whole extra storage overhead is $2 \times l$ times as the original table. So, the storage overhead is attributed to l, while l is due to the specific applications.

From the above analysis, the selectivity for TPAPI on average is n times lower than it for APAPI. So, the retrieval performance of TPAPI may be greatly lower than that of APAPI theoretically.

5 Privacy-Aware Query Processing Engine

This section mainly focuses on how to enforce privacy policies in query processing engine. And we assume that user's access purposes are authenticated. As we known, a general query processing engine consists of two modules: optimizer and executor. So, our efforts focus on extending existing optimizer and executor to enhance privacy policies with incorporating PAPI mechanism.

5.1 Privacy Aware Query Optimizer (PAQO)

As described in [15, 9], the mechanism for a general query optimizer facility can formulated briefly as: 1) choose two optimal access methods for each base relation firstly. One is the cheapest access method returning tuples in an interesting order, another one is the cheapest access method without considering order; 2) choose an overall optimal path (mainly considering the join orders), which is generated by combining these optimal access methods based on dynamic planning algorithm, greedy algorithm, or genetic algorithm etc. Then, the optimal path is passed into the executor in which tuples are processed one by one.

According to principles of the optimizer, in PAQO, the optimal access methods on each relation are restricted within these PAPIs which are created for the access purpose indicated in users' queries. So, when a plan is determined, privacy policies are enforced at tuple level, because accessible tuples are pre-determined by PAPIs. However, we don't know which attributes can be accessed. This problem can be solved by the compliance indicator (**CI**) in PAPIs, **CI** need to be checked in executor when accessing cells in tuples. That's why we extend the existing query executor.

5.2 Privacy Aware Query Executor (PAQE)

Through PAQO, the coarse privacy policies (**PP**) have been enforced. The fine-grained PP enforcement is left for executor to check the compliance indicator (CI) further and return the suitable partially incomplete result.

Without considering the inference violation, the fine-grained PP is enforced according to below rules(called Loose Rules):

- Let $t_j\{a_{j1}, a_{j2}, \ldots, a_{jn}\}$ be a tuple, and $CI_j(b_{j1}b_{j2}\ldots b_{jn})$ is the **CI**.
- If $b_{ji}(CI) = 1(i = 1, \ldots, n)$, output the a_{ji} if necessary;
- If $b_{ji} = 0$, replace a_{ji} by NULL.

Example 4. Display the personal information used for *Marketing.*
Q_2: SELECT * FROM PI_Table WHERE Name='Jone' And Age=18;
The result for Q_2 is displayed as in the **Table 4**.

Table 4. Result under Loose Rules

Name	Gender	Age
Jone	Male	NULL
Jone	NULL	18

Table 5. Result under Strict Rules

Name	Gender	Age
Jone	NULL	18

There is an inference violation from the result of Q_2 in Table 4. The Q_2'owner can easily infer that age for Jone in the first row is also 18 and its intended purposes do not include *Marketing*.

To avoid this inference disclosure, we find out selective attributes (SA) which are used to filter out unqualified tuples in the relation $\mathbb{R}$. With considering the inference control, based on *Loose Rules*, **Strict Rules** for the fine-grained PP enforcement are developed:

- Let $t_j\{a_{j1}, a_{j2}, \ldots, a_{jn}\}$ be a tuple, and $CI_j(b_{j1}b_{j2}\ldots b_{jn})$ is the **CI**.
- If $b_{ji}(CI) = 1(i = 1, \ldots, n)$, output the a_{ji} if necessary;
- If $b_{ji} = 0 \bigwedge A_i \notin SA$, replace a_{ji} by NULL.
- If $b_{ji} = 0 \bigwedge A_i \in SA$, t_j is discarded directly.

According the strict PP enforcing rules, SA for Q_2 is { Name, Age}, and the result for Q_2 is shown in **Table 5**.

To summarize, privacy policies are enforced through two steps: coarse PP enforcement which is fixed in privacy aware query optimizer (PAQO) and fine-grained PP enforcement which is fixed in privacy aware query executor (PAQE).

6 Implementation and Experiments

6.1 Implementation on PostgreSQL

The implementation mainly involves the below aspects: purpose management, PAPI maintenance, and extending query optimizer and executor.

Purposes are stored in a system catalog (called *Pg_Purpose*) which is created as a part of data dictionary (DD) when database is installed initially. DBAs can create new purpose by extended DDL. And each purpose has a unique code.

In reality, purposes, in our implementation, can be considered as special privileges, and it will be granted or revoked as general privileges do. The access purposes are granted to users firstly. If a user intends to access data for a purpose, DBMS checks the access purpose against allowed access purposes in **ACL**. If the access purpose is allowed, then the request is processed; else, rejected directly.

Basic PAPIs are created automatically, when a relation is created, and automatically establish dependency with target relations. The command for PAPIs is extended from the standard command for the index. Maintenance for PAPIs follows the paradigm for general indices. Besides, all PAPIs have to establish the dependency on *Pg_Purpose*, too. When any updates on $\mathcal{FPT}$, cascading actions will take effect on the corresponding PAPIs.

For the privacy aware query optimizer (PAQO), we only need to add a new branch to support the privacy feature without influencing its original function. When users submit privacy-oriented queries, the optimizer chooses optimal path from those PAPIs defined for the access purpose.

According to PAQO, all tuples, which are indexed in PAPIs, can be accessed for privacy-oriented queries. And PAQE enforces **strict rules** on cell level by filtering out the unqualified tuples or suppressing tuple cells.

6.2 Experimental Evaluation

The goal of our experiments is to investigate the feasibility and performance of our mechanism. We mainly focus on comparing the performance against the two traditional techniques, namely query modification techniques, labelling schema (such as element-based labelling PBAC model), varying the number of attributes accessed and privacy policies selectivity. Sequentially, we demonstrate the relationship between the performance of our model and different cardinalities. Finally, we analyze the storage overhead of our model.

Experimental Setup. The main experimental environment is configured as below: CPU is Intel P4C with 1G DDR-400, Redhat Linux AS 4.0 is installed on the machine, and PostgreSQL-7.43 is used as the RDBMS. Our PAPI mechanism is implemented by extending the PostgreSQL-7.43, while the element-base PBAC model is simulated: each attribute is entailed with two extra labels with the type *smallint*, and the privacy policies enforcement function (i.e. Comp_check(AP, AIP, PIP)) is substituted by a simple predicate involving the two corresponding labels. The tested data set is a version of large size tuples schema used in [6] .

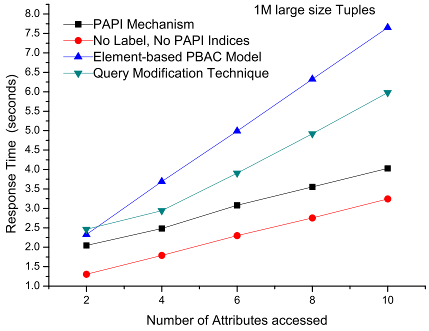


Fig. 3. Mechanisms VS Performance

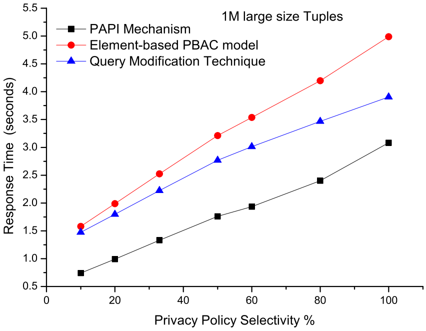


Fig. 4. Selectivity and Performance

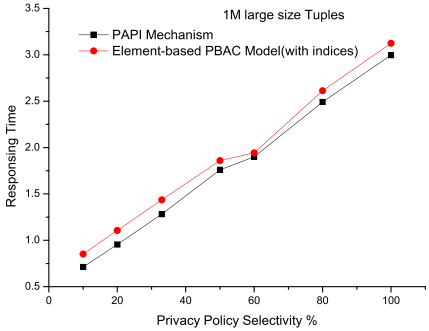


Fig. 5. PAPI and Element-based PBAC

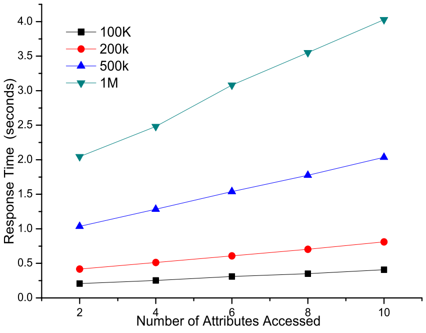


Fig. 6. Cardinality and Performance

Response time is used as the metric to measure the performance. In our experiments, the response time is referred to the retrieval time and a trivial counting time by evaluating the standard aggregate function COUNT(). A sample of queries used in the experiments is:

SELECT COUNT(unique1),COUNT(two),COUNT(unique2),
COUNT(four) FROM mtuples5;
Before we execute the target testing cases, we load the data set into memory as possible as we can, and each testing case is run for ten times.

Experimental Results. In order to compare the performance of different techniques, firstly we assume the selectivity of all data elements (for both actual attributes and purpose labels) to be 100 percent, but vary the number of attributes accessed in each query. The result shown in Fig. 3 demonstrates that PAPI gains the best performance, and has a significant improvement against the query modification technique and element-based PBAC model. Actually PAPI mechanism introduces some extra overhead against the standard relational data model (without any labels and PAPI's indices), because the in PAPI mechanism, it has to access the PAPI's indices to locate every tuples whatever method the tuples are accessed. Fig. 4 shows the performance of different techniques in case of different privacy policy selectivity (i.e. the selectivity of data elements for actual attributes is still 100 percent, but the selectivity of data elements for purpose labels is variable), varying from 10 percent to 100 percent, and all testing queries access the same six attributes. We learn that PAPI mechanism gains the best performance in any selectivity, too.

As we know that element-base PBAC model will achieve a drastic improvement on performance based on the functional index by pre-computing the given function which is used to enforce the privacy policies. Fig.5 shows the comparison between PAPI mechanism and element-base PBAC model(with indices). And from the experimental results, PAPI mechanism also has a little better performance, because in element-base PBAC model, each attribute are entailed with two labels for AIP and PIP respectively, and thus the size of each tuple is enlarged, directly leading to the longer accessing time in both memory and disks. If using the functional index to accelerate PBAC model, it will achieve a significant improvement; unfortunately, it will not support the partially incomplete result which is acceptable in some application environment.

Also, we investigate the scalability of our PAPI mechanism by considering different cardinalities. Fig.6 shows that the query processing cost merely increases in a linear way against the cardinality.

7 Conclusions

In this paper, we summarize the recent achievements on purpose-based privacy enhancing techniques. Motivated by some problems in these techniques, we propose a PAPI mechanism to facilitate the privacy policies enforcement by extending the traditional query processing engine based on two concepts: attribute oriented privacy-aware partial index and tuple oriented privacy-aware partial index. And intended purposes are efficiently maintained by PAPIs. Finally, through experiments, the feasibility and performance of PAPI mechanism are demonstrated. We plan to design a privacy-aware DBMS based on PAPI mechanism with incorporating other techniques, such as suppression, generalization and so on.

References

1. R. Agrawal, J. Keirnan, R. Srikant, and Y. Xu. Hippocratic database. *In Proceedings of the 28th VLDB Conference.*, 2002.
2. J.-W. Byun, E. Berino, and N. Li. Purpose based access control of complex data for privacy protection. *Proceedings of the tenth ACM symposium on Access control models and technologies (SACMAT'05)*, pages 102–110, June 2005.
3. J.-W. Byun and E. Bertino. Vision paper: Micro-views, or on how to protect privacy while enhancing data usability. *To be published in SIGMOD Record.*, 2005.
4. J.-W. Byun, E. Bertino, and N. Li. Purpose based access control for privacy protection in relational dtabase systems. *Technical Report 2004-52,Purdue Univ,2004.*
5. W. W. W. Consortium(W3C). A p3p preference exchange language 1.0 (appel 1.0). available at. *www.zurich.ibm.com/security/enterprise-privacy/epal.*
6. D.Bitton, D.J.DeWitt, and C.Turbyfill. Benchmarking database: system a systematic approach. *In Ninth International Conference on Very Large Data Bases*, pages 8–19, Oct 1983.
7. D.Bitton, D.J.DeWitt, and C.Turbyfill. Benchmarking database system a systematic approach. *In Proceeding of CCS'04.*, pages 25–29, Oct 2004.
8. W. Gasarch. A survey on private information retrieval. *The Bulletin of the EATCS*, 82:72–107, 2004.
9. G. Graef. Query evaluation techniques for large databases. *ACM Computing Surveys*, 25(2):73 – 169, June 1993.
10. IBM. The enterprise privacy authorization language (epal). available at. *www.w3.org/TR/P3P-preferences.*
11. K.LeFevre, R.Agrawal, V.Ercegovac, R.Ramakrishnan, Y.Xu, and D.DeWitt. Limiting disclosure in hippocratic database. *In The 30th International Conference on Very Large Databases*, Aug. 2004.
12. T. C. C. P. S. Organisations. Common criteria for information technology security evaluation, part 2, draft version 3 and version 2.1-2.3, august. 2005.
13. P.Ashley, C. Powers, and M.Schunter. Privacy, access control, and privacy management. *In Third International Symposium on Electronic Commerce*, 2002.
14. R. Sandhu and F. Chen. The multilevel relational(mlr) data model. *ACM Transactions on Information and System Security.*, 1(1):93–132, November 1998.
15. P. Selinger, M.M.Astrahan, D.d.Chamberlin, R.A.Lorie, and T.G.Price. Access path selection in a relational dababase management system. *In Proceedings of the 1979 ACM SIGMOD Conference on the Management of Data.*, May-June 1979.
16. P. Seshadri and A. Swami. Generalized partial indexes. *Proceedings of the Eleventh International Conference on Data Engineering (ICDE)*, pages 420 – 427, Mar. 1995.
17. L. SWEENEY. k-anonymity: A model for protecting privacy. *International Journal on Uncertainty,Fuzziness and Knowledge-based Systems*, 10(5):557–570, 2002.
18. W. W. W. C. (W3C). Platform for privacy preferences (p3p). *Available at www.w3.org/P3P.*

On the Optimal Ordering of Maps, Selections, and Joins Under Factorization

Thomas Neumann[1], Sven Helmer[2], and Guido Moerkotte[3]

[1] Max-Planck Institute of Computer Science, Saarbrücken, Germany
neumann@mpi-inf.mpg.de
[2] University of London, Birkbeck College, London, United Kingdom
sven@dcs.bbk.ac.uk
[3] University of Mannheim, Mannheim, Germany
moerkotte@informatik.uni-mannheim.de

Abstract. We examine the problem of producing the optimal evaluation order for queries containing joins, selections, and maps. Specifically, we look at the case where common subexpressions involving expensive UDF calls can be factored out. First, we show that ignoring factorization during optimization can lead to plans that are far off the best possible plan: the difference in cost between the best plan considering factorization and the best plan not considering factorization can easily reach several orders of magnitude. Then, we introduce optimization strategies that produce optimal left-deep and bushy plans when factorization is taken into account. Experiments (1) confirm that factorization is a critical issue when it comes to generating optimal plans and (2) we show that to consider factorization does not make plan generation significantly more expensive.

1 Introduction

Two things matter in optimizing queries with predicates containing user-defined functions (UDFs): (1) UDFs tend to be much more expensive to evaluate than other functions and (2) the same expensive UDF may be called repeatedly. To deal with the former, the optimizer finds an optimal ordering of selections and joins; to deal with the latter, it identifies common subexpressions in selection predicates, factorizes them (i.e. makes sure that each common subexpression is only evaluated once), and orders their evaluation optimally. Both these problems are already NP-hard in isolation [11, 14], combining them does not make things easier. However, improvements by one or more orders or magnitude are possible if factorization is taken into account during the ordering of joins and selections. Although Yajima first considered ordering expensive selections and joins in 1991 [19], the relevance of factorization in this context has never been considered. Indeed, all existing algorithms for ordering expensive selections and joins yield suboptimal results in the presence of common subexpressions involving expensive predicates. We show that when optimizing queries that contain expensive predicates and also share common subexpressions, it is critical to take both into account. Surprisingly, this can be done without increasing the cost of plan generation.

D. Bell and J. Hong (Eds.): BNCOD 2006, LNCS 4042, pp. 115–126, 2006.

Motivating Example. Let us illustrate the optimization potential with an example from cell biology [3]. In a cell, some proteins are translocated into the endoplasmatic reticulum (ER). Out of these, some are cleaved, that is, their signal sequence is cut off. It is important to know (1) given a protein, does it have an ER signal and (2) is the signal cleaved and, if so, where (cleavage site)? When designing a new prediction method to improve upon existing state of the art predictors[1], it is important to look at cases where the prediction methods agree with or differ from experimental data. Consider the following scenario. The set of sequences under investigation is contained in a relation `Sequence` with a `varchar` attribute `seq` containing the actual amino acid sequence. The results of state of the art predictors for signals and cleavage sites are materialized in relations `SignalPred` and `SitePred`. Two user-defined functions `signalpred` and `sitepred` implement a new pair of prediction methods. Their result is not materialized in a relation since they still change frequently. Actual results of experiments are available in the relation `Experiment`. All relations have an attribute `id` which uniquely identifies a sequence. Figure 1 shows the typical pattern for a query searching for correspondences and deviations between prediction methods and experiments. The operator θ is a simple SQL expression (often using (`not`) `between` on the difference of the arguments of θ) selecting the degree of accordance or deviation the user is interested in.

```
select *
from   Sequence S, Experiment E,
       SignalPred A, SitePred B
where  S.id = E.id
       and S.id = A.id
       and S.id = B.id
       and E.signal θ1 signalpred(S.seq)
       and A.signal θ2 signalpred(S.seq)
       and E.site θ3 sitepred(S.seq)
       and B.site θ4 sitepred(S.seq)
```

Fig. 1. A SQL query template

While building execution plans for the query we follow one of three strategies. Strategy 1 ignores the costs of all (expensive) selections completely and considers only join costs. Strategy 2 takes into account the costs of selections, but does not look for common subexpressions. Finally, Strategy 3 considers the full selection costs as well as factorization. We determined the optimal left-deep tree for the example query using each of the strategies. When calculating the actual costs of a plan, we assumed that common subexpressions need only be evaluated once during query evaluation. The differences in costs are quite astonishing. For Strategy 1 the total costs were $1.3{*}10^{11}$, for Strategy 2 $1.1{*}10^{11}$, and for Strategy 3 $8.8{*}10^{9}$.

Contributions. The contributions of this paper are as follows:

- to describe and quantify the differences in evaluation costs if factorization is not considered by plan generators (the relevance of factorization has been overlooked for almost fifteen years in this area)

[1] e.g. SignalP [1] which correctly predicts the cleavage site in 70% of all cases.

- to present two plan generators that create optimal left-deep and bushy plans taking factorization into account
- to show experimental evidence that plan generation itself does not become more expensive if factorization is taken into account.

The rest of the paper is organized as follows. We formalize the problem in Section 2. Section 3 introduces algorithms that generate optimal left-deep and bushy tree plans while taking factorization into account. Section 4 presents experiments that verify our claim and demonstrate that plan generation performance does not suffer. Section 5 discusses related work, while Section 6 concludes the paper.

2 Formalization

In order to formalize the problem, we need — besides selection and join operators — the map operator (χ) [2] to evaluate UDF calls. It takes an expression as a subscript and evaluates the expression for each tuple in its input during query evaluation. The result of the evaluation is then stored in some additional (new) attribute. Formally, the map operator is defined as follows:

$$\chi_{a:e}(R) = \{t \circ [a : e(t)] \mid t \in R\}$$

where [ ] and $\circ$ represent tuple construction and concatenation. The expression $e(t)$ denotes the result of evaluating e using attribute values provided by the tuple t. The attribute name a must be new, that is, it may not be one of R. When it is of no further interest, we skip this attribute name and abbreviate $\chi_{a:e}$ as χ_e. So, for every UDF call in queries we introduce a map operator and a new attribute for storing the result and for every comparison a selection operator referring to this attribute. The other algebraic operators we use in this paper are the well-known selection and join operators.

The map operator is used to make the UDF calls explicit and visible to the reader. For our approach to work it is not really necessary to implement it in exactly this way. However, we think that introducing an explicit treatment of UDF calls makes the approach more readable.

Applying the above to our example query from the introduction results in the following set of map and selection operators:

$$\sigma_{\texttt{E.signal}\theta\texttt{sigp}}\ \sigma_{\texttt{A.signal}\theta\texttt{sigp}}\ \sigma_{\texttt{E.site}\theta\texttt{sitep}}\ \sigma_{\texttt{B.site}\theta\texttt{sitep}}$$
$$\chi_{\texttt{sigp:signalpred(S.seq)}} \qquad \chi_{\texttt{sitep:sitepred(S.seq)}}$$

When looking at the query in Figure 1, we can identify the following common subexpressions: the function `signalpred` and `sitepred` are called twice with the same parameter `S.seq`. Factoring these means that we only evaluate each of them once.

Generating the optimal evaluation plan for our query now boils down to finding an optimal ordering of maps, selections, and joins. In doing so, we have to pay attention to the following: when an operator's subscript refers to an attribute generated by a map, this map has to be evaluated before that operator. These dependencies induce a dependency graph.

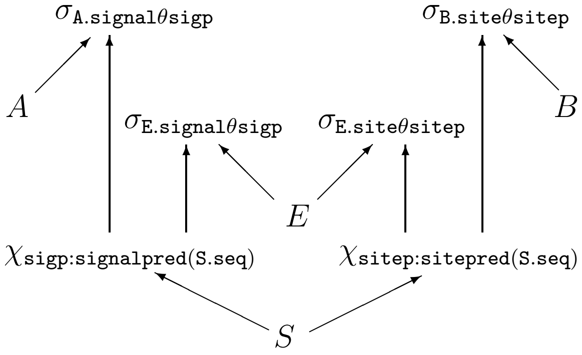

$\sigma_{\texttt{A.signal}\theta\texttt{sigp}}$
$\sigma_{\texttt{B.site}\theta\texttt{sitep}}$
A
B
$\sigma_{\texttt{E.signal}\theta\texttt{sigp}}$
$\sigma_{\texttt{E.site}\theta\texttt{sitep}}$
E
$\chi_{\texttt{sigp:signalpred(S.seq)}}$
$\chi_{\texttt{sitep:sitepred(S.seq)}}$
S

Fig. 2. Dependency graph for example query

2.1 Dependency Graph

The *dependency graph* captures the *consumer/producer* relationship of attributes. Its nodes are the relations and the selection and map operators of the query. There is an edge between two nodes n_1 and n_2, iff n_2 references attributes produced by n_1. For our example query, the dependency graph is shown in Figure 2. The notion of dependency graph and our subsequent algorithms can easily be extended to capture join predicates which require the evaluation of (expensive) UDFs. For ease of exposition, we do not consider this here.

If there is a path in the dependency graph from some χ_j to some σ_i, then the evaluation of σ_i depends on χ_j and we write $\chi_j \rightarrow^* \sigma_i$. We use the notation $\text{req}(\sigma_i)$ to describe the set of map operators a given selection σ_i depends on.

$$\text{req}(\sigma_i) = \{\chi_j \mid \chi_j \ \rightarrow^* \ \sigma_i\}$$

If $\chi_j \in \text{req}(\sigma_i)$ we say that σ_i *requires* χ_j.

As we factorize common subexpressions, we do not have to recalculate map operators that have already been computed in a (partial) plan P. By $\text{maps}(\sigma_i, P)$ we denote the set of map operators in $req(\sigma_i)$ which have not been evaluated in P yet:

$$\text{maps}(\sigma_i, P) = \text{req}(\sigma_i) \ \setminus \ \{\chi_j \mid \chi_j \text{ evaluated in } P\}$$

2.2 Cardinalities

We denote the cardinality of a relation R by $|R|$. As a map operation only adds attributes, but does not influence a relation in any other way, the cardinality of $|\chi_i(R)| = |R|$. The cardinality of the output of a selection σ_i (or a join $\bowtie_i$) depends on its selectivity sel_i, so $|\sigma_i(R)| = |R|{\cdot}\text{sel}_i$ (and $|L \bowtie_i R| = |L|{\cdot}|R|{\cdot}\text{sel}_i$).

2.3 Costs

Our results are cost model invariant. In particular, the algorithms work for any cost model. In order not to overcomplicate the matter, we use a very simple cost model.

Let cost(R) stand for the costs for scanning a relation R. For joins we assume main memory hash joins (only CPU costs) with an average chain length of 1.2:

$$\mathrm{cost}(L \bowtie_i R) = 1.2 \cdot |L| + |R| + 1.2 \cdot |R| \cdot j_i$$

where j_i is the cost for evaluating the join predicate.

Depending on the strategy used for optimizing the query plan, we have to distinguish different costs for the other operators. Each of the following paragraphs focuses on one strategy.

Zeroed Selection Costs. (Strategy 1 from the introduction) This is the simplest alternative. We just ignore the costs of map and selection operators by setting their cost to zero:

$$\begin{aligned}\mathrm{cost}(\chi_i(R)) &= 0\\ \mathrm{cost}(\sigma_i(R)) &= 0\end{aligned}$$

This strategy was employed by early plan generators and resulted in the well-known heuristic of pushing down selections in the query plan.

Maximized Selection Costs. Later it became apparent that selection costs are indeed important and that the traditional heuristics of pushing down selections can result in suboptimal plans (see Section 5 for references). However, factorization was not yet considered, which means that the costs for the selections may be overestimated. We call this model *maximized selection costs* (this is Strategy 2 from the introduction). The cost functions are

$$\begin{aligned}\mathrm{cost}(\chi_i(R)) &= |R| \cdot x_i\\ \mathrm{cost}(\sigma_i(R)) &= \sum_{\chi_j \in \mathrm{req}(\sigma_i)} \mathrm{cost}(\chi_j(R)) + |R| \cdot s_i\end{aligned}$$

where x_i are the costs of evaluating χ_i per tuple and s_i the costs of σ_i per tuple.

Correct Selection Costs. Last but not least we consider the costs for selections under factorization. We call these *correct selection costs* and they are computed as follows:

$$\begin{aligned}\mathrm{cost}(\chi_i(R)) &= |R| \cdot x_i\\ \mathrm{cost}(\sigma_i(R), P) &= \sum_{\chi_j \in \mathrm{maps}(\sigma_i, P)} \mathrm{cost}(\chi_j(R)) + |R| \cdot s_i\end{aligned}$$

This time we only consider the costs of map operators that have not yet been evaluated in P.

3 Algorithms

We present two algorithms for generating optimal query plans: one for generating left-deep trees and one for bushy trees. Depending on the cost model, these algorithms either ignore selections, consider the (full) costs of their required map operators or factorize the costs of the required map operators. Thus, we can use the same algorithms for all three strategies, only the way the costs of a plan are computed is different for each strategy.

3.1 Left-Deep Trees

We start with a relation R_i and add operators to it: $R_i o_{j_1} o_{j_2} o_{j_3} \ldots$ (with $o_{j_k} \in \{\sigma, \Join\}$). The join operators have an implicit right-hand side, which is determined by the current subplan. For example, let us assume that $\Join_{12}$ joins R_1 and R_2, $\Join_{23}$ joins R_2 and R_3, and σ_1 checks a selection predicate on attributes of R_1, then $R_3 \Join_{23} \Join_{12} \sigma_1$ describes the query plan $\sigma_1((R_3 \Join_{23} R_2) \Join_{12} R_1)$. Using this notation, left-deep trees can be generated by choosing one starting relation and then adding all permutations of joins and selections. By choosing each relation as the starting relation (in a partial plan) once, the optimal left deep tree can be found.

```
// Input:      $P$: prefix of a complete plan
// Output: optimal sequence starting with $P$
left-deep($P$) {
  $O = \{o \mid o$ missing in $P\}$
  if($|O| > 0$) {
    $P' = \{p \mid p$ contained in $P\}$;
    if (memoization table contains entry for $P'$) {
      return $P \circ P'$;
    }
    for each $o$ in $O$ {
      $M_o = \{\chi \mid \chi$ required by $o\}$;
      $P_o = P \circ M_o \circ o$;
      if ($P_o$ is valid) {
        $C_o =$ left-deep($P_o$);
      }
    }
    $C = C_o$ with the smallest costs;
    add entry $C \setminus P$ for $P'$ to memoization table;
    return $C$;
  }
  else {
    return $P$;
  }
}
```

Fig. 3. Generating left-deep trees with memoization

Figure 3 shows our algorithm to generate an optimal completion of a partial plan P. If we know from our memoization table how to complete this plan optimally, we are already done. That is, it memoizes for a given (unordered) set of operators P' an (ordered) sequence that will complete a partial plan missing these operators optimally. If we do not find an entry in the memoization table, we consider all operators not yet in P in turn; for each of them, we extend P by that operator and all required map operators, compute the best completion for this new partial solution (recursively) [2] and store the completion. We regard partial plans to be equivalent (for the purpose of storing and retrieving completions) if they are permutations of each other.

3.2 Bushy Trees

For generating an optimal bushy tree we use an iterative dynamic programming algorithm. It first creates all query plans that consist of just a single relation (representing the scan of this relation). It then combines existing plans with operators that have not been applied yet and with other existing plans, creating larger plans. In each step it only keeps the cheapest plan for a given set of operators (e.g. either $R_1 \bowtie R_2$ or $R_2 \bowtie R_1$). In the last step of the outer loop plans containing all operators are generated. The pseudocode of the algorithm is shown in Figure 4. It uses the function `ops()` which returns the set of all operators included in a plan. Further, $\mathcal{A}(P)$ denotes the set of attributes produced by the (sub)plan P. By $\mathcal{F}(\chi)$ we denote the set of attributes that are needed to evaluate the map operator χ.

4 Evaluation

4.1 General Description

In order to estimate the effect of the different strategies, we created randomized chain and star queries[3] for different numbers of relations. Due to space constraints, we only present the results for chain queries here. Nevertheless, the results for star queries are similarly encouraging.

The term chain query refers to the shape of the query graph and means that each relation R_i (except for the first and last relation) is connected to its neighbors R_{i-1} and R_{i+1}. For each query graph size we constructed 1000 queries and averaged their results. The optimal execution plan was found for each of the three different costing strategies.

We measured the average runtime, average actual costs [4] relative to the optimum, and the average number of generated plans. Actual costs mean that for comparing the costs of plans we assumed that the underlying query engine was always able to factorize common subexpressions and computed the actual costs

[2] Plans are checked for validity: attributes referenced by χ must be present in P; attributes referenced by σ must be present in $P \circ M_o$; one of the argument relations of $\bowtie$ must be contained in P and join attributes in $P \circ M_o$.

[3] Chain and star queries (using a star schema) reflect typical queries in real-world database systems.

[4] For the evaluation of selection and join predicates we assumed a cost of 1, for evaluating map operators a cost between 100 and 800 (determined randomly), while the cardinality of the relations ranged from 500 to 10,000 (also determined randomly).

```
// Input:  $R = \{R_1, \dots, R_m\}$
//         $O = \{\sigma_1, \dots, \sigma_n, \bowtie_1, \dots, \bowtie_{m-1}\}$
// Output: optimal bushy tree
bushy($R, O$) {
  for each relation $R_i \in R$ {
    create an optimal access path for $R_i$;
    store this plan;
  }
  for $i = 1$ to $|O|$ {
    for each $\sigma \in O$ {
      for all plans $e$ with $|\texttt{ops}(e)| = i - 1$ and $\sigma \notin \texttt{ops}(e)$ {
        $M_\sigma = \{\chi \mid \chi \texttt{ required by } \sigma\}$;
        $P_\sigma = \sigma(e \circ M_\sigma)$;
        if($P_\sigma$ is valid) {
          keep $P_\sigma$ if it is the cheapest plan with
            operators identical to $\texttt{ops}(P_\sigma)$;
        }
      }
    }
    for each $\bowtie \in O$ {
      for all plans $l$, $r$ with $|\texttt{ops}(l)| + |\texttt{ops}(r)| = i - 1$
        and $\bowtie \ \notin \ (\texttt{ops}(l) \cup \texttt{ops}(r))$ {
        $M^l_\bowtie = \{\chi \mid \chi \texttt{ required by } \bowtie, \mathcal{F}(\chi) \subseteq \mathcal{A}(l)\}$;
        $M^r_\bowtie = \{\chi \mid \chi \texttt{ required by } \bowtie, \mathcal{F}(\chi) \subseteq \mathcal{A}(r)\}$;
        $P_\bowtie = (l \circ M^l_\bowtie) \bowtie (r \circ M^r_\bowtie)$;
        if($P_\bowtie$ is valid) {
          keep $P_\bowtie$ if it is the cheapest plan with
            operators identical to $\texttt{ops}(P_\bowtie)$;
        }
      }
    }
  }
  return the cheapest plan with $|O|$ operations;
}
```

Fig. 4. Generating bushy tree permutations

of each plan taking factorization into consideration. During the optimization, on the other hand, the three different strategies used various cost models ignoring (zeroed, maximized) or taking into account factorization (correct).

4.2 Results for Left-Deep Trees

The results for left-deep trees are shown in Figure 5(a) (runtime of the algorithm), Figure 5(b) (number of generated plans), and Figure 5(c) (relative costs to optimal plan). Comparing the runtime and the number of generated plans we notice that all strategies are on a par with each other. They all generated exactly the same number of plans. This should not come as a surprise, since all strategies were implemented by the same algorithm. The only difference was that the zeroed and maximized strategy were slightly faster. The correct costs strategy had a very slightly higher per-plan overhead.

When looking at the evaluation costs of the different strategies, however, we see huge differences (Figure 5(c); please note the logarithmic scale). Already for a moderate number of relations to be joined the plans generated by the zeroed and maximized strategy are off by a factor of more than 1000. While the maximized strategy usually behaves better than the zeroed strategy, this was not always the case. Its overestimation of the costs of map operators resulted in plans that applied selections too late, which is as bad as applying them too early.

4.3 Results for Bushy Trees

For bushy trees the average runtime and the corresponding average number of generated plans for the different strategies are shown in Figure 5(d) and 5(e). The zeroed strategy was slightly better here, as the optimizer quickly decided to push down selections and therefore had to generate fewer plans. The correct and the maximized strategy generated about the same number of plans. Interestingly, the correct strategy generated slightly fewer plans than the maximized strategy. This is due to the fact that the maximized strategy overestimated the costs, which resulted in less pruning. The difference in average number of generated plans was greater than the runtime difference. This is due to the fact that the correct strategy had a slightly higher per-plan overhead which compensated for some of the savings in the number of plans.

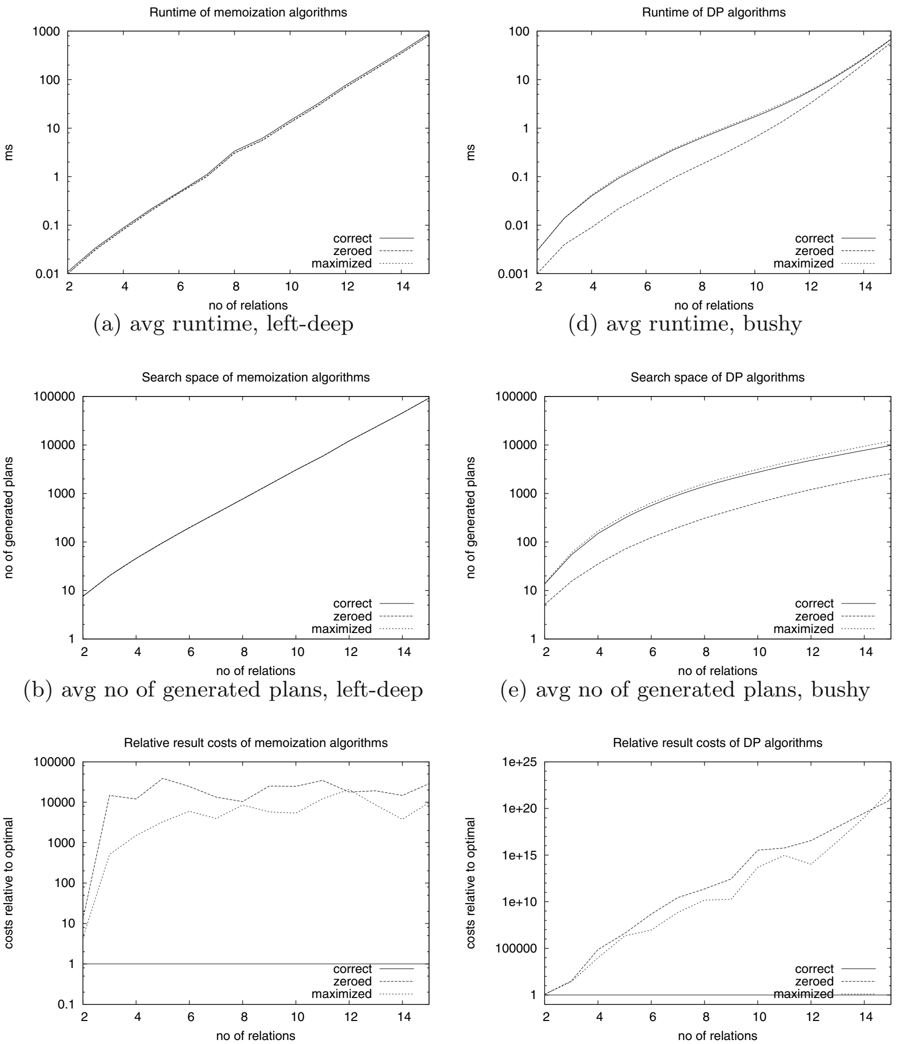
When looking at the error made by the different strategies for bushy trees, it becomes clear that both zeroed and maximized costs produced even larger errors than for left-deep trees. Figure 5(f) shows the average ratio between the actual costs for the produced plans and the optimal plan. In contrast to left-deep trees, where the error seems to level off at a certain point, for bushy trees the error kept growing with the number of relations (again, note the logarithmic scale).

5 Related Work

Related work comes from two areas: (1) optimization techniques for ordering expensive selection predicates and joins and (2) optimization techniques for the efficient evaluation of boolean expressions containing expensive function calls (but not considering joins). The former area is more closely related to this paper.

The problem of ordering expensive selections and joins is considered by many different authors [5, 6, 7, 8, 10, 16, 17, 19]. However, none of these papers considers factoring out common subexpressions. As we have seen, this can lead to suboptimal plans. Furthermore, experiments presented in Section 4 indicate that the plans generated by any of the approaches cited above may be several orders of magnitude worse than the best plan.

Let us now briefly discuss the second area of related work. Kemper et al. discuss several approaches to optimize complex boolean expressions with expensive selection predicates. [12, 13, 18]. However, neither factoring out common subexpressions nor joins are considered. Neumann et al. discuss the optimal ordering



(a) avg runtime, left-deep

(d) avg runtime, bushy

(b) avg no of generated plans, left-deep

(e) avg no of generated plans, bushy

(c) avg relative result costs, left-deep

(f) avg relative result costs, bushy

Fig. 5. Results for evaluating algorithms (please note the logarithmic scales)

of selections taking factorization into account. However, they do not consider joins yet [14]. Only a single paper by Chaudhuri et al. explicitly handles the problem of factorization [4]. However, they do so at the level of physical query optimization. Their work deals with the evaluation of complex boolean predicates (containing many simple comparisons between attribute values and constants) by considering how to merge the results of several index scans optimally. Hellerstein and Naughton discuss caching results for expensive UDFs [9]. While this is an important performance improving technique, it does not solve the problem of finding an optimal evaluation order. In fact, whether results of UDF calls are cached or not is orthogonal to the issue presented here.

6 Conclusion

Early work in plan generation applied the heuristics of pushing selection predicates as far down as possible. Later, it was shown that applying these heuristics in case of expensive predicates can lead to suboptimal plans. Several algorithms have been devised to order expensive selections and joins. The fact that expensive selection predicates can share expensive subexpressions (e.g. in the form of UDF calls) is ignored by all these algorithms. We have shown that ignoring the benefits of sharing may lead to highly inefficient plans.

Motivated by this finding, we proposed two algorithms that optimally order joins, selections, and maps while taking factorization of shared subexpressions into account. The first algorithm produces optimal left-deep trees, the other optimal bushy trees. Using these algorithms we carried out several experiments to illustrate the optimization potential gained when considering common subexpressions. In every experimental setting, improvements of several orders of magnitudes are the rule.

Generating better plans is typically achieved by generating more alternatives, that is by extending the search space of the plan generator (e.g. by going from left-deep trees to bushy trees). Unfortunately, this usually leads to higher runtime and memory consumption during plan generation. Not so for our algorithms; except for the zeroed cost strategy for bushy plans, which generated very bad plans and was slightly faster, there was no noticeable difference in the runtime of the different strategies.

References

1. J. Bendtsen, H. Nielsen, G. von Heijne, and S. Brunak. Improved prediction of signal peptides: SignalP 3.0. *J. Mol. Biol.*, 340:783–795, 2004.
2. R. Bird and P. Wadler. An Introduction to Functional Programming. Prentice Hall, 1988.
3. G. Blobel. Protein targeting. *ChemBioChem*, 1:86–102, 2000.
4. S. Chaudhuri, P. Ganesan, and S. Sarawagi. Factorizing complex predicates in queries to exploit indexes. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 361–372, 2003.
5. S. Chaudhuri and K. Shim. Optimization of queries with user-defined predicates. In *Proc. Int. Conf. on Very Large Data Bases (VLDB)*, pages 87–98, 1996.
6. S. Chaudhuri and K. Shim. Optimization of queries with user-defined predicates. *ACM Trans. on Database Systems*, 24(2):177–228, 1999.
7. J. Hellerstein. Predicate migration: Optimizing queries with expensive predicates. Computer Science Division, EECS Department, University of California at Berkeley, CA 94720, Dec. 1992.
8. J. Hellerstein. Practical predicate placement. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 325–335, 1994.

9. J. Hellerstein and J. Naughton. Query execution techniques for caching expensive methods. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 423–434, 1996.
10. J. Hellerstein and M. Stonebraker. Predicate migration: Optimizing queries with expensive predicates. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 267–277, 1993.
11. T. Ibaraki and T. Kameda. On the optimal nesting order for computing n-relational joins. *ACM Transactions on Database Systems*, 9(3):482–502, September 1984.
12. A. Kemper, G. Moerkotte, K. Peithner, and M. Steinbrunn. Optimizing disjunctive queries with expensive predicates. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 336–347, 1994.
13. A. Kemper, G. Moerkotte, and M. Steinbrunn. Optimization of boolean expressions in object bases. In *Proc. Int. Conf. on Very Large Data Bases (VLDB)*, pages 79–90, 1992.
14. T. Neumann, S. Helmer, and G. Moerkotte. On the optimal ordering of maps and selections under factorization. In *Proc. IEEE Conference on Data Engineering*, pages 490–501, 2005.
15. K. Ono and G. Lohman. Measuring the complexity of join enumeration in query optimization. In *Proc. Int. Conf. on Very Large Data Bases (VLDB)*, pages 314–325, 1990.
16. W. Scheufele and G. Moerkotte. Optimal ordering of selections and joins in acyclic queries with expensive predicates. Technical Report 96-3, RWTH-Aachen, 1996.
17. W. Scheufele and G. Moerkotte. Efficient dynamic programming algorithms for ordering expensive joins and selections. In *Proc. of the Int. Conf. on Extending Database Technology (EDBT)*, pages 201–215, 1998.
18. M. Steinbrunn, K. Peithner, G. Moerkotte, and A. Kemper. Bypassing joins in disjunctive queries. In *Proc. Int. Conf. on Very Large Data Bases (VLDB)*, pages 228–238, 1995.
19. K. Yajima, H. Kitagawa, K. Yamaguchi, N. Ohbo, and Y. Fujiwara. Optimization of queries including adt functions. In *International Symposium on Database Systems for Advanced Applications*, pages 366–376, 1991.

An I/O Optimal and Scalable Skyline Query Algorithm

Yunjun Gao, Gencai Chen, Ling Chen, and Chun Chen

College of Computer Science, Zhejiang University, Hangzhou, 310027, P.R. China
{gaoyj, chengc, lingchen, chenc}@cs.zju.edu.cn

Abstract. Given a set of d-dimensional points, skyline query returns the points that are not dominated by any other point on all dimensions. Currently, BBS (branch-and-bound skyline) is the most efficient skyline processing method over static data in a centralized setting. Although BBS has some desirable features (e.g., I/O optimal and flexibility), it requires large main-memory consumption. In this paper, we present an improved skyline computation algorithm based on best-first nearest neighbor search, called IBBS, which captures the optimal I/O and less memory space (i.e., IBBS visits and stores only those entries that contribute to the final skyline). Its core enables several effective pruning strategies to discard non-qualifying entries. Extensive experimental evaluations show that IBBS outperforms BBS in both scalability and efficiency for most cases, especially in low dimensions.

1 Introduction

Skyline query is one of important operations for several applications involving multi-criteria decision making, and has received considerable attention in the database community. Given a set of d-dimensional points $P = \{p_1, p_2, \ldots, p_n\}$, the operator returns a set of points p_i, which is not dominated by any other point p_j in P on all dimensions, forming the skyline of P. A point dominates another one if it is *as good or better in all dimensions and better in at least one dimension* [19]. Consider, for instance, a common example in the literature, "*choosing a set of hotels that is closer to the beach and cheaper than any other hotel in distance and price attributes respectively from the database system at your travel agents'* [3]". Figure 1(a) illustrates this case in 2-dimensional space, where each point corresponds to a hotel record. The room price of a hotel is represented as the x-axis, and the y-axis specifies its distance to the beach. Clearly, the most interesting hotels are the ones $\{a, g, i, n\}$, called *skyline*, for which there is no any other hotel in $\{a, b, \ldots, m, n\}$ that is better on both dimensions. For simplicity, in this paper, we use the *min* condition on all dimensions to compute the skyline, even though the proposed algorithm can be easily applied to different conditions (e.g., *max* metric). Using the *min* condition, a point p is said to dominate another one q if (i) p is not larger than q in all dimensions, and (ii) p is strictly smaller than q in at least one dimension. This implies that p is preferable to q for the users in real life. Continuing the running example,

D. Bell and J. Hong (Eds.): BNCOD 2006, LNCS 4042, pp. 127–139, 2006.

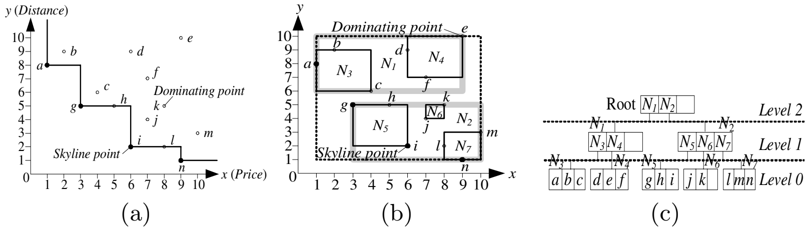


Fig. 1. Example of skyline and an R-tree in 2-dimensional space

hotel a dominates hotels b, d, and e because the former is nearer to the beach and cheaper than the latter.

Skyline query processing has been extensively studied, and a large number of algorithms have been also proposed [1, 3, 5, 7, 9, 12, 13, 15, 16, 19, 21]. These methods can be mainly divided into two categories. Specifically, (i) non-index-structure-based schemes, which do not assume any index structure on the underlying dataset, but compute the skyline through scanning the entire dataset at least once, resulting in expensive CPU overhead; (ii) index-structure-based solutions, which significantly reduce CPU and I/O costs by performing the skyline retrieval on an appropriate index structure (e.g., R*-tree [2]). We concentrate on the second category in this paper. In addition, the skyline computation problem is also closely related to several other well-known problems that have been extensively investigated in the literature, such as *convex hull* [4, 17], *top-k* queries [6, 8, 11, 14], and *nearest neighbor* search [10, 18].

Currently, BBS (branch-and-bound skyline), presented by Papadias *et al.* in [15], is the most efficient skyline query algorithm over static datasets in a centralized setting. It employs a best-first based nearest neighbor search paradigm [10] on dataset indexed by R*-tree. BBS minimizes the I/O overhead, and the considerable experiments of [15] show that it outperforms previous algorithms in terms of CPU and I/O costs for all problem instances. Although BBS has some desirable advantages, it yet needs large memory space. As reported in [15], the heap size of BBS is larger than the *to-do* list size of NN [12] in 2-dimensional space. In fact, we can greatly reduce space consumption used by the heap and speed up the execution of the algorithm via several dominance checking based pruning heuristics (discussed detailedly in Section 3 of this paper) for filtering all the non-qualifying entries that may not contain (become) any skyline point. As known, the less the memory space requires, the more scalable the algorithm is. Thus, in this paper, we present an improved skyline query algorithm, called IBBS, which, like BBS, is depended on best-first nearest neighbor search on R*-tree, whereas (unlike BBS) it enables several effective pruning strategies to discard unnecessary entries. IBBS incorporates the virtues of BBS (e.g., I/O optimal, low CPU cost, etc.), while gaining less main-memory consumption (i.e., smaller heap size). Finally, extensive experiments with synthetic datasets confirm that IBBS outperforms BBS in both efficiency and scalability for most cases, especially in low dimensions.

The rest of the paper is organized as follows. Section 2 reviews existing algorithms for skyline queries, focusing more on BBS as it is more related to our work. Section 3 describes IBBS, together with some pruning heuristics and a proof of its memory space optimality. Section 4 experimentally evaluates IBBS, comparing it against BBS under various setting. Section 5 concludes the paper with some directions for future work.

2 Related Work

To our knowledge, Borzsonyi *et al.* [3] first introduce the skyline operator in the database context and develop two skyline computation methods including divide-and-conquer (D&C) and block-nested-loop (BNL). Chomicki *et al.* [7] present a sort-first-skyline (SFS) algorithm as an improved version of BNL. Tan *et al.* [19] propose the first *progressive* technique that can return skyline points instantly, and develop two solutions for skyline queries, termed Bitmap and Index, respectively. Another two progressive skyline query algorithms, nearest neighbor (NN) and branch-and-bound skyline (BBS), are proposed by Kossmann *et al.* [12] and Papadias *et al.* [15], respectively, based on nearest neighbor search [10, 18] on datasets indexed by R-trees. The great difference of both algorithms is that NN requires multiple nearest neighbor queries, but BBS executes only a single retrieval of the tree. Furthermore, BBS guarantees the minimum I/O cost. Since our work in this paper is more related to BBS, the following discussion describes its executive steps, using an illustrative example.

Table 1. Execution of BBS

Action	**Heap Contents**	S
Visit root	$(N_2, 4)$,$(N_1, 7)$	$\emptyset$
Expand N_2	$(N_5, 5)$,$(N_1, 7)$,$(N_7, 9)$,$(N_6, 11)$	$\emptyset$
Expand N_5	$(N_1, 7)$,$(g, 8)$,$(i, 8)$,$(N_7, 9)$,$(h, 10)$,$(N_6, 11)$	$\emptyset$
Expand N_1	$(N_3, 7)$,$(g, 8)$,$(i, 8)$,$(N_7, 9)$,$(h, 10)$,$(N_6, 11)$,$(N_4, 13)$	$\emptyset$
Expand N_3	**$(g,$8$)$**,**$(i,$8$)$**,**$(a,$9$)$**,$(N_7, 9)$,$(c, 10)$, $(h, 10)$,$(b, 11)$,$(N_6, 11)$,$(N_4, 13)$	$\{g, i, a\}$
Expand N_7	~~$(c,10)$~~,~~$(h,10)$~~,**$(n,$10$)$**,~~$(b,11)$~~, ~~$(N_6,11)$~~,~~$(N_4,13)$~~	$\{g, i, a, n\}$

As an example, suppose that we use the 2-dimensional dataset of Figure 1(a), organized in the R-tree of Figure 1(c), together with the minimum bounding rectangles (MBRs) of the nodes whose capacity is 3. Note that the distances from an intermediate entry (e.g., N_3, N_4, etc.) or a data point (e.g., a, b, etc.) to the beginning of the axes are computed according to L_1 norm, that is, the *mindist* of a point equals the sum of its coordinates (e.g., $mindist(g) = 3 + 5 = 8$) and the *mindist* of a MBR (i.e., intermediate entry) equals the *mindist* of its lower-left corner point (e.g., $mindist(N_5) = 3 + 2 = 5$). Initially, all the entries in the root node are inserted into a heap H sorted in ascending order of their *mindist*. Then, BBS circularly computes the skyline until H becomes empty. Each circulation, it first removes the top entry E with the minimum *mindist* from H and accesses it. Here are two cases. (i) If E is an intermediate entry and not dominated by the existing skyline points, the algorithm en-heaps its child entries there. (ii) If E is a data point and not dominated by the skyline points obtained, the algorithm inserts it into the list S of the skyline as a skyline point. Table 1 illustrates the executive steps of BBS. Also notice that skyline points discovered are bold and pruned entries are shown with strikethrough fonts.

Recently, Balke *et al.* [1] extend the skyline computation problem for the web information systems. Lin *et al.* [13] study continuous skyline monitoring on data streams. Chan *et al.* [5] consider skyline computation for partially-ordered domains. Godfrey *et al.* [9] present an algorithm, called linear-elimination-sort for skyline (LESS), which has attractive worst-case asymptotical performance. Pei *et al.* [16] and Yuan *et al.* [21] independently present the skyline cube consisting of the skylines in all possible subspace.

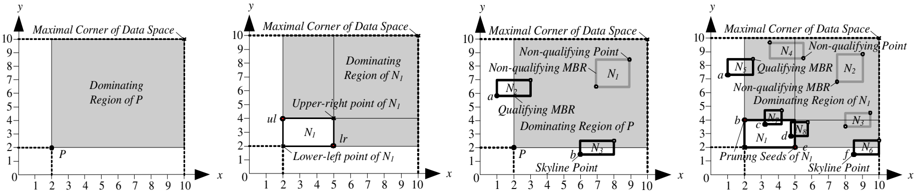
3 Improved Branch-and-Bound Skyline Algorithm

3.1 Dominance Checking Based Pruning Strategy

Consider the execution of BBS demonstrated in Table 1 of Section 2, some non-qualifying entries for the skyline are existed in the heap. For example, entry N_6 inserted into the heap after expanding entry N_2 is such one because it completely falls into the dominating region (DR for short) of entry N_5, and may not contain any skyline point. Similarly, entries h, N_4, b, and c are all redundant ones. Thus, they must be discarded, and not be en-heaped there. In fact, only those entries that may potentially contain (or become) the skyline points (e.g., N_3, g of Figure 1(b)) are needed to be kept in the heap. Based on this observation, it may be helpful to identify these entries and prevent them from being inserted in the heap. Fortunately, we can achieve this goal by careful dominance checking for each entry before it is inserted in the heap. Next, we present several pruning heuristics that is inspired by the analysis of per entry's DR.

Let S be a set of the skyline points, there must be at least one entry E_j ($j \neq i$) that dominates E_i, if an entry E_i does not appear in S. If we can know E_j when inserting E_i into the heap, E_i can be discarded immediately as it is an unnecessary entry for the skyline. However, the problem is now how to get such E_j in order to safely prune E_i. Since such a E_j must come from all the entries that have been visited by the algorithm, we can pick possible E_j from them. To address this problem, we need to consider for an entry E_j its ability to dominate and then prune other entries. For simplicity, we take 2-dimensional data space into account in the following discussion. However, similar conclusions also hold for d-dimensional ($d > 2$) data space.

Toward a point entry P with coordinates (x_1, x_2), its ability to dominate other entries, including point and intermediate entries (i.e., MBRs), is determined by its own values and the boundaries of the data space, that is, the rectangle whose diagonal is the line segment with P and the maximum corner of the data space as coordinates. Any other entry that resides in that region is dominated by P and it excluded from the final skyline. For this reason, we call that rectangle



(a) DR of P (b) DR of N_1 (c) Pruning with P (d) Pruning with N_1

Fig. 2. Illustration of DRs of one point, one MBR, and their pruning ability

the DR of P. Intuitively, the larger P's DR is, the more other entries are dominated by P because a larger rectangle covers more entries in the data space, especially for the entries following independent (uniform) distribution. For ease of comprehension, we use a 2-dimensional illustration as shown in Figure 2 in this discussion. Specifically, Figure 2(a) shows the DR of P (represented as the shaded rectangle), assuming that the maximum corner of data space is 10. The P's ability to dominate other entries is plotted in Figure 2(c) under the same supposition as Figure 2(a). Clearly, in Figure 2(c), entry N_1 is dominated by point P since it fully falls into the DR of P. Hence, N_1 can be pruned instantly, and need not to be inserted into the heap for further consideration. However, entries N_2 and N_3 that intersect the P's DR must en-heap there, because they may contain some skyline points (e.g., a and b). Therefore, we call that N_1 the non-qualifying MBR, but N_2 and N_3 the qualifying MBRs.

Assume that the boundaries of an intermediate entry N are $[{x_1}^l, {x_1}^h] \times [{x_2}^l, {x_2}^h]$, then the coordinates of its lower-left, lower-right, upper-left, and upper-right corners are the points $({x_1}^l, {x_2}^l)$, $({x_1}^h, {x_2}^l)$, $({x_1}^l, {x_2}^h)$, and $({x_1}^h, {x_2}^h)$, respectively. Thus, the DR of N is defined by its own boundaries and the boundaries of the data space. As an example, the DR of entry N_1 is shown in Figure 2(b) (specified the shaded area). From the diagram, we can also see that the N_1's DR is determined by the upper-left, bottom-right corners of N_1 (denoted as two red points), and the maximal corner of data space, respectively. Specifically, let ul be the upper-left vertex of N_1 and lr the lower-right vertex of N_1, then the DR of N_1 is the union of the dominating regions of ul and lr, i.e., formally, DR (N_1) = DR (ul) DR (lr). So, it implies that the N_1's ability to dominate other entries can be done by dominance checking with ul and lr. For this reason, we term that ul and lr the pruning seeds of N_1.

Those entries, including points and MBRs, which completely fall into the DR of N_1 are dominated by N_1, and they must not appear in the final skyline. As known, an intermediate entry N is dominated by a point entry P with coordinates (x_1, x_2) only if its bottom-left corner is dominated by P. Similarly, N is dominated by another intermediate entry N' only if the lower-left vertex of N resides in the DR of N'. Figure 2(d) demonstrates the pruning with a MBR N_1, where the pruning seeds of N_1 are denoted as two red points (i.e., b and e). Evidently, entries N_2, N_3, and N_4 are non-qualifying ones since they are dominated by N_1. As a result, they can be discarded in security, and have not to be inserted in the heap. Entries N_5, N_6, N_7, and N_8, however, need be stored in the heap in order to access them later, because their lower-left corners are not dominated by N_1, and they may contain some skyline points, such as points a, c, d, and f of Figure 2(d) are such ones.

In summary, we can derive the following pruning heuristics to prune the non-qualifying entries for the skyline based on the above discussion. Suppose that, in d-dimensional data space, two point entries P and P' with coordinates $(x_1, x_2, \ldots, x_d)$ and $(x'_1, x'_2, \ldots, x'_d)$ respectively, and two intermediate entries N and N' with boundaries $[{x_1}^l, {x_1}^h] \times [{x_2}^l, {x_2}^h] \times \ldots \times [{x_d}^l, {x_d}^h]$ and $[{x'_1}^l, {x'_1}^h] \times [{x'_2}^l, {x'_2}^h] \times \ldots \times [{x'_d}^l, {x'_d}^h]$, respectively. Then several pruning heuristics are developed as follows.

Heuristic 1. If P' is dominated by P, i.e., (i) $x_i \leq x'_i$ for i $\in [1, d]$, and (ii) $x_i < x'_i$ in at least one dimension, then it can be safely pruned immediately and not be inserted into the heap, since it must not appear in the skyline.

Heuristic 2. If the bottom-left corner of N is dominated by P, then N can be also safely discarded and not be en-heaped there, as it must not contain any skyline point.

Heuristic 3. If P is dominated by N, that is, P fully falls into the N's DR, then P can be safely removed instantly and excluded from the heap, because it may not become a skyline point.

Heuristic 4. If N dominates N', namely, the lower-left corner of N' fully resides in the DR of N, then N' can be also safely discarded immediately and not be kept in the heap, since it may not contain any skyline point.

3.2 Algorithm Description

Like BBS, IBBS is also based on best-first nearest neighbor search. Although IBBS can be applied to various multi-dimensional access methods, in this paper, we assume that the dataset is indexed by an R*-tree due to its efficiency and popularity in the literature. Unlike BBS, IBBS enables several effective pruning heuristics to discard non-qualifying entries in order to greatly decrease the memory space and speed up its execution. In particular, IBBS incorporates two pruning strategies. The first one is that when expanding an intermediate entry, all entries dominating each other in its child nodes are removed according to heuristics 1 to 4 (proposed in Section 3.1 of this paper). The other one involves pruning strategy that checks the contents of the heap H before the insertion of an entry E. If E is dominated by some entry in H, it is pruned immediately and not en-heaped there. In contrast, E is stored in H, and all entries in H that are dominated by it are also discarded accordingly.

The pseudo-code of IBBS is shown in Figure 3. Note that an entry is checked for dominance twice: before it is inserted into the heap and before it is expanded. Furthermore, the algorithm also implements pruning twice. Specifically, line 13 filters all entries dominated by some entry in the heap. Line 15 excludes all entries dominating each other from the heap. Thus, only those entries that contribute to the final skyline are maintained in the heap, such that the maximum heap size (i.e., memory consumption) is reduced by factors, as well as the CPU cost is decreased accordingly.

```
Algorithm IBBS (R-tree R)
/* S is used to keep the final skyline. */
S = Ø;
Insert all entries in the root of R into the heap H;
While H is not empty do;
    Remove the first entry E from H;
    If E is dominated by any point in S then
        Discard E;
    Else   // E is not dominated by any point in S
        If E is a data point then
            Insert E into S;
        Else   // E is an intermediate entry
            For each child entry E_i of E do
                If E_i is not dominated by any point in S then
                    If E_i is not dominated by any entry in H then
                        Insert E_i into H;
            Prune all entries dominating each other in H by heuristics 1 to 4;
End while
End IBBS
```

Fig. 3. Pseudo-code of an IBBS algorithm

Table 2. Execution of IBBS

Action	Heap Contents	S
Visit root	$(N_2,4)$,$(N_1,7)$	Ø
Expand N_2	$(N_5,5)$,$(N_1,7)$,$(N_7,9)$	Ø
Expand N_5	$(N_1,7)$,$(g,8)$,$(i,8)$,$(N_7,9)$	Ø
Expand N_1	$(N_3,7)$,$(g,8)$,$(i,8)$,$(N_7,9)$	Ø
Expand N_3	$\boldsymbol{(g,8)}$,$\boldsymbol{(i,8)}$,$\boldsymbol{(a,9)}$,$(N_7,9)$	$\{g,i,a\}$
Expand N_7	$\boldsymbol{(n,10)}$	$\{g,i,a,n\}$

Continuing the example of Figure 1, for instance, let us give an illustrative example of IBBS to simulate its executive steps for skyline query. Initially, all the entries in the root node are inserted into a heap H sorted in ascending order of their *mindist*, resulting in $H = \{(N_2, 4), (N_1, 7)\}$. Then, the algorithm removes the top entry (i.e., N_2) from H, visits it, and en-heaps its children there, after which $H = \{(N_5, 5), (N_1, 7), (N_7, 9)\}$. Here N_6 is discarded since it is dominated by N_5. Similarly, the next expanded entry is N_5 with the minimum *mindist*, in which the data points are added into $H = \{(N_1, 7), (g, 8), (i, 5), (N_7, 9)\}$. Also notice that h is pruned as it is dominated by g. The algorithm proceeds in the same manner until H becomes empty. The final list S of skyline points becomes $S = \{g, i, a, n\}$. As with the settings of Table 1, Table 2 illustrates the execution of IBBS. From Table 2, we can see that the heap size of IBBS is smaller significantly than that of BBS, which is also verified by the experiments in the next section of this paper.

3.3 Discussion

In this section, we focus on proving the memory space optimality of IBBS, and omit the proofs of its correctness and I/O optimality because they are similar to those of BBS in [15].

Lemma 1. *If an entry E, either an intermediate entry or a data point entry, does not inserted into the heap H, then there must exist another entry E' in H or some skyline point discovered that dominates E.*

Proof. The proof is straightforward since our proposed pruning heuristics discard all entries that are dominated by any other entry in the given dataset before they are en-heaped there. □

Lemma 2. *All entries that may contain (or become) skyline points must be kept in the heap H.*

Proof. Clear, by Lemma 1, all entries in H must not be dominated by any other entry in the given dataset. Also, they act on the skyline, that is, they either contain some skyline points or are skyline points. Thus, Lemma 2 holds. □

Theorem 1. *The main-memory consumption for IBBS is optimal.*

Proof. The Theorem 1 trivially holds, since Lemmas 1 and 2 ensure that the heap H used by IBBS only stores those entries that may contain (or be) skyline points, as well as they are inserted into H at most once by their *mindist*. □

4 Experimental Evaluation

This section experimentally verifies the efficiency of IBBS by comparing it with BBS under a variety of settings. We implemented two versions of IBBS, called IBBS-OS and IBBS-WOS, respectively. Specifically, IBBS-OS incorporates two pruning strategies (described in Section 3.2), but IBBS-WOS employs only the first one, i.e., when expanding an entry, all its child entries dominating each other are removed by heuristics 1 to 4. All algorithms (involving IBBS-OS, IBBS-WOS, and BBS) were coded in C++ language. All experiments were performed on a Pentium IV 3.0 GHz PC with 1024 MB RAM running Microsoft Windows XP Professional. We considered L_1 norm to compute *mindist* from the origin of the data space in all experiments.

4.1 Experimental Settings

Following the common methodology in the literature, we generated three synthetic datasets conforming the independent (uniform), correlated, and anti-correlated, respectively. Figure 4 illustrates such datasets with cardinality N = 10000 and dimensionality d = 2. We utilized these datasets with d varied between 2 and 5, and N in the range [100k, 10M]. All datasets are indexed by R*-tree [2], whose node size was fixed to 4096 bytes resulting in node capacities altered 204 (d = 2), 146 (d = 3), 113 (d = 4), and 92 (d = 5), respectively. All experiments examined several factors, including d, N, and *progressive* behavior, which affect the performance of the algorithms.

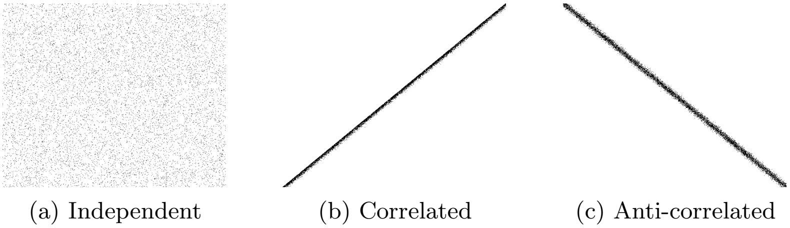

(a) Independent (b) Correlated (c) Anti-correlated

Fig. 4. Illustration of three synthetic datasets

4.2 Experimental Results

The first set of experiments studies the effect of dimensionality d using the datasets with $N = 1\text{M}$ and d varied from 2 to 5. Figure 5 shows the maximum size of the heap (in Kbytes) as a function of d. Clearly, the maximal heap size (MHS for short) of both IBBS-OS and IBBS-WOS almost equals under various dimensionalities, which implies that most of non-qualifying entries are pruned after doing the first pruning strategy, and few entries can be further discarded via the second one. As validated again in the following experiments. For all datasets, however, the MHS of IBBS-WOS is greatly smaller than that of BBS, especially in low dimensions. Notice that the difference of both algorithms decreases gradually with the dimensionality, since the larger overlap among the MBRs at the same level of R-trees occurs in the high-dimensionality [20]. Despite the gain of IBBS-WOS reduces in this case (e.g., $d = 5$), it is yet less than BBS, which is also pointed out by the number at the side of each polyline in the diagrams.

Figure 6 illustrates the number of node access versus d. From these graphs, we can see that three algorithms display the same efficiency for all datasets. This explains the I/O overhead of IBBS is the same as that of BBS. Similar to Figure 6, Figure 7 compares the algorithms in terms of CPU time (in secs). By and large, the CPU costs of both IBBS-WOS and BBS are similar. However, as shown in Figure 7, IBBS-WOS slightly outperforms BBS in the lower dimensionality. The CPU time of IBBS-OS increases fast as d increases, and it is clearly higher than other algorithms. The reason is that IBBS-OS introduces some CPU overhead for implementing the second pruning strategy. Additionally, it is expected that the performance of all algorithms degrades because the overlapping among the MBRs of R-tree increases and the number of skyline points grows.

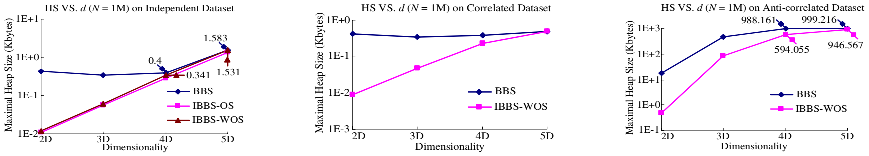


Fig. 5. Maximal Heap Size (Kbytes) VS. d ($N = 1\text{M}$)

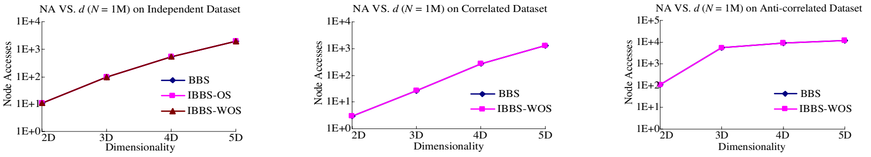


Fig. 6. Node Accesses VS. d (N = 1M)

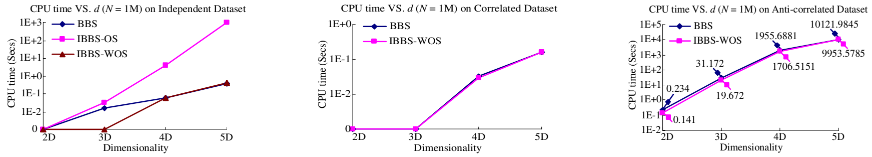


Fig. 7. CPU time VS. d (N = 1M)

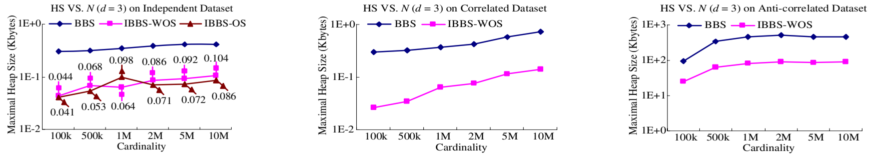


Fig. 8. Maximal Heap Size (Kbytes) VS. N (d = 3)

Next, we investigate the influence of cardinality N. Toward this, we deployed the 3-dimensional (the parameter $d = 3$ is the median value used in Figures 5 to 7) datasets whose cardinality range varies between 100k and 10M. Figures 8-9 show the MHS and CPU cost, respectively, versus N. Obviously, IBBS-WOS exceeds BBS in all cases. Specifically, the heap of IBBS-WOS is several orders of magnitude smaller than that of BBS. For CPU time, IBBS-WOS is also faster than BBS, especially in low dimensions. In addition, as the above experiments, the heap of IBBS-OS is similar to that of IBBS-WOS, but its CPU cost is greatly larger than other algorithms. Also note that, as shown in Figure 9(c), the difference increases with the cardinality, which is due to the positions of the skyline points and the order in which they are discovered.

Finally, we also inspect the *progressive* behavior of the algorithms for skyline query on 3-dimensional datasets. Figure 10 compares the size of the heap as a function of the number of skyline points (NSP for short) for datasets with N = 1M (for dependent and correlated datasets) or 100k (for anti-correlated dataset) and $d = 3$. Note that the NSP in the final skyline is 94, 26, and 13264, for independent, correlated, and anti-correlated datasets, respectively. From the diagrams, we see that IBBS-WOS clearly exhibits smaller heap size than BBS (over orders of magnitude) in all cases, since most of non-qualifying entries are pruned by IBBS-WOS. As expected, the heap size of both IBBS-OS and IBBS-WOS is highly adjacent. On the other hand, notice that the heaps reach their

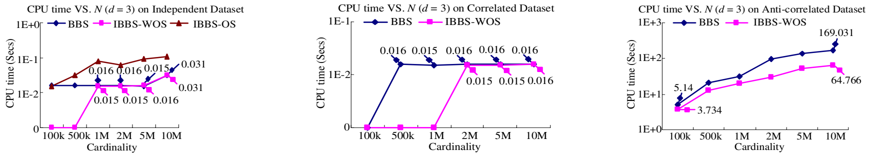


Fig. 9. CPU time VS. N $(d = 3)$

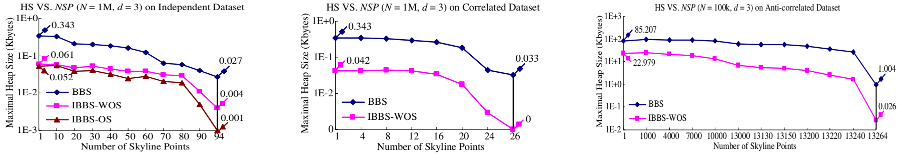


Fig. 10. Maximal Heap Size (Kbytes) VS. NSP $(N = 1\text{M or } 100\text{k}, d = 3)$

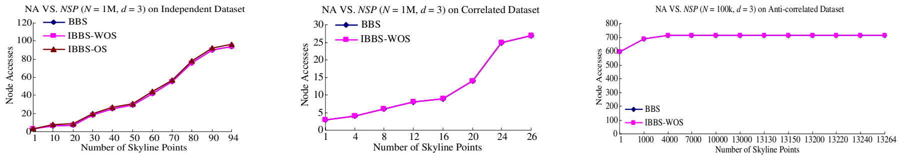


Fig. 11. Node Accesses VS. NSP $(N = 1\text{M or } 100\text{k}, d = 3)$

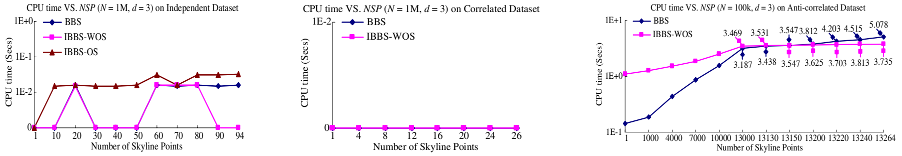


Fig. 12. CPU time VS. NSP $(N = 1\text{M or } 100\text{k}, d = 3)$

maximum size at the beginning of all algorithms, and stepwise decrease with the growth of NSP, which is also shown in Figure 10. The reason of this phenomenon is these algorithms insert respective all necessary entries visited in the heap (due to no any skyline point found) before they discover the first skyline point.

Figures 11 and 12 show all experimental results on the number of node accesses and CPU time, respectively, versus NSP under the same settings as Figure 10. Similar to Figure 6, all algorithms are I/O optimal, and their I/O costs grow as the skyline points returned increase. For CPU cost, both IBBS-WOS and BBS are similar in most cases, as well as they are faster than IBBS-OS. Additionally, in Figure 12(c), notice that BBS outperforms IBBS-WOS initially, which is caused mainly that IBBS-WOS need expend some time to remove non-qualifying entries at the beginning of it. However, the difference gradually decreases with the NSP, and then IBBS-WOS faster than BBS. This happens because the heap of BBS keeps some redundant entries (that can be removed in IBBS-WOS using our proposed pruning heuristics of this paper).

5 Conclusion and Future Work

Although BBS has some desirable features such as I/O optimal and flexibility, it requires large main-memory consumption. Motivated by this problem, an improved skyline query algorithm based on best-first nearest neighbor search, termed IBBS, is proposed in this paper. It enables several dominance checking based pruning strategies to eliminate non-qualifying entries, thus reducing significantly the memory space and speed up slightly the skyline computation. Extensive experiments with synthetic datasets confirm that the proposed algorithm is efficient and outperforms its alternative in both space overhead (i.e., heap size) and CPU cost for most cases, especially in low dimensions. In the future, we plan to study new algorithms for skyline queries relied on breadth-first (or depth-first) nearest neighbor retrieval paradigm. Another interesting topic is to explore parallel skyline query processing methods for various parallel environments (e.g., multi-disk or multi-processor setting).

Acknowledgement. This research was supported by the National High Technology Development 863 Program of China under Grant No. 2003AA4Z3010-03.

References

1. Balke, W.-T., Gntzer, U., Zheng, J.X.: Efficient Distributed Skylining for Web Information Systems. In: EDBT. (2004) 256-273
2. Beckmann, N., Kriegel, H.-P., Schneider, R., Seeger, B.: The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In: SIGMOD. (1990) 322-331
3. Borzsony, S., Kossmann, D., Stocker, K.: The Skyline Operator. In: ICDE. (2001) 421-430
4. Böhm, C., Kriegel, H.-P.: Determining the Convex Hull in Large Multidimensional Databases. In: DaWaK. (2001) 294-306 265–318
5. Chan, C.-Y., Eng, P.-K., Tan. K.-L.: Stratified Computation of Skylines with Partially-Ordered Domains. In: SIGMOD. (2005) 203-214
6. Chang, Y.-C., Chang, Y.-C., Bergman, L.D., Castelli, V., Li, C.-S., Lo, M.-L., Smith, J.: The Onion Technique: Indexing for Linear Optimization Queries. In: SIGMOD. (2000) 391-402
7. Chomicki, J., Godfrey, P., Gryz, J., Liang, D.: Skyline with Presorting. In: ICDE. (2003) 717-719
8. Fagin, R.: Fuzzy Queries in Multimedia Database Systems. In: PODS. (1998) 1-10
9. Godfrey, P., Shipley, R., Gryz, J.: Maximal Vector Computation in Large Data Sets. In: VLDB. (2005) 229-240
10. Hjaltason, G.R., Samet, H.: Distance Browsing in Spatial Databases. ACM TODS **24** (1999) 265-318

11. Hristidis, V., Koudas, N., Papakonstantinou, Y.: PREFER: A System for the Efficient Execution of Multi-parametric Ranked Queries. In: SIGMOD. (2001) 259-270
12. Kossmann, D., Ramsak, F., Rost, S.: Shooting Stars in the Sky: An Online Algorithm for Skyline Queries. In: VLDB. (2002) 275-286
13. Lin, X., Yuan, Y., Wang, W., Lu, H.: Stabbing the Sky: Efficient Skyline Computation over Sliding Windows. In: ICDE. (2005) 502-513
14. Natsev, A., Chang, Y.-C., Smith, J.R., Li., C.-S., Vitter. J.S.: Supporting Incremental Join Queries on Ranked Inputs. In: VLDB. (2001) 281-290
15. Papadias, D., Tao, Y., Greg, F., Seeger, B.: Progressive Skyline Computation in Database Systems. ACM TODS **30** (2005)41-82
16. Pei, J., Jin, W., Ester, M., Tao, Y.: Catching the Best Views of Skyline: A Semantic Approach Based on Decisive Subspaces. In: VLDB. (2005) 253-264
17. Preparata, F., Shamos, M. Computational Geometry: An Introduction. Springer-Verlag (1985)
18. Roussopoulos, N., Kelley, S., Vincent, F.: Nearest neighbor queries. In: SIGMOD. (1995) 71-79
19. Tan, K.-L., Eng, P.-K., Ooi, B.C.: Efficient Progressive Skyline Computation. In: VLDB. (2001) 301-310
20. Theodoridis, Y., Sellis, T.K: A Model for the Prediction of R-tree Performance. In: PODS. (1996) 161-171
21. Yuan, Y., Lin, X., Liu, Q., Wang, W., Yu, J.X., Zhang, Q.: Efficient Computation of the Skyline Cube. In: VLDB. (2005) 241-252

A Novel Clustering Method Based on Spatial Operations

Hui Wang

School of Computing and Mathematics, University of Ulster at Jordanstown
Newtownabbey, BT37 0QB, Northern Ireland, UK
H.Wang@ulster.ac.uk

Abstract. In this paper we present a novel clustering method that can deal with both numerical and categorical data with a novel clustering objective and without the need of a user specified parameter. Our approach is based on an extension of database relation – hyperrelations. A hyperrelation is a set of hypertuples, which are vectors of sets.

In this paper we show that hyperrelations can be exploited to develop a new method for clustering both numerical and categorical data. This method merges hypertuples pairwise in the direction of increasing the density of hypertuples. This process is fully automatic in the sense that no parameter is needed from users. Initial experiments with artificial and real-world data showed this novel approach is promising.

1 Introduction

The clustering of data is to organise data by abstracting the underlying structure of the data, either as a grouping of objects or as a hierarchy of groups. The representation can then be investigated to see if these data groups accord to preconceived ideas or to suggest new experiments (1). The objective of clustering is simply to find a convenient and valid organisation of the data. Clustering algorithms are geared toward finding structure in the data, organised around *clusters*. A cluster is comprised of a number of similar objects collected or grouped together.

In the context of knowledge discovery from databases, clustering is the process of discovering a set of categories to which objects should be assigned. Clustering algorithms are required to discover distinct categories using an unlabeled set of data. Objects in the dataset are then assigned (often as a by-product of the clustering process) to these categories.

Most of the existing clustering algorithms are either for numerical data only or for categorical data only. In the case of mixed data (that is, some attributes are numerical while others are categorical) the *numerical-only* clustering algorithms have to treat categorical attributes as numerical in some ways; while the *categorical-only* algorithms have to treat numerical attributes as categorical in some ways (2). Many existing clustering algorithms also needs some parameters from users. For example, the number of clusters (one of the most common parameters demanded from users), the neighbourhood radius and minimum number of points (3), and the number of sub-cells into which to partition a cell (4).

D. Bell and J. Hong (Eds.): BNCOD 2006, LNCS 4042, pp. 140–151, 2006.

It would then be desirable to have a clustering algorithm which can treat numerical and categorical data uniformly and which needs little input from users. In this paper we present a method for clustering having this trait. Our method is based on an extension of database relation.

The extension of a relation is underpinned by a mathematical structure called a *domain lattice* (5), which underlies any relational data scheme. A domain lattice is a set of all *hypertuples* in a problem domain, equipped with a partial ordering. A hypertuple is a vector of sets; in contrast, a database tuple is a vector of single values in its basic form. A *hypertuple* is then a generalisation of a database tuple. A *hyperrelation* is a set of hypertuples. The concept of relations in database theory and applications can be generalised to hyperrelations. Domain lattice has previously been exploited to address the data reduction problem in data mining (5).

In this paper we show that the hyperrelations and the domain lattice can be further exploited to develop a new method for clustering both numerical and categorical data uniformly. In Section 2 we introduce some notation and concepts for use in this paper, including hyperrelations and domain lattice. The central notion of our approach – density of hypertuples – is introduced in Section 3. Section 4 examines three fundamental issues about clustering from our density perspective. An efficient algorithm for clustering is presented and analysed in Section 5. Experimental results are reported in Section 6. Section 7 concludes the paper.

2 Definitions and Notation

Figure 1 illustrates the concepts of simple relations, hyperrelations, and domain lattice, which we define formally below.

2.1 Order and Lattices

A *partial order* on a set $\mathcal{L}$ is a binary relation $\leq$ which is reflexive, antisymmetric and transitive. A *semilattice* $\mathcal{L}$ is a nonempty partially ordered set such that for $x, y \in \mathcal{L}$ the least upper bound $x + y$ exists. Then for $A \subseteq \mathcal{L}$, its least upper bound exists and is denoted by $\text{lub}(A)$. The greatest element of $\mathcal{L}$, if it exists, is denoted by 1; if $\mathcal{L}$ is finite then 1 exists, and it is equal to $\sum_{a \in \mathcal{L}} a$.

The sub-lattice of $\mathcal{L}$ generated from $M \subseteq \mathcal{L}$, written by $[M]$, is $[M] = \{t \in \mathcal{L} : \exists X \subseteq M \text{ such that } t = \text{lub}(X)\}$. The greatest element in $[M]$ is $\text{lub}(M)$.

For $A, B \subseteq \mathcal{L}$, we say A *is covered by* B (or B *covers* A), written $A \preccurlyeq B$, if for each $a \in A$ there is some $b \in B$ such that $a \leq b$. We write $A \prec B$ if $A \preccurlyeq B$ and $B \not\preccurlyeq A$.

2.2 Domain Lattice

Let $\mathbf{D}$ be a relation with a schema $\Omega = \{x_1, \cdots, x_T\}$ and domains V_x of attributes $x \in \Omega$.

Let $\mathcal{L} \overset{\text{def}}{=} \prod_{x \in \Omega} 2^{V_x}$. Then $\mathcal{L}$ is a semilattice (in fact, it is a Boolean algebra, but we will not need this here) under the ordering

$$\forall t, s \in \mathcal{L}, t \leq s \Longleftrightarrow t(x) \subseteq s(x) \text{ for all } x \in \Omega. \tag{1}$$

with the least upper bound or the *sum* of $t, s \in \mathcal{L}$ given below

$$t + s \overset{\text{def}}{=} \langle t(x) \cup s(x) \rangle_{x \in \Omega} \tag{2}$$

If $t \leq s$ we say t is *below* s. $\mathcal{L}$ is called *domain lattice* for $\mathbf{D}$. The elements of $\mathcal{L}$ are called *hypertuples*; in particular, the elements t of $\mathcal{L}$ with $|t(x)| = 1$ for all $x \in \Omega$ are special hypertuples called *(simple) tuples*. A set of hypertuples is called a *hyperrelation*, and a set of simple tuples is called a *(simple) relation*. Simple relations are database relations in the traditional sense.

Note that $t(x)$ is the projection of tuple t onto attribute x. In practical terms, $t(x)$ can be treated as a set if x is a categorical attribute, and it can be treated as an interval if x is numerical.

In domain lattice $\mathcal{L}$, $\mathbf{D}$ is the set of simple tuples given in the dataset. There is a natural embedding of $\mathbf{D}$ into $\mathcal{L}$ by assigning

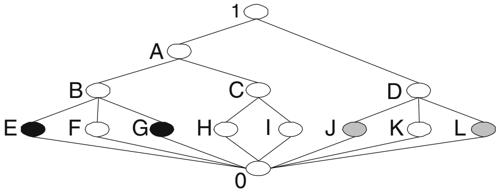
$$\Omega(a) \mapsto \langle \{x_1(a)\}, \{x_2(a)\}, \ldots, \{x_T(a)\} \rangle.$$

and we shall identify $\mathbf{D}$ with result of this embedding. Thus we have $\mathbf{D} \subseteq \mathcal{L}$.

In the sections below our discussion focuses mainly on a subset of the domain lattice (sublattice) $[\mathbf{D}]$ generated from the dataset $\mathbf{D}$.

Node	Size	Colour	Shape	Label
E	Large	Red	Triangle	+
G	Large	Blue	Circle	+
J	Small	Blue	Triangle	-
L	Small	Green	Rectangle	-

(a)



(b)

Node	Size	Colour	Shape	Label
B	Large	{Blue,Red}	{Circle,Triangle}	+
D	Small	{Blue,Green}	{Rectangle,Triangle}	-

(c)

Fig. 1. (a) A relation extension. (b) An abstract *domain lattice* implied in the relation. (c) A hyper relation.

3 Density of Hypertuples

In the sequel we shall use $\mathbf{D}$ as described above as a generic dataset, $\mathcal{L}$ as the domain lattice implied in $\mathbf{D}$, and $[\mathbf{D}]$ as the sublattice generated from $\mathbf{D} \subseteq \mathcal{L}$. The sum operation and partial ordering are as defined in the previous section.

Clustering is a partition $\mathcal{P}$ of $\mathbf{D}$ with classes $\{\mathbf{D}_1, \cdots, \mathbf{D}_K\}$. Each class $\mathbf{D}_i$ is a subset of data objects in $\mathbf{D}$ and is called a *cluster*. In traditional approaches to clustering a cluster is represented by the set of objects in the cluster, or by the center of gravity of the cluster (k-means) or by one of the objects of the cluster located near its center (k-medoid) (6). In our approach we represent a cluster by a hypertuple. For each class $\mathbf{D}_i$ we merge all simple tuples in $\mathbf{D}_i$ by the lattice sum operation in Eq. 2 resulting in a hypertuple $h_i = \text{lub}(\mathbf{D}_i)$. Then we get a hyperrelation $H = \{h_1, \cdots, h_K\}$. Therefore we can take clustering as a process of transforming a simple relation ($\mathbf{D}$) into a hyperrelation (H). Putting this formally, a clustering of $\mathbf{D}$ is a hyperrelation $H \subseteq [\mathbf{D}]$ and therefore $h \in H$ implies there is $A \subseteq \mathbf{D}$ such that $h = \text{lub}(A)$.

Clearly there are many possible partitions of the dataset. To choose one from among them we need a measure of hyperrelations. In our approach we use the measure of density, defined below.

Definition 3.1. *Let $h \in [\mathbf{D}]$ be a hypertuple, and $x \in \Omega$ be an attribute. The* magnitude *of $h(x)$ is defined as*

$$\text{mag}(h(x)) = \begin{cases} \max(h(x)) - \min(h(x)), \textit{ if } x \textit{ is numerical} \\ |h(x)|, \textit{ if } x \textit{ is categorical} \end{cases} \tag{3}$$

Note that $h(x)$ is the projection of h onto attribute x, $\min(h(x))$ is the minimal value in $h(x)$ while $\max(h(x))$ is the maximal value.

Definition 3.2. *The* volume *of h is defined as*

$$\text{vol}(h) = \prod_{x \in \Omega} \text{mag}(h(x)) \tag{4}$$

The coverage *of h is* $\text{cov}(h) \overset{\text{def}}{=} \{d \in \mathbf{D} : d \leq h\}$.

Definition 3.3. *The* density *of h is defined as*

$$\text{den}(h) = |\text{cov}(h)| / \text{vol}(h) \tag{5}$$

The density *of hyperrelation H,* $\text{den}(H)$*, is then the average density of the hypertuples in H.*

The above definition of density can not be directly applied to compare different hypertuples since different hypertuples may differ at different attributes, and different attributes may have different scales. Therefore we need to re-scale the attributes up to a same *uniform scale*. The re-scaling can be achieved as follows.

Table 1. A relation on the scheme $\{A_1, A_2\}$ where attribute A_1 is categorical and A_2 is numerical

	A_1	A_2
t_0	a	2
t_1	f	10
t_2	c	4
t_3	f	9
t_4	c	3
t_5	e	7
t_6	b	1
t_7	d	6

Let λ be the expected uniform scale. For an attribute $x \in \Omega$, the *re-scaling coefficient* is $s(x) \overset{\text{def}}{=} \lambda / \operatorname{mag}(V_x)$. Note that V_x is the domain of attribute x. Then the volume of a hypertuple h after re-scaling is $\operatorname{vol}(h) = \prod_{x \in \Omega} s(x) \times \operatorname{mag}(h(x))$. The density definition can be re-scaled similarly.

This re-scaled notion of density is fine for hypertuples. But there is a problem for simple tuples. Consider Table 1. Suppose the uniform scale is 2. Then $s(A_1) = 2/3$ and $s(A_2) = 1$. Following the above definition, the density for all simple tuples is 0 since the projection of each simple tuple to (numerical) attribute A_2 contains only one value. This is not desirable, the reason for which will be seen in the next section. Therefore we need a method to allocate density values to simple tuples in such a way that the values can be compared with the density values of hypertuples for the purpose of clustering. Our solution is through *quantization of attributes*. For an attribute $x \in \Omega$ the measurement of a unit after re-scaling is $|V_x|/\lambda = 1/s(x)$. For a tuple t, if $t(x)$ is less than the unit value $(1/s(x))$ it should be treated as one unit. If t is a simple tuple then $t(x)$ is treated as a unit for all $x \in \Omega$ and hence $\operatorname{vol}(t) = 1$. Since a simple tuple covers only itself, i.e., $\operatorname{cov}(t) = \{t\}$, we have $\operatorname{den}(t) = 1$. Consequently $\operatorname{den}(H) = 1$ if H is a *simple* relation. If t is a hypertuple, then $\operatorname{den}(t)$ may be greater or less than 1.

In the rest of this paper whenever we talk of density we refer to the re-scaled and quantized density.

The notion of density is also used in some well known clustering methods (7; 3; 4), but their uses of this notion are different from ours: they are defined for numerical attributes only and they are not re-scaled and quantized. Our definition of density applies to both numerical and categorical attributes and, since re-scaled and quantized, can be used to compare among hypertuples and among hyperrelations.

4 Merging Hypertuples to Increase Density

Having a notion of density as defined above we now present our clustering method. Our philosophy for clustering is *merging tuples to increase the density of hyperrelations*: for any set of tuples, if their sum has higher density then we are inclined to merge them and use their sum to replace this set of tuples. We discuss our method along three fundamental axes regarding any clustering methods: *the criteria for clustering, the determination of the number of clusters* and *the assignment of new tuples to clusters.*

4.1 Criteria for Clustering

An important notion in clustering is *neighbourhood* (or *similarity*). "Similar" objects should be clustered together. In the context of domain lattice, "similar" tuples should end up in same hypertuples. The meaning of neighbourhood varies in different approaches and contexts. For example, the Euclidean distance (or L_p metric [1] in general) and density (7; 8) for numerical data; and the *Jaccard coefficient* [2] (9; 1), the *links* [3] (10), the *co-occurence* in hypergraph (11; 2) [4], the *interestingness* (12) and the *share* (13) for categorical data .

Clustering is then to optimise one or more of these measures one way or another. In hierarchical clustering there is a basic operation — *merge*: two data objects can be merged if they are *neighbouring* or *close enough.* A prerequisite for this approach is the availability of a *proximity matrix* (1). In the case of numerical data this matrix is obtained by some distance measure, e.g., Euclidean distance; in the case of categorical data it is usually not available.

Our approach is hierarchical, and two tuples are deemed neighbours if the density of their sum (see Eq. 2) is higher than the density of the hyperrelation containing only the two tuples (i.e., the average density of the two tuples). More formally, let $\mathbf{D}$ be the dataset and $\mathcal{H} = \{H : \ H \text{ is a clustering of } \mathbf{D}\}$. Our objective is to find $H_0 \in \mathcal{H}$ such that $\text{den}(H_0) > \text{den}(\mathbf{D})$ and $\text{den}(H_0) = \max\{\text{den}(H) : H \in \mathcal{H}\}$. In other words our expected clustering should have the highest possible density. We call this H_0 the *optimal clustering* of $\mathbf{D}$. From the previous section we know that $\text{den}(\mathbf{D}) = 1$ and hence $\text{den}(H_0)$ should be much greater than or equal to 1.

The optimal clustering of the data in Table 1 is shown in Table 2. Readers can check for themselves that any other hyperrelations obtained by merging simple tuples in the dataset using the lattice sum operation has lower density. For example, merging $\{t_0, \cdots, t_3\}$ and $\{t_4, \cdots, t_7\}$ results in a hyperrelation in Table 3, which has lower density.

With such a criteria we can obtain a proximity matrix for any relational data, no matter it is numerical, categorical or mixed. Table 4 is a proximity matrix for the data in Table 1, where entry (i, j) is 1 if $\text{den}(t_i + t_j) \geq \text{den}(\{t_i, t_j\})$ and 0 otherwise.

[1] $L_p = (\sum_1^d |x_i - y_i|^p)^{1/p}$, $1 \leq p \leq \infty$ and d is the dimensionality of the data points.

[2] The Jaccard coefficient for similarity between two sets S_1 and S_2 is $|S_1 \cap S_2|/|S_1 \cup S_2|$.

[3] The number of links between a pair of data points is the number of common neighbours for the points.

[4] In this approach each tuple in the database is viewed as a set of data objects, and the entire collection of tuples is treated as a hypergraph. The *co-occurence* between two tuples is the number of common elements and is noted as the weight of the edge between the two hyper notes.

Table 2. The optimal clustering of the relation in Table 1 obtained by our method. The uniform scale used is 4, so the re-scaling coefficients are $s(A_1) = 2/3$ and $s(A_2) = 4/9$. The density of this hyperrelation is then 1.313. Note that the density values are re-scaled, and that the density for the original dataset is 1.

	A_1	A_2	Coverage cov()	Density den()
t'_0	$\{a, b, c\}$	$\{1, 2, 3, 4\}$	$\{t_0, t_2, t_4, t_6\}$	1.500
t'_1	$\{d, e, f\}$	$\{6, 7, 9, 10\}$	$\{t_1, t_3, t_5, t_7\}$	1.125

Table 3. An arbitrary hyperrelation obtained by merging simple tuples in Table 1. The uniform scale used is the same as in Table 2, so are the re-scaling coefficients. The density of this hyperrelation is 0.5625.

	A_1	A_2	Coverage cov()	Density den()
t''_0	$\{a, c, f\}$	$\{2, 4, 9, 10\}$	$\{t_0, t_1, t_2, t_3\}$	0.5625
t''_1	$\{b, c, d, e\}$	$\{1, 3, 6, 7\}$	$\{t_4, t_5, t_6, t_7\}$	0.5625

Table 4. A proximity matrix for the relation in Table 1

	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7
t_0	1	0	1	0	1	0	1	0
t_1	0	1	0	1	0	1	0	0
t_2	1	0	1	0	1	1	1	1
t_3	0	1	0	1	0	1	0	1
t_4	1	0	1	0	1	0	1	1
t_5	0	1	1	1	0	1	0	1
t_6	1	0	1	0	1	0	1	0
t_7	0	0	1	1	1	1	0	1

This approach has a major advantage: *numerical and categorical attributes can be treated uniformly*. Simple tuples, either numerical or categorical or a mixture of the two, can thus be *uniformly* measured for their neighbourhood.

4.2 Determination of the Number of Clusters

Some clustering algorithms require the number of clusters be given by users. Our approach can determine the number of clusters automatically; it can also be tuned to find a required number of clusters as stated by a user.

As discussed in the previous section our criteria for clustering is to maximise the density of hyperrelations. Naturally the *optimal* number of clusters should be the number of hypertuples in the optimal clustering. Whether or not we can find the optimal clustering depends on the algorithm used.

For the data in Table 1, its optimal clustering is shown in Table 2. Therefore the optimal number of clusters for this dataset is 3.

In some data mining exercises, however, we have a preconceived (given) number of clusters and we wish the clustering algorithm to find the given number of clusters for us. Examples are: in business systems, to allocate stock to a given number of warehouses, or to allocate tuples to a given number of disk volumes in physical database design (14; 15). Some clustering algorithms require the number of clusters be given in this way.

Suppose we wish to find N clusters. With the availability of a proximity matrix we can take an agglomerative hierarchical approach. Assume we have a hierarchy of clusterings (hyperrelations) of the dataset, $Q_0, Q_1, \cdots, Q_q$, where $Q_0 = \mathbf{D}$, $Q_q = \text{lub}(\mathbf{D})$ and $|Q_i| = |Q_{i-1}| - 1$. Clearly we can select a hyperrelation Q_k in the hierarchy such that $|Q_k| = N$.

4.3 Assignment of New Data Tuples to Clusters

Suppose we have already clustered a dataset, resulting in a clustering $H = \{h_1, \cdots, h_K\}$ where each h_i is a hypertuple. When new data arrives we may need to assign the new data objects to the existing clusters to get a new clustering[5]; or we may want to assign new data objects to clusters for data mining purposes like categorisation, classification or association. In traditional approaches to clustering this is usually done via calculating distances (or proximities) between a new data object and the clusters using some metric and assigning the new data object to whichever cluster is closest to the new data object (1). In our approach we have available proximity matrices so we can do it in a similar way, but in a more general context (since our method works for both numerical and categorical data). Specifically, we sum the new data object with each cluster and see if, and to what extent, the sum increases density. If none of the sums increases density, then it is likely the new data object belongs to a new cluster; otherwise the new data object is assigned to one cluster the sum of which with the new data object has the greatest increase of density.

More formally, let t be a new data object – a simple tuple. We have two main steps for the assignment procedure:

– If $\text{den}(h_i + t) < \text{den}(\{h_i, t\})$ for all $i = 1, \cdots, K$, then t is taken as a new cluster on its own.
– Otherwise assign t to cluster h_{i_0} such that $(\text{den}(h_{i_0} + t) - \text{den}(\{h_{i_0}, t\}))/\text{den}(\{h_{i_0}, t\})$ is highest.

For an example, consider Table 2. If we are given a new tuple $t = \langle b, 8\rangle$, then we have $\text{den}(t'_0 + t) = 0.804$ and $\text{den}(t'_1 + t) = 1.055$. However $\text{den}(\{t'_0, t\}) = 1.25$ and $\text{den}(\{t'_1, t\}) = 1.063$. This indicates that the new tuple should stand as a new cluster. If the new tuple is $t = \langle b, 5\rangle$, then $\text{den}(t'_0 + t) = 1.406$ and $\text{den}(t'_1 + t) = 0.844$, and $\text{den}(\{t'_0, t\})$ and $\text{den}(\{t'_1, t\})$ remain the same. This indicates that we should assign this new tuple to t'_0.

5 An Efficient Algorithm for Clustering

Based on the above discussions we designed an efficient clustering algorithm, LM/Clus. The following is an outline of the algorithm.

[5] This is in fact incremental clustering (8).

```
– Input: $\mathbf{D}$ as defined above.
– Initialisation: $Q_0 = \mathbf{D}$; $flag = 1$; $i = 0$;
– WHILE ($flag = 1$)
    1. $flag = 0$; $H = Q_i$;
    2. WHILE (there are $x, y \in Q_i$ that haven't been examined)
         Let $w = x + y$;
         IF (den($w$) $>$ den($\{x, y\}$)
             THEN $Q_{i+1} = Q_i \cup \{w\} \setminus \{x, y\}$; $i = i + 1$; $flag = 1$;
                   //Replace $x$ and $y$ by their sum.
– Output: $H$.
```

Execution starts with $Q_0 = \mathbf{D}$. Then a pair of elements $x, y \in Q_0$ such that $x + y$ increases density is merged, resulting in Q_1. The next loop starts from Q_1 and results in Q_2. This process continues until Q_m where no pair of elements can be merged in this way. Thus we get a sequence $Q_0, Q_1, \cdots, Q_m$, where Q_0 is the original dataset and Q_m is the quasi-optimal clustering. Clearly $Q_0 \preccurlyeq Q_1 \preccurlyeq \cdots \preccurlyeq Q_m$, and $|Q_i| = |Q_{i-1}| - 1$. Therefore this is an agglomerative hierarchical clustering algorithm (1).

This algorithm has a worst case complexity of $O(n \log n)$ where $n = |\mathbf{D}|$. In our implementation of the algorithm we take advantage of the operations in our Lattice Machine (5) so the average computational complexity is close to linear (see below). The algorithm is implemented as part of our Lattice Machine based KDD suite, called Dr .

6 Experimental Results

In this section we present experimental results showing the effectiveness and efficiency of our clustering algorithm LM/Clus. We used two types of data: artificial data and real world data. These datasets are a good mix of numerical and categorical data. Artificial datasets were generated from known clusters with added noise, and they are mainly used to show the effectiveness of the algorithm (i.e., can the known clusters be discovered?) as well as efficiency. The data generator we used is also available in our Dr system. Real world datasets are public and are frequently used in KDD literature. They are used in our experiment mainly to show the efficiency of the algorithm since the underlying clusters are not known.

6.1 Artificial Datasets

We used two seeds to generate our artificial datasets. The seeds are described in Table 5. For each seed we generated four datasets of varying sizes with 2% random noise added, and with $100, 1000, 5000, 10000$ tuples respectively. The time used to cluster these data is shown in Table 6. From this table we can see that the algorithm is close to linear in the number of tuples. The underlying cluster structures were fully recovered. Readers are invited to evaluate our system which is available online (for the Web address see the footnote on page 148).

Table 5. Two seeds used to generate artificial data

	$Attribute1$	$Attribute2$	$Attribute3$
Cluster 1	$[0, 4]$	$[100, 130]$	$\{a, b, c\}$
Cluster 2	$[6, 10]$	$[160, 199]$	$\{c, d, e\}$

(a) Seed one: gd1. The first two attributes are numerical and the third is categorical.

	$Attribute1$	$Attribute2$	$Attribute3$	$Attribute4$
Cluster 1	$[0, 4]$	$[100, 130]$	$[1000, 1300]$	$\{a, b, c\}$
Cluster 2	$[6, 10]$	$[160, 199]$	$[1400, 1650]$	$\{d, e, f\}$
Cluster 3	$[3, 7]$	$[140, 150]$	$[1750, 1999]$	$\{c, d\}$

(b) Seed two: gd2. The first three attributes are numerical and the fourth is categorical.

Table 6. Time in seconds used to cluster the artificial data

	gd1.100x2	gd1.1000x20	gd1.5000x100	gd1.10000x200
Time	0.44	3.84	22.24	93.65

	gd2.100x2	gd2.1000x20	gd2.5000x100	gd2.10000x200
Time	1.15	10.27	61.24	229.98

Table 7. Some general information about the real world data and the time in seconds used to cluster the data

Dataset	#Attributes	#Size	#Numeric Attribute	#Categorical Attribute	Clustering Time
German	20	1000	6	14	820.14
Heart	13	270	9	4	56.68
Iris	4	150	4	0	2.53

6.2 Real World Datasets

We chose three public datasets to show the efficiency of the algorithm for clustering: `German Credit`, `Heart Disease` and `Iris`, all available from UCI Machine Learning Data Repository. Some general information about the datasets and clustering time are shown in Table 7.

7 Conclusion

In this paper we present a novel method of automatically clustering both numerical and categorical data or mixed data in a uniform way. The first major contribution of this paper is the provision of a uniform measure of density for both numerical and categorical data. After re-scaling and quantization this measure can be used to compare among any (simple or hyper) tuples and among any (simple or hyper) relations to see which is denser. Since the density measure is *local*, its calculation is very efficient. Based on this measure our clustering method is simply to transform simple relations (original data) to hyperrelation guided by the density measure. Data tuples are merged with the aim of increasing the density of hyperrelations. The optimal clustering is the hyperrelation with highest possible density, and the number of hypertuples in this hyperrelation is the *optimal* number of clusters.

Another major contribution of this paper is the provision of an efficient algorithm for clustering, LM/Clus. This algorithm is (agglomerative) hierarchical and it takes advantage of our (local) measure of density. It examines pairs of tuples and merges those which increase the density of the relation. This process continues until the density of the relation cannot be increased. Experiments with both artificial data and real world data showed that this algorithm is very efficient and is, in average, close to linear in the number of data tuples. Experiments with the artificial datasets showed that this algorithm is also effective as it recovers completely the underlying cluster structures used to generate the datasets.

Bibliography

[1] Jain, A.K., Dubes, R.C.: Algorithms for Clustering Data. Prentice Hall, Englewood Cliffs, New Jersey (1988)

[2] Gibson, D., Kleinberg, J., Raghavan, P.: Clustering categorical data: An approach based on dynamical systems. In: Proc. 24th International Conference on Very Large Databases, New York (1998)

[3] Ester, M., Kriegel, H.P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining, AAAI Press (1996) 226–231

[4] Wang, W., Yang, J., Muntz, R.: STING: A statistical information grid approach to spatial data mining. In: Proc. 23rd Int. Conf. on Very Large Databases, Morgan Kaufmann (1997) 186–195

[5] Wang, H., Düntsch, I., Bell, D.: Data reduction based on hyper relations. In: Proceedings of KDD98, New York. (1998) 349–353

[6] Kaufman, L., Rousseeuw, P.J.: Finding Groups in Data: An Introduction to Cluster Analysis. John Wiley & Sons (1990)

[7] Schikuta, E.: Grid clustering: an efficient hierarchical clustering method for very large data sets. In: Proc. 13th Int. Conf. on Pattern Recognition. Volume 2., IEEE Computer Society Press (1996) 101–105

[8] Ester, M., Kriegel, H.P., Sander, J., Wimmer, M., Xu, X.: Incremental clustering for mining in a data warehousing environment. In: Proc. 24th International Conference on Very Large Databases. (1998)

[9] Duda, R.O., Hart, P.E.: Pattern classification and scene analysis. John Wiley & Sons (1973)

[10] Guha, S., Rastogi, R., Shim, K.: ROCK: A robust clustering algorithm for categorical attributes. Technical Report 208, Bell Laboratories (1998)

[11] Han, E.H., Karypis, G., Kumar, V., Mobasher, B.: Clustering based on association rule hypergraphs. In: 1997 SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery. (1997)

[12] Gray, B., Orlowska, M.E.: Clustering categorical attributes into interesting association rules. In: Proc. PAKDD98. (1998)

[13] Hilderman, R.J., Carter, C.L., Hamilton, H.J., Cercone, N.: Mining market basket data using share measures and characterized itemsets. In: Proc. PAKDD98. (1998)

[14] Bell, D.A., McErlean, F., Stewart, P., Arbuckle, W.: Clustering related tuples in databases. Computer Journal **31**(3) (1988) 253–257

[15] Stewart, P., Bell, D.A., McErlean, F.: Some aspects of a physical database design and reorganisation tool. Journal of Data and Knowledge Engineering (1989) 303–322

A FP-Tree-Based Method for Inverse Frequent Set Mining

Yuhong Guo, Yuhai Tong, Shiwei Tang, and Dongqing Yang

Department of Computer Science
Peking University, Beijing 100871, China
{yhguo, yhtong, tsw, dqyang}@pku.edu.cn

Abstract. Recently, the inverse frequent set mining problem has received more attention because of its important applications in different privacy preserving data sharing contexts. Several studies were reported to probe the NP-complete problem of inverse frequent set mining. However, it is still an open problem that whether there are reasonably efficient search strategies to find a compatible data set in practice. In this paper, we propose a FP-tree-based method for the inverse problem. Compared with previous "generation-and-test" methods, our method is a zero trace back algorithm, which saves huge computational costs. Furthermore, our algorithm provides a good heuristic search strategy to rapidly find a FP-tree, leading to rapidly finding the compatible databases. More importantly, our method can find a *set* of compatible databases instead of finding only *one* compatible database in previous methods.

1 Introduction

As the frequent itemsets can be considered as a kind of summary of the original data set, recently the inverse frequent set mining problem inferring the original data set from given frequent itemsets with the supports has received more attention because of its potential threat to privacy [1], its use in synthetic benchmark data set generation [2], and its potential application in sensitive association rule hiding in database [3].

Inverse frequent set (or "itemset") mining can be described as follows: "Given a collection of frequent itemsets and their supports, find a transactional data set (or "database") such that the new dataset precisely agrees with the supports of the given frequent itemset collection while the supports of other itemsets would be less than the pre-determined threshold" [4]. This inverse data mining problem is related to the questions of how well privacy is preserved in the frequent itemsets and how well the frequent itemsets characterize the original data set. It has very practical applications in different privacy preserving data sharing contexts from privacy preserving data mining (PPDM) to knowledge hiding in database (KHD), as the problem roots in people's increasing attention to information protection either to individual private data preserving or to business confidential knowledge hiding.

Mielikainen first proposed this inverse mining problem in [1]. He showed finding a dataset compatible with a given collection of frequent itemsets or deciding whether there is a dataset compatible with a given collection of frequent sets is NP-complete.

D. Bell and J. Hong (Eds.): BNCOD 2006, LNCS 4042, pp. 152 – 163, 2006.

1.1 Related Work

Towards the NP-complete problem, several methods were proposed. The authors of [2, 3] designed a linear program based algorithm for *approximate* inverse frequent itemset mining, aiming to construct a transaction database that *approximately* satisfies the given frequent itemsets constraints. As for the *exact* inverse frequent itemset mining, Calder in his paper [5] gave a naive "generate-and-test" method to "guess" and build a database horizontally(transaction by transaction) from given frequent itemsets. On the contrary, the authors of [4] proposed a vertical database generation algorithm to "guess" a database vertically---column by column when looking the transaction database as a two-dimensional matrix. Unfortunately, under the generation-test framework, neither of the algorithms works well in terms of effectiveness and efficiency as they belong to simple enumerative search approaches essentially, which blindly try all possible value assignments, even those devoid of solutions. This means, once the "test" processes fail, the algorithms must rollback some costly "generate-and-test" operations, leading to huge computational cost.

Thus, a feasible and sufficient solution to the *exact* inverse frequent itemset mining is still expected. Obviously, if the given frequent sets collection comes from a real database, at least this original database must be one which exactly agrees with the given even though it is computationally hard to "render" it. The questions and challenges are: Is it unique? Can we find an efficient algorithm without trace back to find one? Can we find a good heuristic search strategy to reach one quickly?

This paper describes our effort towards finding an efficient method to find a set of databases that *exactly* agree with the given frequent itemsets and their supports discovered from a real database. Compared with previous "generation-and-test" methods, our proposed FP-tree-based method is a zero trace back algorithm, which saves huge computational costs. Furthermore, our algorithm provides a good heuristic search strategy to rapidly find a FP-tree, leading to rapidly finding the compatible databases. More importantly, our method can find a *set* of compatible databases instead of finding only *one* compatible database in previous methods.

1.2 Paper Layout

In section 2 we define the inverse frequent set mining problem that we focus on. In section 3 we review the FP-tree structure, and in section 4 we present our proposed algorithm. We analyze correctness and efficiency of our algorithm and discuss the number of databases generated in section 5. Section 6 summarizes our study.

2 Problem Description

Let $I = \{I_1, I_2, ..., I_m\}$ be a set of items, and a transaction database $D = \{T_1, T_2, ..., T_n\}$ where $T_i (i \in [1..n])$ is a transaction which contains a set of items in I. The support of an itemset $A \subseteq I$ in a transaction database D over I, denoted *support(A)*, is defined as the number of transactions containing A in D. A is a frequent itemset if A's support is no less than a predefined minimum support threshold "σ". If A is frequent and there exists no superset of A such that every transaction containing A also contains the superset, we say that A is a frequent *closed* itemset.

The well-known frequent itemset mining problem aims to find all frequent itemsets from a transaction database. And the objective of frequent closed itemset mining is to find all frequent *closed* itemsets. Conversely, inverse frequent itemset mining is to find the databases that satisfy the given frequent itemsets and their supports. If the given frequent itemsets are frequent *closed* itemsets, we call the inverse mining process *inverse frequent closed itemset mining*. Furthermore, if the frequent closed itemsets and their supports are discovered from a real database, we call the inverse mining process *inverse real frequent closed itemset mining*. Here, "real" only represents "existent", which means the real database can be an artificial data set.

In this paper, we focus on the *inverse real frequent closed itemset mining* defined as: Given a set of items $I = \{I_1, I_2, ..., I_m\}$, minimum support threshold "σ", and a set of all frequent *closed* itemsets $F = \{f_1, f_2, ...f_n\}$ with fix supports $S = \{support(f_1), support(f_2), ..., support(f_n)\}$ discovered from a real database D, find a set of databases *DBs* in which each database *D'* satisfies the following constraints:

(1) *D'* is over the same set of items *I*;

(2) From *D'*, we can discover exactly same set of frequent closed itemsets"*F*"with the same support"*S*"under the same minimum support threshold"σ".

3 Frequent Pattern Tree

Frequent pattern tree (or FP-tree in short) proposed by Jiawei Han and efficiently used in frequent set mining, is an extended prefix-tree structure for storing compressed, crucial information about frequent patterns.

TID	Items	Ordered Frequent Item Sets
1	A，B，E	B，A，E
2	B，D	B，D
3	B，C	B，C
4	A，B，D	B，A，D
5	A，C	A，C
6	B，C	B，C
7	A，C	A，C
8	A，B，C，E	B，A，C，E
9	A，B，C	B，A，C

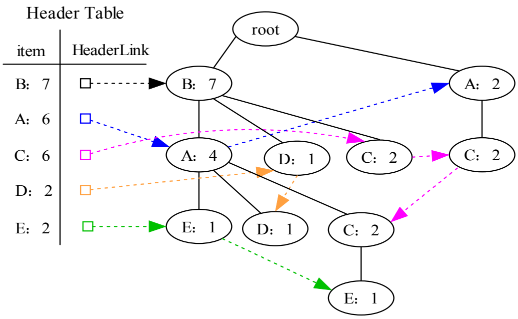


Fig. 1. A transaction database and its frequent pattern tree (FP-tree)

Fig. 1 gives an example of a transaction database and its FP-tree, which will be used in the next section. The database includes nine transactions comprising the items in the set *{A, B, C, D, E}*, which are shown in the mid column of the table. The FP-tree is constructed by two scans of the database. First scan of the database derives a *list* of frequent items *〈B:7, A:6, C:6, D:2, E:2〉* (the number after ":" indicates the support), in which items ordered in support descending order. The frequent items in each transaction are listed in this ordering in the rightmost column of the table. The

FP-tree forms in the second scan of the nine transactions in the rightmost column, with each transaction ‘climbing’ the FP-tree one by one. An item header table is built to facilitate tree traversal. Details of FP-tree construction process can be found in [6].

4 Proposed Method

4.1 Basic Idea

Our method to generate a database *D* from given frequent itemsets uses FP-tree as a transition “bridge” and can be seen as the reverse process of the FP-tree-based frequent itemsets mining method proposed in [6]. The idea comes from the fact that FP-tree is a highly compact structure which stores the complete information of a transaction database *D* in relevance to frequent itemsets mining. Thus we can look upon FP-tree as a medium production between an original database and its corresponding frequent itemsets. Intuitively, FP-tree reduces the gap between a database and its frequent itemsets, which makes the transformation from given frequent itemsets to database more smoothly, more naturally and more easily.

Our method works as follows. First, we try to “guess” a FP-tree that satisfies all the frequent itemsets and their supports. We call such a FP-tree a compatible FP-tree. Second, generate a corresponding database *TempD* directly from the compatible FP-tree by outspreading all the paths of the tree. Third, generate expected databases based on *TempD* by scattering some infrequent itemsets into the transactions in *TempD*, with the “new” itemsets brought below the given minimum support threshold.

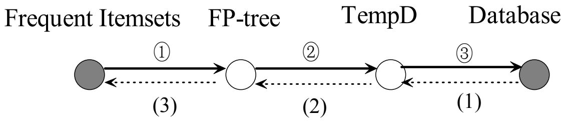


Fig. 2. Basic process of FP-tree-based inverse frequent set mining vs. frequent set mining

Fig. 2 shows the basic process of our proposed method for inverse frequent set mining, which is marked as ①, ② and ③. It corresponds the three steps described above. The process of the FP-tree-based frequent itemsets mining is also shown in the Fig. 2, which is composed of the three steps: (1), (2) and (3). Detailed information about the FP-tree-based frequent itemset mining process can be found in [6].

4.2 Algorithm

The sketch of our proposed inverse frequent closed set mining algorithm, which is composed of three procedures, is given as follows.

```
Gen_DB(F, I,σ)
Begin
    DBs← ∅, $I_{\overline{F}}$ ← I - { all items in F };
    FP←Gen_FPtree(F,σ);
    TempD←Outspread(FP);
```

```
4.  DBs←DBs ∪ {TempD};
5.  NewTempD←insert "some" items in $\overline{IF}$ into "some" transactions in TempD, be
    sure that each item in $\overline{IF}$ can only be inserted less than"$\sigma$" transactions;
6.  DBs←DBs ∪ {NewTempD};
7.  Goto 5 until no different NewTempD generated;
8.  Return DBs;
End

Gen_FPtree(F, σ)
Begin
1.  Create the root of an FP-tree, FP, and label it as "null";
2.  F' ←Sort(F);
3.  While (F'  !=∅) DO
      a)  Select the first itemset f1:s1 , where f1 is the itemset and s1 is its support;
      b)  Let f1 be [p|P], where p is the first element and P is the remaining list of
          f1;
      c)  Insert_tree([p|P]:s1, FP);
      d)  Update F':
            For all f∈F' and f⊆f1,
              i.  support(f)←support(f) - s1;
              ii. if (support(f)=0) then F'←F'-{f};
      e)  F' ←Sort(F');
4.  Return FP;
End

Outspread(FP)
Begin
1.  TempD ← ∅
2.  if (FP=null) then return TempD
    else
      (a) search the tree by in-depth order to find a leaf node ln:s, where ln is its item
          and s is its count;
      (b) t←all items in the  path from the root FP to the leaf node ln;
      (c) For i=1 to s, TempD ← TempD  ∪ { t};//TempD can include duplicates for t
      (d) Update FP:
          For each node n in the path from the root FP to the leaf node ln,
            i.  n.count←n.count - s;
            ii. if (n.count=0) then delete n from the tree FP;
      (e) Outspread(FP);
End
```

The input of the algorithm is a set of items *I*, minimum support threshold "σ"and frequent closed itemsets collection *F* with the support *S*. The output is a set of databases *DBs*. Each element of the *DBs* is a transaction database that agrees with *F*.

In the main procedure "*Gen_DB()*", we use *FP* to represent the tree obtained from the frequent closed itemsets collection *F* by calling the sub-procedure "*Gen_FPtree()*". We use *TempD* to represent the result of outputting *FP* by calling the sub-procedure "*Outspread()*". Notice that *FP* and *TempD* include only the items occurring in *F*. $\overline{IF}$ represents infrequent items included in *I* but not occurring in *F*. *NewTempD* is used to record the new generated database based on *TempD*.

In the sub-procedure "*Gen_FPtree()*", the function *Sort(F)* sort the itemsets in *F* by the number of items and support in descending order. Moreover, the items in each itemset are sorted in 1-itemset's support descending order. The function *Insert_tree([p|P]:s1, FP)*is performed as follows: If *FP* has a child *N* such that *N.item-name=p.item-name*, then increment *N*'s count by *s1*; else create a new node *N*, and let its count be *s1*, its parent link be linked to *FP*. If *P* is nonempty, call *Insert_tree(P,N)* recursively.

We use *F'* to store the sorted frequent itemsets so far by the number of items and support in descending order. First, an itemset *f1:s1* in the forefront of *F'* "climbs" the FP-tree. Then, the supports of all frequent itemsets in *F'* that are subset of *f1* subtract *s1* and *F'* is updated. The two steps repeat until all supports of the frequent itemsets in *F'* are equal to "0", and *F'* is equal to $\varnothing$. The sort routine of *F'* insures that each time the longest itemset with highest support is submitted first to "climb" the FP-tree. That is, the longer itemsets with higher supports are always satisfied prior to the shorter itemsets with lower supports during the FP-tree generation. This heuristic idea leads that once the longest itemset with highest support in the forefront of *F'* "climbs" the tree, the remaining tasks decrease sharply because more supports will probably be subtracted and more itemsets will probably be wiped off in the updating process of *F'*.

In the sub-procedure "*Outspread()*", the itemsets on each path of the FP-tree "come down" from the tree and form transactions of *TempD* one by one until the tree is equal to null. The result of this sub-procedure *TempD* can be seen as the status of an "Ordered Frequent Items" transaction database in [6] deleting infrequent items in each transaction (like the database in the rightmost column of the table in Fig. 1).

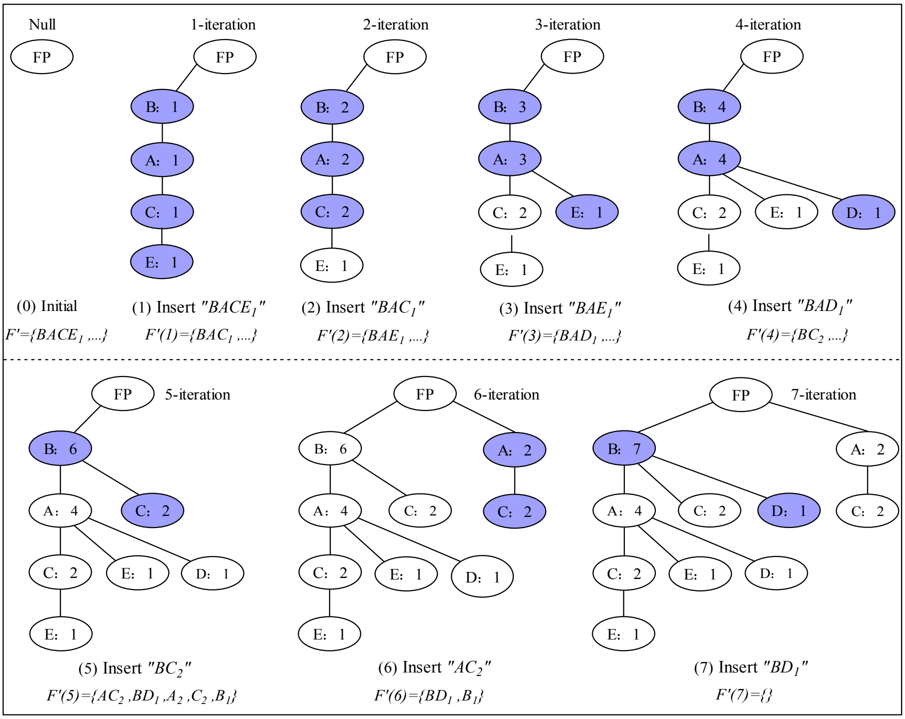
Lines 4-7 of the procedure "*Gen_DB()*" generate a set of databases *DBs* by scattering some infrequent items (elements of $\overline{IF}$) into *TempD,* just be sure that each infrequent item can only be scattered less than "σ"(minimum support threshold) transactions of *TempD*. Concretely speaking, suppose the number of infrequent items equals to n ($|\overline{IF}|=n$), $\overline{IF} = \{item_1, ..., item_i, ..., item_n\}$, $|TempD|=m$, then $DBs=\{TempD\} \cup NewTempDSet_1 \cup \dots \cup NewTempDSet_i \cup \dots \cup NewTempDSet_n$, where $NewTempDSet_1$ is a set of all the new generated databases by scattering $item_1$ into *TempD*, and $NewTempDSet_i$ is a set of all the new generated databases by scattering $item_i$ into all the previous generated databases in $\{TempD\} \cup \dots \cup NewTempDSet_{i-1}$.

4.3 Example

Let's illustrate our algorithm with an example: Given $I=\{A, B, C, D, E\}$, minimum support threshold "$\sigma =1$", and frequent closed itemsets collection $F=\{EABC_1, EAB_2, DBA_1, DB_2, C_6, BC_4, BCA_2, AC_4, A_6, AB_4, B_7\}$ (the subscripts represent supports) discovered from the transaction database in Fig. 1 of section 3.

① Generate FP-tree

We first sort itemsets in F by the number of items and support in descending order. We get $F'=\{EABC_1, BCA_2, EAB_2, DBA_1, AB_4, BC_4, AC_4, DB_2, B_7, A_6, C_6\}$. Then the items in each itemset are sorted in 1-itemset's support descending order. Now $F'=\{BACE_1, BAC_2, BAE_2, BAD_1, BA_4, BC_4, AC_4, BD_2, B_7, A_6, C_6\}$.



(0) Initial $F'=\{BACE_1, ...\}$

(1) Insert "$BACE_1$" $F'(1)=\{BAC_1, ...\}$

(2) Insert "BAC_1" $F'(2)=\{BAE_1, ...\}$

(3) Insert "BAE_1" $F'(3)=\{BAD_1, ...\}$

(4) Insert "BAD_1" $F'(4)=\{BC_2, ...\}$

(5) Insert "BC_2" $F'(5)=\{AC_2, BD_1, A_2, C_2, B_1\}$

(6) Insert "AC_2" $F'(6)=\{BD_1, B_1\}$

(7) Insert "BD_1" $F'(7)=\{\}$

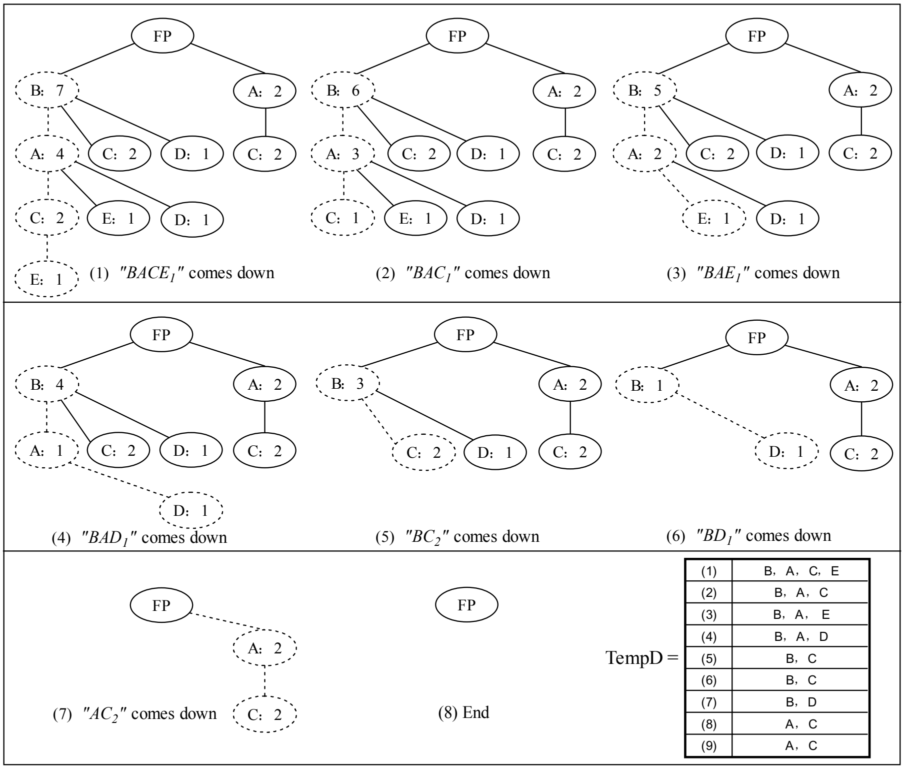
Fig. 3. The change of the FP-tree during the FP-tree generation

During the first iteration, the first itemset "$BACE_1$" is selected and inserted to the tree rooted as "*FP*". We get the tree like Fig.3-(1). Then *F'* is updated by subtracting "1" from the supports of all subsets of "*BACE*" occurring in *F'*. Itemsets with support equal to "0" are wiped off during updating. Now we get $F'(1)= F'\text{-}\{BACE, BAC, BAE, BA, BC, AC, B, A, C\}_1=\{BAC_1, BAE_1, BAD_1, BA_3, BC_3, AC_3, BD_2, B_6, A_5, C_5\}$. The itemsets in *F'(1)* has already been sorted by the number of items and support in descending order, so the result of function *Sort(F')* performing on *F'(1)* is still *F'(1)*. We can see that after the first iteration, *F'* becomes *F'(1)* whose itemsets and related supports decrease much, which means the remaining itemsets and their related supports needed to satisfy in the followed iteration decrease much. This dramatically reduces the cost effects on the FP-tree generation.

During the second iteration, "BAC_1" is inserted to the tree and we get $F'(2)=\{BAE_1, BAD_1, BA_2, BC_2, AC_2, BD_2, B_5, A_4, C_4\}$. Then "$BAE_1$", "$BAD_1$", "$BC_2$", "$AC_2$" and "$BD_1$" are inserted to the tree (or we say "climb" the tree) one after the other and we get the following update sequence of F' after each iteration: 〖 $F'(3)=\{BAD_1, BC_2, AC_2, BD_2, BA_1, B_4, C_4, A_3\}$; $F'(4)=\{BC_2, AC_2, BD_1, C_4, B_3, A_2\}$; $F'(5)=\{AC_2, BD_1, A_2, C_2, B_1\}$; $F'(6)=\{BD_1, B_1\}$; $F'(7)=\varnothing$〗. The "Gen_FPtree" process terminates after seven iterations when $F'=\varnothing$. Fig. 3 shows the change of the FP-tree during the whole process of the FP-tree generation.

② Generate temporary transaction database TempD by outspreading FP-tree

Fig. 4 shows the change of the FP-tree during the whole process of the *TempD* generation. First, "$BACE_1$" which is the leftmost branch of the original FP-tree "comes down" from the tree. At the same time, all the items in this branch form into the first transaction of *TempD* : *TempD(1)=(B, A, C, E)* . After deleting the "$BACE_1$" branch from the tree, the "*FP*" tree changes into the form shown as Fig.4-(2). By calling the *Outspread(FP)* recursively, we perform the similar operations on the remaining six "*FP*" trees (Fig.4-(2) to Fig.4-(7)) in turn. "BAC_1", "BAE_1", "BAD_1",



(1)	B, A, C, E
(2)	B, A, C
(3)	B, A, E
(4)	B, A, D
(5)	B, C
(6)	B, C
(7)	B, D
(8)	A, C
(9)	A, C

Fig. 4. The change of the FP-tree during the *TempD* generation

"BC_2", "BD_1" and "AC_2" comes down from the tree one by one and we get the remaining eight transactions of *TempD*: *TempD(2)=(B, A, C)*; *TempD(3)=(B, A, E)*; *TempD(4)=(B, A, D)*; *TempD(5)=(B, C)*; *TempD(6)=(B, C)*; *TempD(7)=(B, D)*; *TempD(8)=(A, C)*; *TempD(9)=(A, C)*. Notice that when "BC_2" and "AC_2" come down, our algorithm generates two same transactions respectively. The whole algorithm ends when *FP=null* (Fig.4-(8)) and we get a temporary transaction database *TempD* shown as the table in Fig. 4.

③ Generate a set of databases DBs by scattering infrequent items into TempD

In last process ②, we have generated a transaction database *TempD* from the FP-tree generated in ①. In fact, *TempD*, which involves only frequent items, keeps all information about frequent itemsets and constitutes skeleton of the compatible databases we are to find. By scattering infrequent items into *TempD*, we can get more than one database, exactly a set of databases satisfying the given constraints.

In our example, the set of infrequent items $I\overline{F}=\varnothing$, as all the items in the set of $I=\{A, B, C, D, E\}$ occur in the set of frequent itemsets $F=\{EABC_1, EAB_2, DBA_1, DB_2, C_6, BC_4, BCA_2, AC_4, A_6, AB_4, B_7\}$. So according to lines 4-7 of the procedure "*Gen_DB()*" in our whole algorithm, no *NewTempD* is generated and the eventual *DBs={TempD}*. This means we find only one transaction database *TempD* satisfying *F* in this example. Interestingly, *TempD* (see Fig.4) happens to be the transaction database shown in Fig.1 of section 3, without regard to the order of transactions and the items order in each transaction. Another interesting thing is the FP-tree generated in this example by our inverse mining algorithm (see Fig.3-(7)) happens to be the tree shown in Fig.1 of section 3 generated from the transaction database in Fig. 1 by the FP-growth algorithm in [6], without regard to the order of children of each node. What do the interesting results indicate? At least we can get the following three valuable hints from the interesting results.

First, it validates the correctness of the result, as the input frequent sets *F* is discovered from the database in Fig. 1. So from *TempD* we must be able to discover exactly the same *F*, and *TempD* really satisfies *F*. Second, it indicates the feasibility and effectiveness of our method, as we really find a database satisfying *F* only based on the inputs (I, F, σ) and our algorithm, without knowing any other things about the original database. Third, it induces us to think: It is what factors that lead to the interesting results? How many compatible databases can be found by the proposed algorithm in usual cases? These questions will be probed in the next section 5.

5 Analysis

In this section, we analyze the correctness, efficiency of our algorithm. Then we focus on discussing the number of compatible databases that our algorithm can generate.

(1) Correctness

The correctness of our algorithm can be ensured by the three steps during our algorithm performing. The first step "Generate FP-tree" insures the generated FP-tree is compatible with the given frequent sets constraints, because all the given frequent sets "climb" the FP-tree and FP-tree can store the complete information in relevance to frequent itemsets mining. The second step "Generate *TempD* by outspreading FP-tree" ensures the generated *TempD* is also compatible with the given frequent sets constraints, because all frequent sets "come down" from the FP-tree and from *TempD* we can construct a same FP-tree. The third step "Generate a set of databases *DBs* by scattering infrequent items into *TempD*" guarantees all the frequent sets and their supports related information keeps down exactly, no changes happen on any frequent sets' supports, and no new frequent sets are brought. So that all the databases in *DBs* preserve the complete and exact information of the given frequent sets constraints and are correct compatible databases we are to find.

(2) Effectiveness

The effectiveness of our algorithm lies in the two facts. One fact is our algorithm is a zero trace back algorithm with no rollback operations, since during constructing a compatible FP-tree process each itemset "climbs" the FP-tree following the prescribed order. And the remaining two transformations "from FP-tree to *TempD*" and "from *TempD* to the set of compatible databases" are natural and direct, with no rollback too. The other fact is, with the longest itemset with highest support "climbing" the FP-tree first during each iteration, our algorithm provides a good heuristic search strategy to rapidly find a compatible FP-tree.

Suppose the number of the given frequent closed sets in collection *F* is *k* and the number of transactions generated in *TempD* is *m*, i.e. $|F|=k$, $|TempD|=m$. Then the FP-tree construction can be accomplished in $O(klogk+(k-1)log(k-1)+...+1)$ time, in which *klogk* represents the time to sort the *k* frequent closed sets in *F* in the frequent sets length and support descending order. The number of elements in *F* decrease one each time the first frequent set climbs the FP-tree. The time consumed in *TempD* generation is determined by the number of branches in the FP-tree and approximates to $O(m)$. Hence the first two processes in our algorithm can both be accomplished in polynomial time. The most time-consuming process in our algorithm may be the third process to generate a set of compatible databases *DBs*. This is because our algorithm may generate an exponential number of compatible databases (see the number of compatible databases analysis in part (3) of this section). But it does not show our algorithm is inefficient. On the contrary, it shows the effectiveness of our algorithm because we can generate so many compatible databases. In fact, in our algorithm the generation of new databases is very easy and quick just scattering a new infrequent item into all the previous databases in prescribed principle. It may be time-consuming only because there are so many answers to be output.

All in all, with no trace back and with the good search strategy, our algorithm can work very effectively generating lots of compatible databases.

(3) The number of compatible databases

Fig. 5 illustrates mapping relation among compatible database space, compatible FP-tree space and given frequent closed sets collection, which helps to probe the number of compatible databases that our algorithm can output. In Fig. 5, *FCS* is a frequent closed sets collection discovered from one of the databases in DBs_i undergoing $TempD_i$ and $FP\text{-}tree_i$ by FP-growth method in [6]. DBs_j is the output set of compatible databases generated from *FCS* undergoing $FP\text{-}tree_j$ and $TempD_j$ by our algorithm. All

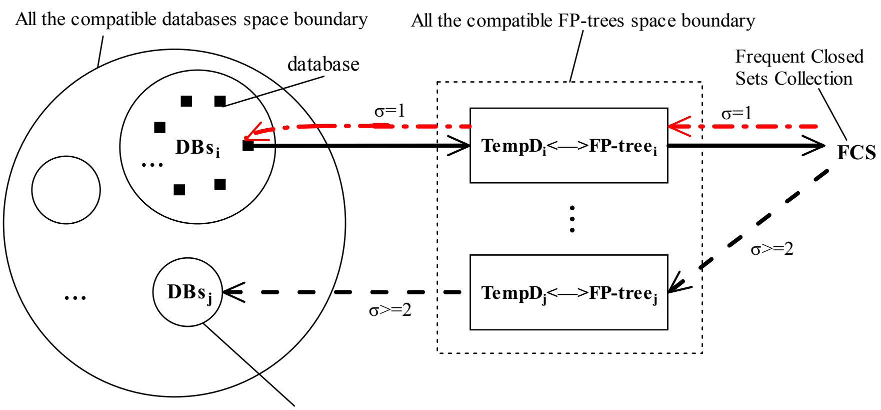

All the compatible databases space boundary
All the compatible FP-trees space boundary
database
Frequent Closed Sets Collection
σ=1
σ=1
DBs_i
…
TempD_i<—>FP-tree_i
FCS
σ>=2
…
DBs_j
TempD_j<—>FP-tree_j
σ>=2

Fig. 5. Mapping relation among database space, FP-tree space and FCS

the databases in DBs_i map into the same $FP\text{-}tree_i$ and have the same number of transactions as $TempD_i$, so do the DBs_j, $FP\text{-}tree_j$ and $TempD_j$. The figure shows that what our algorithm outputs is only a small part (a class having the same number of transactions and corresponding to the same FP-tree) of the whole compatible database space. Then how many databases our algorithm can output?

The number of compatible databases that our algorithm can output (indicated as $|DBs|$) is related to the three parameters: (1) the number of transactions in *TempD*, i.e.$|TempD|$; (2) the number of infrequent items in $\overline{IF}$, i.e. $|\overline{IF}|$; (3) the minimum support threshold"σ". Suppose $|TempD| = m$, $|\overline{IF}| = n$ and $f(n)$ represents the number of generated databases after the n-th infrequent item in $\overline{IF}$ has been scattered into all of the previous generated databases fully, we have the recurrence equation: $f(n+1)= f(n)+ f(n)(C_m^1 + C_m^2 + \cdots + C_m^{\sigma-1})(n\geq 0,\ 2\leq\sigma\leq m,\ m\geq 1,\sigma\in N)$ (C_m^1 means the number of selecting one transaction from m transactions of a generated database that has not included the $(n+1)$-th infrequent item); and $f(0)=1$ which means when there is no infrequent items in $\overline{IF}$, there is only one compatible database (*TempD*) our algorithm finds. By solving the recurrence equation, we get $|DBs| = f(n) = (1+C_m^1+C_m^2+\cdots+C_m^{\sigma-1})^n$ $(n\geq 0,\ \ 2\leq\sigma\leq m,\ \ m\geq 1,\sigma \in N)$. When$\sigma=1$, $|DBs|=1$; and when$\sigma<<m$, $|DBs|$ is in direct proportion to $(\sigma C_m^{\sigma})^n$. In practice we can limit the number of compatible databases to be generated when $|DBs|$ is astronomical or when we are trying to find fixed number of compatible databases.

Notice that our other examples show usually(when$\sigma\geq 2$) $FP\text{-}tree_j$ is different from $FP\text{-}tree_i$, DBs_i and DBs_j are disjoint, and the database set that our algorithm outputs does not include the original database. However, when$\sigma=1$, $FP\text{-}tree_j$ happens to be the same with $FP\text{-}tree_i$, leading $TempD_j$ is just the same with $TempD_i$, and the only compatible database we are to find is just the original database because there exists no infrequent items can be scattered under "$\sigma=1$". This explains the two interesting facts in the example in section 4: The *TempD* in Fig.4 happens to be the original database in Fig.1; and the FP-tree in Fig.3-(7) happens to be the same FP-tree in Fig.1. The two dashed lines with arrowheads in Fig.5 illustrate the different execution paths of our algorithm in usual case ($\sigma >= 2$) and special case ($\sigma = 1$).

6 Conclusions

We have presented a feasible and efficient algorithm for the NP-complete problem of inverse frequent set mining. The algorithm can effectively generate a set of databases that *exactly* agree with the given frequent closed itemsets and their supports discovered from a real database. Compared with previous "generation-and-test" methods, our method is a zero trace back algorithm, without rollback operations during the databases' generation, which saves huge computational costs. Furthermore, our algorithm provides a good heuristic search strategy to rapidly find a FP-tree satisfying the given frequent sets constraints, leading to rapidly finding the compatible databases. More importantly, our algorithm can find a *set* of compatible databases (usually a lot of databases) instead of finding only *one* compatible database in previous methods. We also probe the number of databases found by our algorithm.

This study is just our first step towards solving this inverse mining problem. More work will be done in the near future, such as refinement of the algorithm, and empirical experiments on real databases. However, for this NP-complete inverse mining problem, our study has shown that there do exist reasonably efficient search strategies and solutions to find *some* (at least one, not all, but usually a lot of) data sets compatible with a given data set. This study can be used to deal with privacy preserving data sharing, in which data owners will have a choice in releasing different versions of the original data for different sharing (benchmark, mining, etc.).

References

1. Mielikainen, T.: On Inverse Frequent Set Mining. In: IEEE ICDM Workshop on Privacy Preserving Data Mining, IEEE Computer Society (2003) 18–23
2. Wu, X., Wu, Y., Wang, Y., Li, Y.: Privacy-Aware Market Basket Data Set Generation: A Feasible Approach for Inverse Frequent Set Mining. In: Proc. 5th SIAM International Conference on Data Mining (2005)
3. Wang, Y., Wu, X.: Approximate Inverse Frequent Itemset Mining: Privacy, Complexity, and Approximation. In: Proc. 5th International Conference on Data Mining (2005) 482–489
4. Chen, X., Orlowska, M.: A Further Study on Inverse Frequent Set Mining. In: Proc. 1st International Conference on Advanced Data Mining and Applications (ADMA), Lecture Notes in Computer Science, Vol. 3584. Springer-Verlag (2005) 753–760
5. Calders, T.: Computational Complexity of Itemset Frequency Satisfiability. In: Proc. 23rd ACM PODS 04, ACM Press (2004) 143–154
6. Han, J., Pei, J., Yin, Y.: Mining Frequent Patterns without Candidate Generation. In: Proc. of the ACM SIGMOD International Conference on Management of Database (2000) 1–12

SC-Tree: An Efficient Structure for High-Dimensional Data Indexing

Ben Wang and John Q. Gan

Department of Computer Science, University of Essex,
Colchester CO4 3SQ, UK
{bwangm, jqgan}@essex.ac.uk

Abstract. In content-based information retrieval (CBIR) of multimedia data, high-dimensional data indexing and query is a challenging problem due to the inherent high dimensionality of multimedia data. As a data-based method, metric distance based high-dimensional data indexing has recently emerged as an attractive method because of its ability of making use of the properties of metric spaces to improve the efficiency and effectiveness of data indexing. M-tree is one of the most efficient indexing structures for searching data from metric space, and it is a paged, balanced, and dynamic tree that organizes data objects in an arbitrary metric space with fixed sizes for all its nodes. However, inherent disadvantages are veiled in the M-tree and its variants, which prevent them from further improvement of their indexing and query efficiency. To avoid these disadvantages, this paper proposes a sorted clue tree (SC-tree), which essentially modifies the nodes, entries, indexing algorithm, and query algorithm of the M-tree but reserves its advantages. Experimental results and complexity analyses have shown that the SC-tree is much more efficient than the M-tree with respect to the query time and indexing time without sacrificing its query accuracy.

1 Introduction

Efficient access is essential for content-based information retrieval of large multimedia databases because multimedia data are usually characterized by high-dimensional features which bring about the curse of dimensionality problem in similarity searching operation.

There are two general categories of partitioning methods for data indexing: space-based partitioning and data-based partitioning. Space-based partitioning [12] [14][15][16][17] is also called grid-based partitioning, which partitions each dimension of the space into intervals and thus the whole space into grids. Although this is a simple partitioning method, the number of grids increases exponentially with the space dimension, resulting in the curse of dimensionality problem. Data-based partitioning [3][4][7] can also be called prototype-based partitioning, clustering-based partitioning, or distance-based partitioning. The number of partitions in data-based partitioning depends on the data distribution, e.g., the number of clusters, which is not directly related to the space dimension. Therefore, data-based partitioning does not

D. Bell and J. Hong (Eds.): BNCOD 2006, LNCS 4042, pp. 164 – 176, 2006.

have the curse of dimensionality problem and has found wide applications in high-dimensional data indexing.

As a data-based method, metric distance based high-dimensional data indexing has recently emerged as an attractive method because it is able to make use of the properties of metric spaces to improve the efficiency and effectiveness of data indexing [3][4][19]. Typical metric distance based indexing structures include vantage point tree (VP-tree) [18], multiple vantage points tree (MVP-tree) [5], geometric near-neighbour access tree (GNAT) [6], and paged metric tree (M-tree) series [2][7]. VP-tree partitions a data set according to distances that the objects have with respect to a vantage point, and then utilizes the triangle inequality to filter data objects to reduce the similarity search cost. However, due to its small fan-out, VP-tree structure is very deep, thus a search operation is time-consuming. MVP-tree employs multiple vantage points, and exploits pre-computed distances to reduce the number of distance computations during query process, but it is static and cannot be incrementally updated. GNAT captures the geometry of a data set by hierarchically breaking it down into regions. Long preprocessing time is the main disadvantage of GNAT. M-tree [7] is a paged, balanced, and dynamic tree that organizes data objects in an arbitrary metric space with fixed sizes for all its nodes. Since metric spaces strictly cover vector spaces, M-tree has a far more general applicability than multi-dimensional access methods, such as R-tree [13] and its variants [10]. For instance, a set of strings can be compared and organized in the M-tree according to edit distance which is defined as the minimal number of character changes needed to transform one string into another. In recent years, four important improvements have been made to the original M-tree: complex similarity search, approximate search, cost models, and user-defined distances. Complex similarity search handles several features, such as color, shape, or texture [8]. Approximate searching introduces PAC-NN (probably approximately correct nearest neighbor) queries, where error bound and accurate ratio can be tuned during a query period to trade query accuracy for query time [9]. Cost models concern the distance distribution of objects and predict both I/O and CPU query costs [11]. The user-defined distance approach develops a QIC-M-tree (QIC stands for query, index, comparison distances), which involves several distinct metrics at the same time [10]. However, three inherent disadvantages are veiled in the M-tree and its variants, which prevent the M-tree and its invariants from further improvement of their indexing and query efficiency. To avoid these disadvantages, this paper proposes a sorted clue tree (SC-tree), which essentially modifies the nodes, entries, indexing algorithm, and query algorithm of the M-tree but reserves its advantages.

In this paper, the M-Tree indexing structure is briefly introduced in section 2. The SC-tree is proposed in section 3. Experimental results and analyses are given in section 4. And conclusion is given in section 5.

2 M-Tree

M-tree is an efficient indexing structure for searching data from metric space [7]. A generic metric space is a pair, M = (U, d), where U is a domain of feature vectors and *d* is a distance function with the following postulates: symmetry, positivity, and triangle inequality:

$$d(x, y) = d(y, x). \textit{ (Symmetry)} \tag{1}$$

$$d(x, y) \geq 0 \textit{ and } d(x, y) = 0 \textit{ iff } x = y. \textit{ (Positivity)} \tag{2}$$

$$d(x, y) + d(y, z) \geq d(x, z). \textit{ (Triangle inequality)} \tag{3}$$

For the sake of self-containment, this section briefly describes the indexing structure and indexing and query algorithms of the M-tree.

2.1 Indexing Structure

M-tree indexing structure is constructed by hierarchical nodes. Each node consists of a fixed number of entries. There are two types of nodes: internal nodes and leaf nodes, corresponding to two types of entries. An internal entry, stored in internal nodes, contains a routing object, covering radius, a pointer to its sub-tree (a node at the next level), and the distance between the routing object and its parent. The routing object is defined as the representative centroid of objects in the sub-tree, and the covering radius is the farthest distance between any objects in the sub-tree and the routing object. However, in a leaf entry, an object identifier, its feature vector, and the distance between the object and its parent are recorded.

The formal definitions of the M-tree node, leaf entry, and internal entry are as follows: An M-tree node has a fixed number of entries, defined as $entries_i$, $i < numOfEntries$. A leaf entry has the format of [O_i, $oid(O_i)$, $d(O_i, P(O_i))$], where O_i is defined as the feature vectors of the routing object, $oid(O_i)$ as the object identifier, and $d(O_i, P(O_i))$ as the distance between O_i and its parent object $P(O_i)$. An internal entry has the format of [O_r, $r(N_r)$, $ptr(N_r)$, $d(O_r,P(O_r))$], where O_r is defined as the routing object, $r(N_r)$ as the covering radius of sub-tree N_r, and $ptr(N_r)$ as the pointer to N_r. For each O_i' in the sub-tree rooted at N_r, it has the property $d(O_r, O_i') \leq r(N_r)$.

2.2 Indexing Algorithm

The indexing algorithm of the M-tree inserts data objects into its nodes one by one. The insert algorithm recursively locates the most suitable internal or leaf node to accommodate a new data object. The strategy to find the most suitable node is to minimize the enlargement of the covering radius of the entries at each level. If a node is full of entries, the split algorithm will be called to deal with the overflow situation. Regardless of the specific split policy, the semantics of covering radius has to be preserved after each splitting operation.

In general, the indexing algorithm of the M-tree follows a bottom-up approach. Initially, an empty root node is generated. The first leaf entry is generated by selecting an object from the data set and inserting it into the root node. A leaf entry is inserted into a node if the node is not full. Otherwise, the split algorithm partitions the node into two sub-trees and a new node is generated at the same time. From these sub-trees, two routing objects are chosen as the new internal entries, whose pointers point to the sub-trees respectively. These two internal entries are then inserted into the new node. At this moment, the first M-tree with one root node and two sub-trees is formed. After that, the second leaf entry is generated by selecting another object form the data set and inserting it into the root node. The covering radius of the inserted entry in the root node should be updated. If the current node (the root node in this case) has sub-trees, the entry is recursively inserted into one of the sub-trees until a leaf node is reached. If the leaf node is full, the split algorithm has to be called. Otherwise, the entry is inserted into the leaf node. Following the above procedure, all the objects in the data set are inserted into the M-tree indexing structure.

2.3 *k*-NN Query Algorithm

The *k*-NN query algorithm retrieves *k* most similar objects with respect to a given query object *Q*. A priority queue PR and an array *NN* with *k* elements are utilized in the algorithm. The *PR* is a queue of pointers to active sub-trees where qualified objects can be found. A lower bound that records the distance between any object in the sub-trees and the query object *Q* is also kept in the *PR*, and the node with the minimal lower bound will be chosen. Since the pruning criterion of *k*-NN query algorithm is dynamic, the search radius is the distance between *Q* and its current *k*-th nearest neighbor. The order of the accessing nodes is crucial for high query performance. The query algorithm starts from the root node. It firstly locates active sub-trees of the root node and their lower bounds, and inserts them into the PR. After that, the query algorithm chooses one sub-tree from the *PR,* stores the node identifiers and the distances from the query *Q* in the array *NN*, and returns a *k*-NN value, d_k., which is used later as the search radius to remove sub-trees in the *PR* whose lower bounds exceed d_k. At the end of execution, the *k*-NN query results are stored in the array $NN[i] = [oid(O_i), d(O_i,Q)]$, $0 \leq i < k$, where $oid(O_i)$ is the object identifier of the *i*-th nearest neighbor of *Q* and $d(O_i,Q)$ represents the distance of the *i*-th nearest neighbor from *Q* [10].

2.4 Advantages and Disadvantages

M-tree has the following major innovative properties [9]. It is a balanced and incremental updating indexing structure that is able to index data sets from generic metric spaces. It is also dynamic and scalable. The *k*-NN query can be performed on the M-tree, with query results ranked in terms of the distances with respect to a given query object. It is suitable for indexing high-dimensional data.

However, M-tree has three inherent disadvantages that largely limit its indexing and query efficiency. Firstly, the entries in a node are stored randomly. As a result, the split algorithm has to find two farthest objects by comparing every pair of objects in the entry, which is obviously not efficient for the splitting operation. Secondly, in both the insert and split algorithms, locating the parent of the current node is needed frequently, but the searching has to inefficiently travel from the root node to all sub-trees until the current node is located [9]. Thirdly, for *k*-NN search algorithm, the chosen node is added into the priority queue *PR* without sorting the position of sub-trees according to the lower bounds between the query object and sub-trees. As a result, it influences the order of accessing nodes. The first two disadvantages will largely decrease the indexing efficiency, and the second and the third disadvantages will add much unnecessary query time. In order to improve the indexing and query efficiency for the M-tree, the SC-tree is proposed in the next section.

3 SC-Tree

SC-tree proposed in this paper is a high-dimensional data indexing structure that sorts entries in nodes, maintains a pointer to its parent for each node, and supports indexing and querying data from metric space. The entries in the SC-tree are sorted by the distance between routing objects and their parents. The pointer from current entry to its parent is called a “clue”. Details about the indexing structure, indexing algorithm, and query algorithm of the SC-tree are described in the following subsections.

3.1 Indexing Structure

The indexing structure of the SC-Tree includes two parts: nodes and entries. The entries in a node are sorted according to distances between routing objects and their parent objects, represented as *distFromParent*. There are a fixed number of entries in an internal node or leaf node, which are inserted into the node in ascending order of *distFromParent*. More formally, the entry and node structure are defined as follows:

An entry has five attributes: the feature vector of an routing object, O_n, the pointer to the root of the sub-tree, *sub-tree*, the object identifier of the entry, *oid*, the covering radius of its sub-tree, *coverRadius,* and the distance between the routing object and its parent, *distFromParent*. If the entry is internal, set *oid*=1. If the entry is a leaf one, set *sub-tree=Nil* and *coverRadius=0.*

A node has five attributes: the number of total entries in the node, *totalEntries*, the entries in a node, $entries_i$ ($i < totalEntries$), the number of non-empty entries in the node, *currentEntries*, the pointer to the parent node, *parentNode*, and the index of the entry in *parentNode, entryIndex,* which points to the current node. To locate the parent entry of current object is to simply return *parentNode[entryIndex].* The *parentNode[entryIndex].routObjectFeature* is the feature vector of the current object.

3.2 Indexing Algorithm

The indexing algorithm specifies how objects are inserted and how to deal with node overflow when a node has already been full before inserting a new object. In this section, an insert algorithm and a split algorithm are described in detail, with < change i > denoting the major differences between the SC-tree and the M-tree.

```
Insert Algorithm: Insert(treeNode, entry(O_n))
Input parameters: treeNode, entry(O_n)
Return: updated treeNode with the inserted entry(O_n)
S1. Get all the entries in treeNode.
S2. If treeNode is not a leaf node
  S2.1. Select those entries whose covering radiuses will not increase if entry(O_n)
        is inserted into them.
```

```
    S2.2. If the selected entries are not empty, select an entry, denoted as chosenEntry, whose routing object $O_r$ is the closest to the routing object $O_n$ of entry($O_n$).
    S2.3. Else select the chosenEntry with the minimal distance (d($O_r$, $O_n$) – coverRadius).
    S2.4. Get the sub-tree of chosenEntry, recursively call Insert(sub-tree, entry($O_n$)).
S3. Else the treeNode is a leaf node.
    S3.1. If the treeNode is not full, insert entry($O_n$) in the treeNode in ascending order of distFromParent and increase currentEntries by 1. <change 1>
    S3.2. Else Split (treeNode, entry($O_n$)).
S4. Return the updated treeNode.
```

Split Algorithm: Split(treeNode, entry(O_n))

```
Input parameters: treeNode, entry($O_n$)
Return: splitTreeNode
S1. Set combinedEntries = {entries of treeNode $\cup$ entry($O_n$)} and sort combinedEntries by distFromParent.
    S2. Get the parent node of treeNode by its parentNode pointer. <change 2>
S3. If treeNode is the root node (its parentNode is empty)
    S3.1. Set entry1 = the first entry of treeNode. <change 3>
    S3.2. Set entry2 = the last entry of treeNode. <change 4>
S4. Else if treeNode is not the root node
    S4.1. Set entry1 = parentNode[entryIndex]. <change 5>
    S4.2. Set entry2 = the last entry of combinedEntries. <change 6>
    S4.3. Set routObject1 = the routing object of entry1.
    S4.4. Set routObject2 = the routing object of entry2.
S5. Divide combinedEntries into two tree nodes, treeNode1 and treeNode2, based on the distances from the objects in combinedEntries to routObject1 and routObject2.
S6. If treeNode is the root node
    S6.1. Allocate a newRootNode.
    S6.2 Store entry1 and entry2 in newRootNode.
    S6.3 Record the parentNode and entryIndex for treeNode1 and treeNode2. <change 7>
    S6.4. Set splitTreeNode = newRootNode.
S7. Else if treeNode is not the root node
    S7.1. Replace parentNode[entryIndex] = entry1. <change 8>
    S7.1. If parentNode is full
        S7.1.1. Split(parentNode, entry2).
    S7.2. Else if parentNode is not full
        S7.2.1. Store entry2 in parentNode.
S8. Set splitTreeNode = parentNode.
S9. Return splitTreeNode.
```

In the construction of an M-tree, the split algorithm is frequently called in the insert algorithm, hence its efficiency will largely influence the efficiency of the insert algorithm. There are two disadvantages in the split algorithm of the M-tree. The first is that the distance between each object in an entry and its parent object has to be calculated in order to choose two routing objects from the split entry. The second is that the split algorithm has to travel from the root node to all its sub-trees until the current node is reached in order to find the parent node of current node. To overcome these two disadvantages, there are several noticeable modifications in the SC-tree, compared with the M-tree. In step S3.1 of the insert algorithm, the entry is inserted into the tree node in ascending order of *distFromParent*. This modification makes it possible to implement steps S3.1, S3.2, and S4.2 in the split algorithm. Due to the modifications, the SC-tree simply selects the first object and the last object from the entry as the routing objects for the split sub-trees because they are sorted by the distance between any object and their parent object. Furthermore, the modifications also speed up step S5 of the split algorithm because it is almost done for distributing objects to the first tree node, *treeNode1*, in which the objects are already sorted. Another modification is in step S6.3 of the split algorithm, in which the parent node of the current node is recorded .This modification, which is based on the indexing structure of SC-Tree, makes steps S2, S4.1, and S7.1 in the split algorithm much more efficient. To get the parent object of current node, *parentNode* and *entryIndex* attributes of current node can be directly returned.

3.3 *k*-NN Query Algorithm

The *k*-NN query algorithm of the SC-tree implements its search logic, which is described as follows:

```
k-NN Query Algorithm: k-NN(startNode, query, k)
Parameters: startNode, query , k
Return: an array NN storing k-NN query results
S1. If startNode is a rootNode
  S1.1. Initialize an array NN.
  S1.2. Choose active sub-trees based on their lower bounds, and insert them into the
      priority queue PR.
      Note: Different from the M-tree, the SC-tree inserts active sub-trees in
      ascending order of their lower bounds, which are defined by

          $d_{min} = \max\{\text{distFromRoutObjectToQuery - coverRadius}\}$      (4)

      where distFromRoutObjectToQuery represents the distance between the
      routing object and the query object. <change 9>
  S1.3. If the PR is not empty, select the first sub-tree, denoted as chosenNode, for
      which the $d_{min}$ is minimal. Set $d_k = d_{min}$ when $d_{min} < d_k$. <change 10>
    S1.3.1. Select the parentNode of chosenNode by its pointer to parent.
        <change 11>
    S1.3.2. Calculate the distance distFromParentToQuery between the routing
        object of parentNode[entryIndex] and the query object.
    S1.3.3. If the entry in chosenNode satisfies the following condition:

        $|\text{distFromParentToQuery - distFromParent}| \leq (d_k + \text{coverRadius})$      (5)

        where distFromParentToQuery is the distance between the parent object
        to the query object, and distFromParent is the distance between the
        routing object and the parent object.
      S1.3.3.1. Calculate distFromRoutObjectToQuery.
      S1.3.3.2. If distFromRoutObjectToQuery < $d_k$
```

```
        S1.3.3.2.1. Perform an ordered insertion of distFromRoutObjectToQuery
                    into NN, and get back the new k-NN distance $d_k$.
        S1.3.3.2.2. Remove entries in the PR if their lower bounds $d_{min}$ exceed $d_k$.
                      Firstly, the position in the PR where the first sub-tree with its
                    lower bound exceeds $d_k$ is found by binary search. Secondly, all
                    the entries after that position in PR will be removed. <change
                    12>
    S1.4. Else the PR is empty
      S1.4.1 Return the array NN.
S2. If startNode is not a rootNode
      Return null (empty tree node).
```

In M-tree active sub-trees and their lower bounds are not sorted in the priority queue *PR*, which means that the sub-tree with the minimal lower bound has to be searched before it can be accessed by the query object. Another disadvantage of M-tree is that its algorithm has to travel from the root node to all sub-trees in order to locate the parent node of the current node. In order to avoid the first disadvantage, in step S1.2 of the *k*-NN query algorithm of the SC-tree, the sub-trees are sorted in ascending order of their lower bounds in the *PR*. As a result, in step S1.3, the first sub-tree stored in the *PR* is the one with the minimal lower bound, i.e., the nearest one to the query object. Furthermore, it also speeds up step S1.3.3.2.2 as a result of the sorted lower bounds of sub-trees. To avoid the second disadvantage, the SC-tree simply returns the attributes *parentNode* and *entryIndex* of current node.

In order to test the indexing and query efficiency of the SC-tree, experiments are carried out and analyzed in the next section.

4 Experimental Results and Analyses

Experiments have been carried out on five high-dimensional datasets: Census, Corel, FaceR, Forest, and Synthetic, in order to evaluate the performance of the proposed method. The Census data set contains 22,784 139-dimensional feature vectors, which is a highly clustered dataset with about 80% vectors clustered in 20% regions. The Corel dataset contains 68,040 32-dimensional image feature vectors. The FaceR dataset contains 2000 99-dimensional feature vectors extracted from face images. The Forest dataset contains 41012 54-dimensional feature vectors among which 44 are Boolean attributes and 10 are real-valued attributes. Finally, the Synthetic data set, generated by Aggarwal [1], contains 12,040 40-dimensional feature vectors, which is a very sparse data set. In our experiments, the proposed SC-tree and the M-tree are tested on the five data sets. All objects in the data set are inserted into the M-tree and the SC-tree one by one in the indexing stage. Every object acts as a query object during the *k*-NN query stage. The number of children nodes for each node (fan-out), denoted as *CN*, is chosen from 10, 20, 30, 40, 50, and 60. The indexing efficiency is measured by indexing time, while the *k*-NN query efficiency is measured by the query time spent on all the query objects in a data set. The query accuracy is measured by a query accuracy ratio, which is defined as the ratio of the number of correctly returned query results to the total number of query results.

Firstly, the indexing performance of the M-tree and the SC-tree are compared in Fig. 1~5. It is clear that the indexing time of the SC-tree is much shorter than that of the M-tree. In fact, the SC-tree is about 20% quicker than the M-tree on five data sets. It is probably because in the M-tree objects were inserted into nodes randomly, but in the SC-tree objects were inserted into nodes in order. As a result, in the split operation of the SC-Tree, the first and the last entries can be easily used for choosing routing objects. However, the split operation of the M-tree has to calculate the distance between every pair of entries in a node to select two entries with the farthest distance. Both the SC-tree and the M-tree construct indexing structures by inserting objects in the data set one by one. It is reasonable to analyze the efficiency of inserting and splitting operations to reflect the indexing efficiency. Let the number of entries in a node be np . For the SC-tree, the time to insert one object into a node is $O(\log(np))$, and the time to select two new routing objects from the split node in splitting operation equals to $O(1)$ by selecting the first object and the last object as the new routing objects, thus the total indexing time complexity of SC-tree can be approximated as $[O(\log(np)) + O(1)] \approx O(\log(np))$. For the M-tree, the time to insert one object into a node is $O(1)$ by adding the object to the end of the node directly, the time to select routing objects from the split node in splitting operation is $C_n^2 = np(np-1)/2$ by comparing every pair of objects in the split node, and thus the total indexing time complexity of M-tree can be approximated as $[O(np(np-1)/2) + O(1)] \approx O(np^2)$. From the analysis of the indexing time complexity, the indexing time of the SC-tree is much shorter than that of the M-tree.

Secondly, the query time of the two methods are compared in Fig. 6~10. It can be seen that the *k*-NN query time of the SC-tree is shorter than that of the M-tree. In the *k*-NN search algorithm, the chosen node in the M-tree is added into the priority queue *PR* without sorting the positions of sub-trees according to the lower bounds between the query object and sub-trees. As a result, it influences the order of accessing nodes. While the SC-tree sorts the sub-trees in ascending order of the lower bounds between the query object and sub-trees, thus the nearest sub-tree to the query can be accessed firstly. Consequently, the SC-tree reserves better candidate objects and prunes irrelative sub-tree at an earlier stage, which greatly reduces distance calculations. Another important difference between the SC-tree and the M-tree is the pointer to the parent of the current object. If there are *pp* objects in the indexing tree, the level of the tree is $O(\log pp)$. For instance, if a tree has two entries in each node and contains 8 objects, then the level of the tree equals $\log_2(8) = 3$. The M-tree has to locate its parent node by travelling the indexing tree, which starts from the root node until the node itself is reached. The time complexity of travelling in the M-tree is $O(\log pp)$, whilst it is $O(1)$ in the SC-tree directly using pointer *parentNode[entryIndex]*. Because locating a parent node is a very frequent operation in both indexing and query, this complexity has a great impact on the indexing and query efficiency.

Finally, the query accuracy ratios for the M-tree and the SC-tree are very similar, as shown in Fig. 11 and Fig. 12 respectively. The query accuracy ratios of both the

SC-tree and the M-tree are between 92% and 99% when *NC*=10~60, which are quite stable.

From the above experimental results and analyses, it can be concluded that, without sacrificing the query accuracy, the SC-tree largely improves the indexing and query efficiency in comparison with the M-tree.

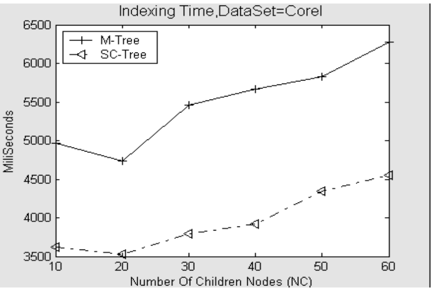


Fig. 1. Indexing time on Corel

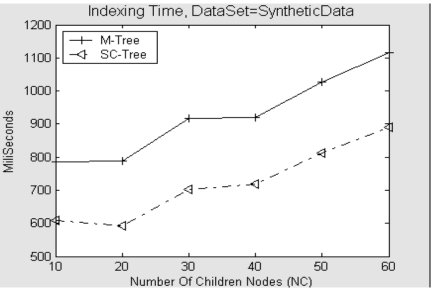


Fig. 2. Indexing time on Synthetic Data

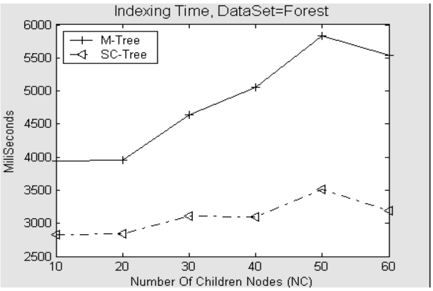


Fig. 3. Indexing time on Forest

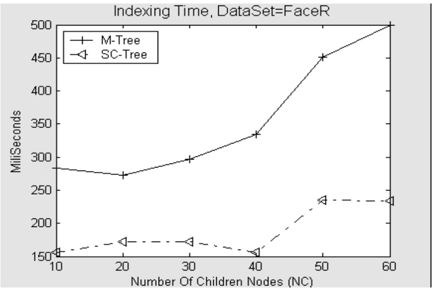


Fig. 4. Indexing time on FaceR

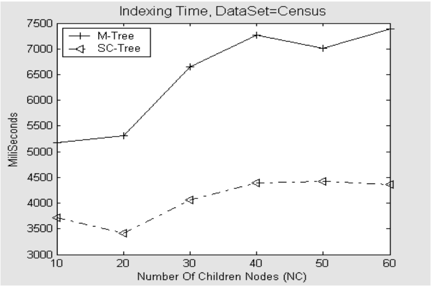


Fig. 5. Indexing time on Census

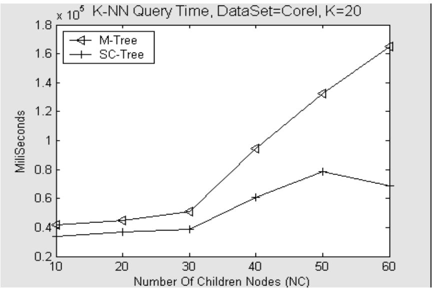


Fig. 6. *k*-NN query time on Core

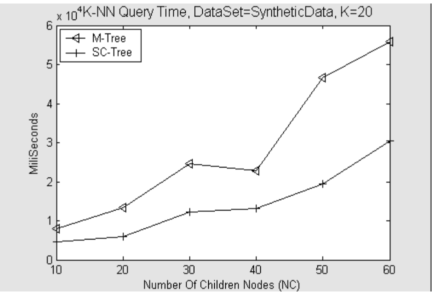


Fig. 7. *k*-NN query time on Synthetic Data

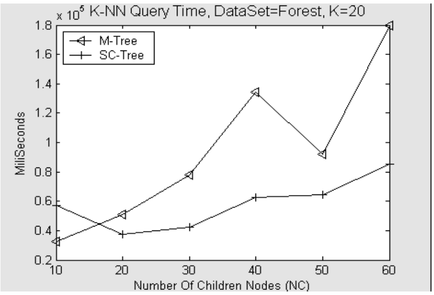


Fig. 8. *k*-NN query time on Forest

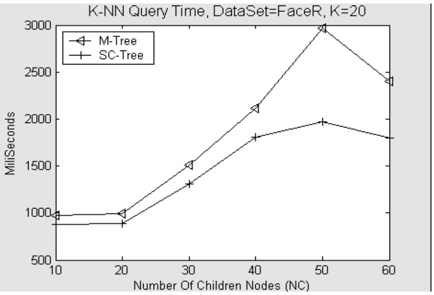


Fig. 9. *k*-NN query time on FaceR

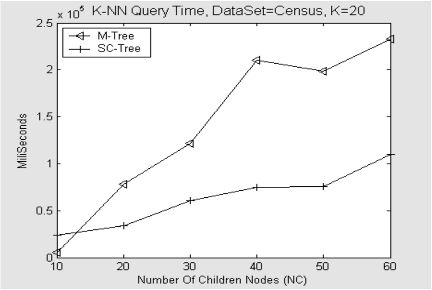


Fig. 10. *k*-NN query time on Census

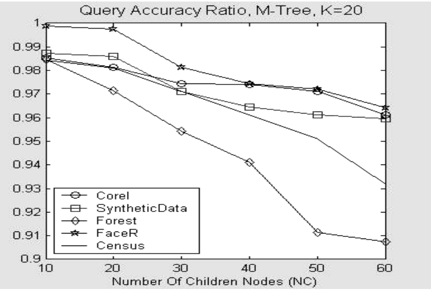


Fig. 11. *k*-NN query accuracy ratios on 5 data sets by M-tree

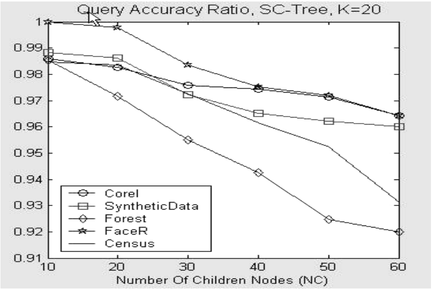


Fig. 12. *k*-NN query accuracy ratios on 5 data sets by SC-tree

5 Conclusion

M-tree is an efficient dynamic indexing structure which indexes and queries data objects from a generic metric space and utilizes the triangle inequality postulate to prune irrelative sub-trees during the query stage. This paper proposes an SC-tree indexing structure which inherits the advantages of the M-tree and overcomes its disadvantages. Experimental results and complexity analyses show that the SC-tree is much more efficient than the M-tree with respect to the query time and indexing time without sacrificing its query accuracy.

References

1. Aggarwal, C. C., Procopiuc, C., Wolf, J.L., Yu, P. S., Park, J. S.: Fast algorithms for projected clustering. Proc. of the ACM SIGMOD Conference, Philadelphia, USA (1999) 61-72
2. Bartolini, I., Ciaccia, P., Patella, M.: String matching with metric trees using an approximate distance. Proc. of the 9th Int. Symposium on String Processing and Information Retrieval (SPIRE), Lisbon, Portugal (2002) 271-283
3. Beckmann, N., Kriegel, H. P., Schneider, R., Seeger, B.: The R*-tree: An efficient and robust access method for points and rectangles. Proc. of the 1990 ACM SIGMOD International Conference on Management of Data, Atlantic City, NJ (1990) 322-331
4. Berchtold, S. Keim, D. A., Kriegel, H. P.: The X-tree: An index structure for high-dimensional data. Proc. 22nd Int. Conference on Very Large DataBases (VLDB), Bombay, India (1996) 28-39
5. Bozkaya, T., Ozsoyoglu, M.: Distance-based indexing for high-dimensional metric spaces. Proc. of ACM SIGMOD, Tucson, USA (1997) 357-368
6. Brin, S.: Near neighbor search in large metric spaces. Proc. 21nd Int. Conference on Very Large DataBases (VLDB), San Francisco, USA (1995) 574-584
7. Ciaccia, P., Patella, M., Zezula, P.: M-tree: An efficient access method for similarity search in metric spaces. Proc. Int. Conference of VLDB, Athens, Greece (1997) 522-525
8. Ciaccia, P., Patella, M., Zezula, P.: Processing complex similarity queries with distance-based access methods. Proc. of the 6th EDBT, Spain (1998) 9-13
9. Ciaccia P., Patella, M.: PAC nearest neighbor queries: Approximate and controlled search in high-dimensional and metric spaces. Proc. of the 16th Int. Conference on Data Engineering (ICDE), California, USA (2000) 244-255
10. Ciaccia , P., Patella, M.: Searching in metric spaces with user-defined and approximate distances. ACM Transactions on Database Systems, Vol. 27, (2002) 398- 437
11. Ciaccia,P., Nanni, A., Patella, M.: A query-sensitive cost model for similarity queries with M-tree. Proc. of the 10th Australasian Database Conference (ADC), New Zealand, (1999) 65-76
12. Finkel, R., Bentley, J. : Quad-trees: A data structure for retrieval on composite keys. ACTA Informatica, Vol. 4, (1974) 1-9
13. Guttman, A.: R-trees: A dynamic index structure for spatial searching. Proc. of ACM SIGMOD, Boston, USA (1984) 47-57
14. Heisterkamp, D. R., Peng, J.: A kernel vector approximation file for nearest neighbor search using kernel methods. Proc. of the 6th Kernel Machines Workshop at Neural Information Processing Systems Conference, Whistler, Canada (2002) 1-12
15. McNames, J.: A fast nearest neighbor algorithm based on a principal axis search tree. IEEE Transactions on Pattern Analysis and Intelligence, Vol. 23, (2001) 964-976
16. Nievergelt, J., Hinterberger, H., Sevcik, K.C.: The grid file: An adaptable, symmetric multikey file structure. ACM Trans. on Database Systems, Vol. 9, (1984) 38-71

17. Robinson, J.: The KDB-tree: A search structure for large multidimensional dynamic indexes. Proc. of the ACM SIGMOD Int. Conference on Management of Data, Ann Arbor, Michigan (1981) 10-18
18. Uhlmann, J. K.: Satisfying general proximity/similarity queries with metric trees. Information Processing Letters, Vol. 40, (1991) 175-179
19. Zezula,P., Savino, P., Amato, G., Rabitti, F.: Approximate similarity retrieval with M-trees. VLDB Journal, Vol. 7, (1998) 275-293

A Heterogeneous Computing System for Data Mining Workflows

Ping Luo[1,2], Kevin Lü[3], Qing He[2], and Zhongzhi Shi[1]

[1] Key Laboratory of Intelligent Information Processing,
Institute of Computing Technology, Chinese Academy of Sciences,
P.O. Box 2704-28, Beijing 100080 China

[2] Graduate School of the Chinese Academy of Sciences, Beijing, China

[3] Brunel University, Uxbridge, U.K. UB8 3PH

luop@ics.ict.ac.cn

Abstract. The computing-intensive Data Mining (DM) process calls for the support of a Heterogeneous Computing (HC) system, which consists of multiple computers with different configurations, connected by a high-speed LAN, for increased computational power and resources. DM process can be described as a multi-phase pipeline process, and in each phase there could be many optional methods. This makes the workflow of DM very complex and can be modelled only by a Directed Acyclic Graph (DAG). An HC system needs an effective and efficient scheduling framework, which orchestrates all the computing hardware to perform multiple competitive DM workflows. Motivated by the need of a practical solution of the scheduling problem for the DM workflow, this paper proposes a dynamic DAG scheduling algorithm according to the characteristics of execution time estimation model for DM jobs. Based on an approximate estimation of job execution time, this algorithm first maps DM jobs to machines in a *decentralized* and *diligent* (defined in this paper) manner. Then the performance of this initial mapping can be improved through *job migrations* when necessary. The scheduling heuristic used in it considers the factors of both the *minimal completion time* criterion and the *critical path* in a DAG. We implement this system in an established Multi-Agent System (MAS) environment, in which the reuse of existing DM algorithms is achieved by encapsulating them into agents. Practical classification problems are used to test and measure the system performance. The detailed experiment procedure and result analysis are also discussed in this paper.

Keywords: Data mining, heterogeneous computing, directed acyclic graph, multi-agent system environment.

1 Introduction

Current Data Mining (DM) tools contain a plethora of algorithms, but lack the guidelines to appropriately select and arrange these algorithms according to the nature of the problem under analysis. Given a practical DM problem,

D. Bell and J. Hong (Eds.): BNCOD 2006, LNCS 4042, pp. 177–189, 2006.

an expedient solution is to evaluate all the possible DM schemes modeled as a Directed Acyclic Graph (DAG), and rank them according to certain performance metrics. This yields a Grid computing problem, which aims to construct an Heterogeneous Computing (HC) system, supporting the executions of DM workflows.

An HC system, which consists of multiple computers with different configurations connected by a high-speed LAN, responses multiple computational requests of DM simultaneously. This system emerges as the provider of Internet-based data mining services, and offers an attractive option for small to medium range organizations, which are the most constrained by the high cost of data mining software, and consequently, stand to benefit by paying for software usage without having to incur the costs associated with buying, training and maintenance.

This study aims to construct such an HC system and mainly focuses on the effective and efficient scheduling framework to orchestrate all the computing hardware in it to perform multiple competitive DM workflows. According to the characteristics of execution time estimation model for DM jobs, we propose a *dynamic* scheduling framework for DM workflows. It has the following features:

- The scheduling operation performs in a totally *decentralized* and *diligent* manner, which avoids the computation bottleneck for centralized scheduling and increases the system robustness.
- This scheduling framework supports simultaneous computing of multiple competitive DAGs. The execution sequence of DM jobs considers the factors of both the precedence constraints in a DAG and the arrival order of these DAGs.
- This scheduling framework is tolerant to approximate time estimations of DM jobs. The initial mapping, based on the approximate running time estimations, will be improved by *job migrations*.

The arrangement of the rest of this paper is as follows. Section 2 describes the DM workflow for classification as a running example and formalizes the scheduling problem. In Section 3 we propose the dynamic scheduling algorithm for competitive DM workflows. Section 4 evaluates the performance of the data mining HC system with the presented scheduling algorithm by real-world datasets. The related work and conclusions will be given in Section 5. The discussions about the execution time estimation model for DM jobs, the implementation issues of this DM HC system in a Multi-Agent System (MAS) environment, and the details about the approximate execution time estimation method used in the experiment are omitted due to the space limitation. The full version of this paper can be downloaded from [1].

2 Data Mining Workflow for Heterogeneous Computing

2.1 Data Mining Workflow for Classification: A Running Example

The DM workflow for classification in Figure 1, used as a running example in this paper, aims to find the optimal classification pattern for the input dataset.

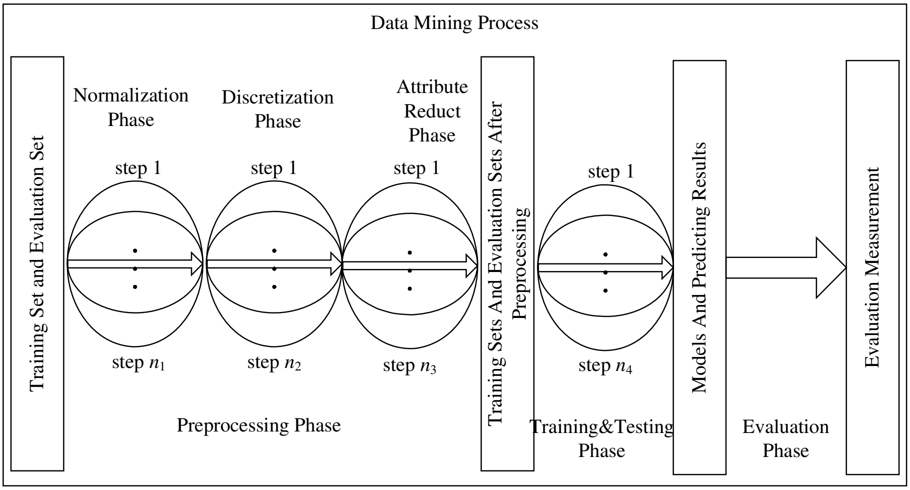


Fig. 1. Data mining process for classification

It is a complex, highly dynamic, and resource-intensive process, which consists of several different phases. In each phase, many different algorithms are available with different parameters. The workflow in Figure 1 consists of preprocessing, training&testing and evaluation phase. The preprocessing phase can be subdivided into three sequential sub-phases of normalization, discretization, and attribute reduction. The mining steps within a phase are optional operations, which would output different results. For convenience and clarity, we give the following definitions.

Definition 1 (DM Step). *A DM step corresponds to a particular algorithm to be executed, provided a dataset and a certain set of input parameters for it. Each DM step Λ is described as a quadruple:*

$$\Lambda = (A, F, D, \boldsymbol{P})$$

where A is the data mining algorithm, F is the data mining phase that contains the algorithm A, D is the input dataset and $\boldsymbol{P}$ is the vector of algorithm parameters.

Definition 2 (DM Path). *Let Λ_1=$(A_1, F_1, D_1, \boldsymbol{P_1}), \cdots, \Lambda_k$=$(A_k, F_k, D_k, \boldsymbol{P_k})$, DM Path is $\boldsymbol{\Lambda} = (\Lambda_1, \cdots, \Lambda_k)$, where $F_i (1 \leq i \leq k)$ is the i-th phase of the whole k-phase data mining process.*

In Figure 1, a DM path can be easily obtained after we select a DM step from each mining phase. If there are n_1, n_2, n_3, n_4 different DM steps in each of the four phases of normalization, discretization, attribute reduction and training&testing respectively, the number of all possible DM paths would be $n_1 \times n_2 \times n_3 \times n_4$ according to the Multiply Theorem. Along a DM path, a mining step transfers its output to the following step until the path terminates and the final result would be obtained. Then, using the training and validation datasets as an input of the DM path, a measurement will be obtained for this path according to certain evaluation criterion. For classification problems, the evaluation measurements could be accuracy, weighted accuracy and AUC (Area Under Curve), etc. After exhaustively evaluating all the DM paths, ranks of all resultant patterns for all DM paths are generated.

2.2 Workflow Model of Data Mining

We model the DM workflow as a weighted DAG, $G = G(V, E)$, where $V = \{v_1, \cdots, v_n\}$ is a set of weighted nodes and E is a set of weighted directed edges, representing data dependencies and communications between nodes. A node in the DAG represents a job (referred to as the corresponding DM step), which must be executed without preemption on a host. Consider the HC system consisting of l machines $m_1, \cdots, m_l$, the weight vector of a node v is referred to as the computation cost vector $\boldsymbol{\Delta(v)} = \{\Delta(v, m_1), \cdots, \Delta(v, m_l)\}$, where $\Delta(v, m_i)$ represents its execution time on a machine m_i. $e_{ij} = (v_i, v_j) \in E$ indicates data transportation from job v_i to v_j, and $|e_{ij}|$ represents communication cost between these two jobs if they are not executed on the same machine. The precedence constraints of a DAG require that a node should not start executing before it gathers all the data from its predecessors. The node without predecessors is called the *entry* of G. The node without successors is called the *end* of G. The *critical path* of G is the longest path (there can be more than one longest path) from an entry to an end of G. The weight of this path is the sum of the weights of the nodes and edges along this path. In the following, a *task* refers to a DAG and a *job* refers to a node in a DAG.

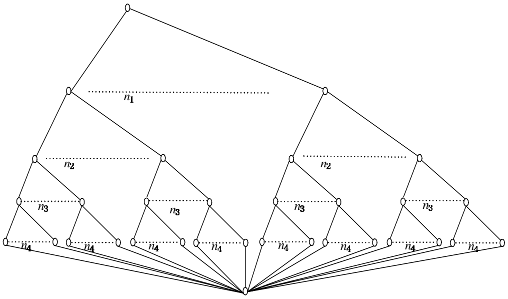


Fig. 2. The DAG of classification workflow

Figure 2 is the corresponding un-weighted DAG of the DM process in Figure 1. The direction of all the edges in Figure 2 is from the node in the upper layer to the one in the lower layer. If we feed the dataset to the uppermost node in Figure 2, after the whole computation the lowermost node in this figure will output the rank of all patterns for all DM paths, indicating the optimal classification pattern.

3 Dynamic Scheduling for Competitive DAGs in an HC System

We consider the following 4 issues in developing the scheduling algorithm within an HC system.

- The time estimation model for DM jobs. It is assumed to be provided in advance as a function with three parameters: 1) DM algorithm, 2) feature vector of input data and 3) user-specified algorithm attributes. Thus, the execution time of the node is not known a priori until its input datasets are all gathered. When a DAG is being processed, only if all the predecessors of a node are finished, the estimation model then can use the gathered immediate results to predict the execution time of this node. Therefore, the mapping process must be performed during the job executions and only *dynamic* scheduling can be adopted under this situation.
- Because it is hard to induce an accurate execution time estimation model of DM jobs, this scheduling algorithm should be tolerant to approximate time estimations of DM jobs.
- To avoid the the computation bottleneck for centralized scheduling and increase the system robustness, this algorithm should be totally decentralized.
- When multiple competitive DAGs arrive at an HC system, the execution sequence of DM jobs should consider the factors of both the precedence constraints in a DAG and the arrival order of these DAGs.

Therefore, the scheduling, in fact, can be described as a problem of *dynamic scheduling for competitive DAGs*. The scheduling objective is to minimize the average makespan (the time when the last job of a DAG finishes) of competitive DAGs. This problem has been proved, in general, to be NP-complete [2], thus requiring the development of heuristic techniques [3, 4] for practical usage. In this paper we propose a scheduling framework, which satisfies the aforementioned issues.

It should be noted that the communication cost between computers within an HC system is ignored due to the following reasons: 1) the network bandwidth within an HC system is high speed and 2) even if the volume of the transferred data is large, its corresponding processing time on a computer is much longer than its communication time.

3.1 Decentralized and Diligent Job Mapping

We propose a *decentralized* and *diligent* scheduling algorithm, compared with the algorithm in [3], which performs in a *centralized* and *lazy* manner. A scheduler resides on each machine. When a job Λ is finished, the scheduler on the same machine will find all the *ready* jobs (A job is *ready* when all the input data from its predecessors are available) in the successive nodes of Λ, and then map them to suitable machines immediately. The heuristic *min-min* [5] for mapping a class of independent jobs can be used to map this group of ready jobs. We call this scheduling paradigm diligent in the sense that the mapping decision is made as soon as a job is ready. The pseudo-code for scheduling algorithm on each machine of an HC system is presented in Algorithm 1.

Algorithm 1. Scheduling Algorithm on Each Machine of an HC system

```
if a job $\Lambda$ finished on the same machine then
    $S = \{\Lambda^{'} | \Lambda^{'} is\ the\ successor\ of\ \Lambda\}$
    $S^{'} = \{\Lambda^{''} | \Lambda^{''}\ is\ ready\ and\ \Lambda^{''} \in S\}$
    while $S^{'} \neq \Phi$ do
        according to min-min, find the best pair of job $\Lambda^{''} \in S^{'}$ and machine $m$, based
        on current job pending queues of each machine
        map job $\Lambda^{''}$ to machine $m$
        update the job pending queue on machine $m$
        $S^{'} = S^{'} - \{\Lambda^{''}\}$
    end while
end if
```

3.2 Job Execution Control with Priority

Usually, the pending job queue E on each machine, which stores all the waiting jobs for executing, is processed in a FIFO manner. To consider the critical path factor of a DAG, the jobs from a DAG will be executed in descending order of their estimation times. Thus, Algorithm 2 for job execution control is proposed, which supports the execution of multiple DAGs. In this algorithm when a new job arrives at a machine and suppose at the same time the machine is executing another job, it will be inserted into the job queue E at a suitable position, to keep that the jobs from *the same* DAG are arranged in descending order of their estimation times while the positions of the jobs from the other DAGs in the queue E will not be changed. Then, the jobs in the pending job queue E are processed in a FIFO manner.

Altogether Algorithms 1 and 2 form the whole heuristic scheme, which consider both the minimal completion time criterion and the critical path of a DAG. These two factors are integrated and implemented in job mapping process and job execution control, respectively.

3.3 Job Migration After Initial Mapping

The system efficiency of an HC system is defined in (1)

$$\eta = \frac{t_{computation}}{t_{total}} \tag{1}$$

where $t_{computation}$ is the system CPU time for computation and t_{total} is the total system CPU time. Then the system waste μ is defined in (2)

$$\mu = \frac{t_{idle}}{t_{total}} \tag{2}$$

where t_{idle} is the system blocking time and t_{total} is the same as the one in (1). It is clear that $\eta + \mu = 1$ because $t_{computation} + t_{idle} = t_{total}$. Furthermore, the system waste μ can be divided into two parts: the intrinsic system waste μ_i and the system waste μ_s caused by approximate execution time estimations of DM jobs.

Algorithm 2. Algorithm for Job Execution Control with Priority on machine m

```
loop
  $E$ is the job queue on machine $m$
  if a new job $\Lambda$ arrives at machine $m$ and the machine is executing one another job then
    if $|E|$==1 then
      append $\Lambda$ to the end of $E$
    else
      $newPosition = |E| + 1$
      for $i = |E|$ to 2 do
        $\Lambda^{'}$ is the $i$-th element of $E$
        if $\Lambda$ and $\Lambda^{'}$ are from the same DAG then
          if $\Delta(\Lambda^{'}, m) < \Delta(\Lambda, m)$ then
            move $\Lambda^{'}$ to the $newPosition$-th position of $E$
            $newPosition = i$
          else
            break
          end if
        end if
      end for
      move $\Lambda$ to the $newPosition$-th position of $E$
    end if
  end if
  if a job-finished notification received then
    remove the front job of $E$
    if $E \neq \Phi$ then
      execute the new front job of $E$
    end if
  end if
end loop
```

Consider the HC system consisting of l machines $m_1, \cdots, m_l$ and the job pending queue (including the current executing job) on each machine is $E_1, \cdots, E_l$, respectively. $|E_i|$ $(0 \leqslant i \leqslant k)$ is the number of jobs in E_i. μ_i counts at the time that $\exists E_i$ such that $|E_i| = 0$ and $\nexists E_j$ such that $|E_j| > 1$. This kind of system waste is intrinsic, because a job is the computation atom, representing the minimal granularity for parallelization, and can be only executed on a single machine. The other kind of system waste μ_s increases while $\exists E_i, E_j$ such that $|E_i| = 0$ and $|E_j| > 1$. It is caused by the mapping decision based on inaccurate execution time estimations of DM jobs. μ_i is intrinsic, so it is unavoidable. And μ_s is seemingly also unavoidable because the task for accurate time estimation of DM jobs is so difficult. However, the technique of job migration after initial mapping can decrease μ_s. The key point of the job migration is that when $|E_i| = 0$ and $|E_j| > 1$ a suitable job Λ in E_j would migrate from m_j to m_i and begins executing immediately on m_i. The satisfying condition for migration is that $t_{completion}(\Lambda, m_j) > \Delta(\Lambda, m_i)$, which means that the completion time of Λ on m_i is early than that on m_j. Conformed to the job execution priority in 3.2, the job in the front of the job pending queue is firstly selected to check the migration condition. Thus, a system monitor is created for the whole HC system, checks the job pending queue on each machine every T_m time units and is responsible for job migrations. The pseudo-code for this system monitor is in Algorithm 3.

Algorithm 3. Algorithm for Job Migration after Initial Mapping

```
while ($t$=the current system time) $mod\ T_m$=0 do
  receive the copy of job pending queues on each machine $\boldsymbol{E} = \{E_1, \cdots, E_l\}$
  $\boldsymbol{E}_{idle} = \{E_i|E_i \in \boldsymbol{E}\ and\ |E_i| = 0\}$ {In $\boldsymbol{E}_{idle}$, $E_i$ is arranged in decrease order of
  the computing speed of the corresponding host, which $E_i$ is from}
  $\boldsymbol{E}_{busy} = \{E_i|E_i \in \boldsymbol{E}\ and\ |E_i| > 1\}$ {In $\boldsymbol{E}_{busy}$, $E_i$ is arranged randomly}
  if $|\boldsymbol{E}_{idle}| > 0$ and $|\boldsymbol{E}_{busy}| > 0$ then
    $\boldsymbol{E}(i)$ is the $i$-th element of $\boldsymbol{E}$
    for $i = 1$ to $|\boldsymbol{E}_{busy}|$ do
      pop the front of $\boldsymbol{E}_{busy}(i)$ {the front is running on machine $i$}
    end for
    $q = \sum_{E_i \in \boldsymbol{E}_{busy}} (|E_i| - 1)$
    $E_{candidate}$=null {$E_{candidate}$ is the queue of candidate jobs for migration}
    while $|E_{candidate}| < q$ do
      for $i = 1$ to $|\boldsymbol{E}_{busy}|$ do
        if $\boldsymbol{E}_{busy}(i)$ is not empty then
          $\Lambda$ =pop the front of $\boldsymbol{E}_{busy}(i)$
          add $\Lambda$ to $E_{candidate}$
        end if
      end for
    end while
    while $|E_{candidate}| > 0$ and $|\boldsymbol{E}_{idle}| > 0$ do
      $\Lambda$ =pop the front of $E_{candidate}$
      $m$ is the machine which owns $\Lambda$
      $m^{'}$ is the machine which owns $\boldsymbol{E}_{idle}(0)$
      if $t_{completion}(\Lambda, m) > \Delta(\Lambda, m^{'})$ then
        $\Lambda$ migrates from $m$ to $m^{'}$
        pop the front of $\boldsymbol{E}_{idle}$
      end if
    end while
  end if
end while
```

3.4 The Overall Scheduling Framework

In summary, the scheduling framework consists of three parts: 1) a scheduler on each machine, 2) a job execution controller on each machine, and 3) a system monitor for job migration. The job mapping process starts immediately after a job on the same machine is finished. All the ready jobs in the successors of the finished job are mapped, by the scheduler on the same machine, to the machines according to Algorithm 1. According to Algorithm 2 the job execution controller is responsible for inserting a mapped job into a suitable position and executing them one after another. The system monitor migrates a job according to Algorithm 3. Therefore, this scheduling framework first maps jobs to machines in a decentralized and diligent manner, based on a approximate estimation of job execution time. Then the performance of this initial mapping can be improved through the use of job migration. The scheduling heuristic considers both the minimal completion time criterion and the critical path in a DAG. These two aspects are integrated and implemented in job mapping process and job execution control process, respectively.

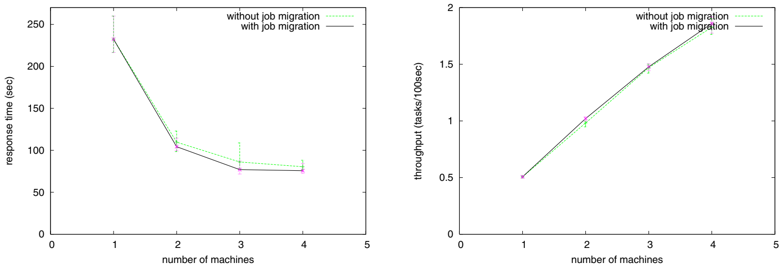
4 Experiment Procedure and Results

Based an established MAS environment named MAGE [6], we have developed a data mining HC system with the newly proposed scheduling framework. In our experiment 9 machines with different configurations are used to form this HC system. The main configurations of these machines are listed in Table 1. The performance metrics measured in the experiments include task response-time, system throughput and system efficiency defined in the following. To measure these metrics, a DM task for classification denoted by G^*, is constructed for the whole experiment process. The corresponding DAG of this task, which contains 16 jobs, is isomorphic with the DAG in Figure 2. After removing the end node of the DAG it becomes a tree, which indicates that all the successors of an internal node in the tree can be mapped once its execution is completed. The input data for this DAG is from a practical classification problem, well logging analysis to identify the pay zones of gas or oil in the reservoir formations. It contains 2000 labeled examples with 10 numeric condition attributes.

Table 1. Machine Configuration List

Machine Type Index	CPU	Main Memory	Machine Amount
1	3 GHz	512 M	5
2	2.8 GHz	512 M	1
3	2.4 GHz	1024 M	1
4	2.2 GHz	512 M	1
5	731 MHz	448 M	1

The experiments are performed in two parts. In the first part, the 4 machines from Machine Type 1 are used to form a homogeneous system, in order to measure task response time and system throughput versus the number of joining machines with the same configuration. Let the arrival time of the task G be $a(G)$, the completion time of G be $c(G)$, then the response-time of G is $r(G) = c(G) - a(G)$. The system throughput is defined by the number of G^*s, which are completed by the system in a fixed time.



(a) response time versus homogeneous machines

(b) throughput versus homogeneous machines

Fig. 3. The experimental results for homogeneous computing

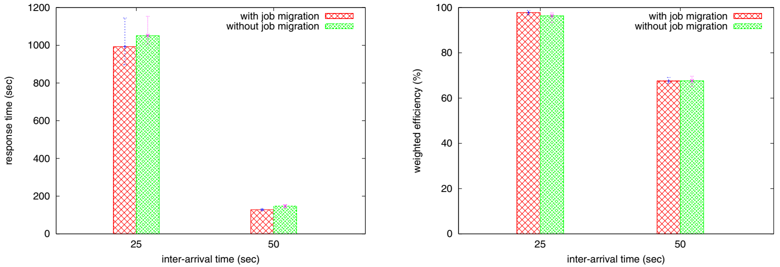
The second part of the experiments is to evaluate the scheduling performance in a heterogeneous system, which contains all the 9 machines listed in Table 1. In these experiments exponential distribution is used to generate the task sequence, including 100 G^*s. These tasks are assigned under two average task arrival intervals, t_l and t_h, where $t_l = 25$ seconds, $t_h = 50$ seconds. The task arrival time is generated, which satisfies $\frac{|t_a - t|}{t} < 0.06$, where t_a is the actual average inter-arrival time of the task sequence and t is the expected inter-arrival time. We record the average response time of the tasks in the sequence and compute the *weighted* system efficiency in (3), which considers the machine heterogeneity in an HC system.

$$\eta_{weighted} = \frac{t_{computation}}{t_{total}} = \frac{\sum_{i=1}^{l} \frac{t_{computation}(i)}{\rho_i}}{\sum_{i=1}^{l} \frac{t_{total}(i)}{\rho_i}} \tag{3}$$

where $t_{computation}(i)$ is the system CPU time for the computation on machine i, $t_{total}(i)$ is the total system CPU time on machine i, l is the number of machines in our system, and ρ_i is the performance coefficient for machine i. All the above experiments are performed under two situations, with and without job migrations after initial mapping, and repeated five times.

Figure 3(a) and Figure 3(b) show the results from the first part of experiments. Figure 3(a) illustrates that the response time of a single task G^* decreases along with the increase of the number of machines. However, the response time decreases in a non-linear manner and eventually reaches at a minimal level, because in our application the minimal computing granularity is a job, which could not be broken down any further for parallelization. In theory, the minimum response time of a DAG is the weight sum of the critical path in the DAG. Figure 3(b) shows that the throughput of the HC system increases close to linear along with the increase of the number of joining machines. These two figures also show that the use of job migration could improve the system performance in terms of task response time and system throughput.

The results from the second part of the experiments can be seen in Figure 4(a) and Figure 4(b). In Figure 4(a) it can be found that through the use of job



(a) average response time of the tasks in task sequence

(b) weighted efficiency when executing the tasks in task sequence

Fig. 4. The experimental results for heterogeneous computing

migration technique the average response times of the 100 tasks decrease 5.58% and 13.21% for the cases of 25-second inter-arrival and 50-second inter-arrival, respectively. The weighted efficiency of the HC system is also improved through job migration, as shown in Figure 4(b).

5 Related Work and Conclusions

The issues of building a computational Grid for Data Mining have been recently addressed by a number of researchers. WEKA4WS [7] adapts the Weka toolkit to a Grid environment and exposes all the 78 algorithms as WSRF-compliant Web Services. FAEHIM (Federated Analysis Environment for Heterogeneous Intelligent Mining) [8] is Web Services based on a toolkit of DM and mainly focuses on the composition of existing DM Web Services by Triana problem solving environment [9]. The Knowledge Grid [10, 11] is a reference software architecture for geographically distributed knowledge discovery systems. It is built on top of a computational Grid of Globus and uses basic Grid services to implement the DM services on connected computers. A visual environment for Grid application (VEGA) is developed in this system, supporting visual DM plan generation and automatic DM plan execution.

To make good use of the computing hardware in heterogeneous systems for DM workflow a scheduling framework is urgently needed. Although this computing paradigm can be achieved by exposing all the DM algorithms as Web Services on every host in this system or by dynamic Web Service deployment, however, the scheduling framework for DM DAG applications, in general, has drawn a very little attention except for the scheduling heuristics mentioned in [11]. Paper [11] also emphasizes the importance of scheduling algorithm in Knowledge Grid and uses the concept of *abstract hosts* to represent any computing host.

To the best of our knowledge, the study in this paper is the first attempt in developing a data mining HC system with an efficient and effective scheduling framework. It is formalized as a problem of dynamic scheduling for competitive DM DAGs in a heterogeneous computing system. According to the characteristics of execution time estimation model of DM jobs, a new scheduling framework is presented with three features: totally decentralized, the hybrid heuristic scheme, and the use of job migration after initial mapping. The DM computing platform with this scheduling framework has been implemented in a multi-agent system environment. Its performance has also been tested by real-world datasets, which is demonstrated by our experiments. It should also be noted that the scheduling framework in this paper is a generic dynamic scheduling algorithm for DAGs, and thus has wide applicability in other fields besides data mining.

Acknowledgements

Our work is supported by the National Science Foundation of China (No.60435010), the national 863 Project (No.2003AA115220), the national 973 Project (No.2003CB317004) and the Nature Science Foundation of Beijing (No.4052025). Kevin Lü gratefully acknowledges the support of K.C.Wong Education Foundation, Hong Kong.

References

1. Ping Luo, Kevin L*ü*, Qing He, and Zhongzhi Shi. A heterogeneous computing system for data mining workflows. Technical report, Institute of Computing Technology, Chinese Academy of Sciences, 2006. http://www.intsci.ac.cn/users/luop/.
2. D. Fernandez-Baca. Allocating modules to processors in a distributed system. *IEEE Transaction on Software Engineering*, 15(11):1427–1436, 1989.
3. Michael Iverson and Fusun Ozguner. Dynamic, competitive scheduling of multiple dags in a distributed heterogeneous environment. In *Proceedings of the Eighth Heterogeneous Computing Workshop*, 1999.
4. Rizos Sakellariou and Henan Zhao. A hybrid heuristic for dag scheduling on heterogeneous systems. In *Poceedings of the 13th Heterogeneous Computing Workshop*, 2004.
5. Tracy D. Braun, Debra Hensgen, Richard F. Freund, Howard Jay Siegel, Noah Beck, Lasislau L. Boloni, Muthucumara Maheswaran, Albert I. Reuther, James P. Robertson, Mitchell D. Theys, and Bin Yao. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, 61(6):810–837, 2001.
6. Zhongzhi Shi, Haijun Zhang, Yong Cheng, Yuncheng Jiang, Qiujian Sheng, and Zhikung Zhao. Mage: An agent-oriented programming environment. In *Proceedings of IEEE International Conference on Cognitive Informatics*, pages 250–257, 2004.
7. D. Talia, P. Trunfio, and O. Verta. Weka4ws: a wsrf-enabled weka toolkit for distributed data mining on grids. In *Proceedings of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases*, Porto, Portugal, 2005.
8. Ali Shaikh Ali, Omer F. Rana, and Ian J. Taylor. Web services composition for distributed data mining. In *Proceedings of International Conference on Parallel Processing Workshops*, pages 11–18, 2005.

9. The Triana Problem Solving Environment. http://www.trianacode.org.
10. M. Cannataro and D. Talia. Knowledge grid an architecture for distributed knowledge discovery. *Communication of the ACM*, 46(1), 2003.
11. M. Cannataro, A. Congiusta, A. Pugliese, D. Talia, and P. Trunfio. Distributed data mining on grids: Services, tools, and applications. *IEEE Transactions on Systems, Man and Cybernetics*, 34(6):2451– 2465, 2004.

An Efficient System for Detecting Outliers from Financial Time Series

Carson Kai-Sang Leung, Ruppa K. Thulasiram, and Dmitri A. Bondarenko

The University of Manitoba, Winnipeg, MB, Canada
{kleung, tulsi, umbonda1}@cs.umanitoba.ca

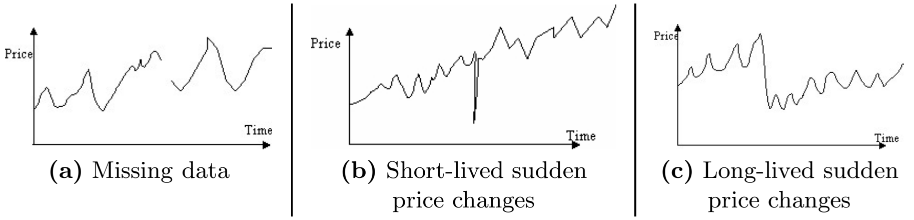
Abstract. In this paper, we develop an efficient system to detect outliers from real-life financial time series comprising of security prices. Our system consists of a data mining algorithm and a statistical algorithm. When applying each of these two algorithms individually, we observed its strengths and weaknesses. To overcome the weaknesses of the two algorithms, we combine the algorithms together. By so doing, we efficiently detect outliers from the financial time series. Moreover, the resulting (processed) datasets can then be used as input for some financial models used in forecasting future security prices or in predicting future market behaviour. This shows an alternative role of our outlier detection system—serving as a pre-processing step for other financial models.

Keywords: Databases, data mining, computational finance.

1 Introduction

Data mining refers to the search for implicit, previously unknown, and potentially useful information or patterns that might be embedded in data. Many of the existing studies focused on finding patterns that apply to the majority of items in the dataset [1, 2, 8, 10, 11, 14]. However, patterns that apply to the minority of items can also be interesting and important. For example, a rare event could be an indication of some unusual, suspicious, or criminal activities. Hence, several studies focused on *outlier detection* [6, 7, 9, 13], which aimed to analyse and find these exceptional activities from datasets like the performance statistics of professional athletes, workers' compensation data, and medical test data. Moreover, outlier detection could be used in various application areas such as e-commerce and finance. In this paper, we show how outlier detection can be applicable to financial data in an emerging cross-disciplinary area of research, known as *computational finance*, that addresses problems in finance or business (e.g., option pricing, portfolio management) by using advanced scientific computing or data mining techniques. In this area, several models have been developed to forecast future security prices and to predict future market behaviour. These models usually rely on standardised historical data, and are sensitive to data variations. *Any unusual noise (i.e., outliers / data polluters) present in the data may lead to incorrect forecast or prediction.* To ensure good prediction of price behaviour, many models require historical price data over a long

D. Bell and J. Hong (Eds.): BNCOD 2006, LNCS 4042, pp. 190–198, 2006.



(a) Missing data

(b) Short-lived sudden price changes

(c) Long-lived sudden price changes

Fig. 1. Data polluters

period of time (e.g., $\geq$ 10 years) for thousands of securities. Similarly, to ensure accuracy in applications like option pricing and risk management, several models require the input price series to be free from noise. Although the price series are normally obtained from reliable sources (e.g., Bloomberg), the series might still contain some *data polluters* (as shown in Fig. 1) such as: (a) missing data[1], (b) short-lived sudden price changes[2], and (c) long-lived sudden price changes[3].

Outliers can significantly influence financial model outputs (e.g., the forecast or prediction). Hence, to achieve better forecast or prediction, we need a system for detecting and eliminating outliers as well as pre-processing data. Algorithms in such a system should: (i) run efficiently on large datasets; (ii) detect both missing data and short-lived sudden price changes as *outliers*, and eliminate them; (iii) accommodate and ignore long-lived sudden price changes (as they should *not* be considered as outlier); and (iv) allow user input and control.

Over the past few years, some studies [12, 15, 16] in finance have suggested that noise/outlier detection from time series forms a fundamental problem. They often require data for their financial models to be free from noise. However, these studies mainly dealt with portfolio selection but *not* focused on outlier detection; they did not mention how to remove noise either. To detect outliers, other related works have been proposed. Some of them used statistical techniques like principal component analysis [5, 15] and independence component analysis [3], while some others used data mining techniques like clustering[4] [4] and anomaly detection [6, 7, 9]. However, most of these works did not use both data mining and statistical techniques. In contrast, we apply both techniques in this paper.

Our key contribution of this paper is the development of an efficient system for detecting outliers from financial time series. More specifically, our technical contributions of this paper are as follows:

[1] They could be caused by market closure in observation of a bank holiday, the stock not being traded on that day, or incomplete information at the sources.

[2] They could be caused by a price recording error or by market over-reaction.

[3] They are usually caused by a *stock split*—a situation when the stock price moves far up (under the normal condition), the issuing company splits the stock into two (or more) so as to keep up with the market demand while at the same time making it affordable for common investors by reducing the original price to half (or lesser).

[4] An item in the data is an *outlier* if it does not belong to any clusters.

- We develop a *data mining algorithm*, which uses a distance-based outlier detection technique to detect outliers from time series of security prices. The algorithm checks the values of security prices within certain distance so as to verify if the current price is an outlier.
- We also develop a *statistical algorithm*, which uses normal distribution properties of data to detect outliers. Moving averages are used in this algorithm to render the algorithm efficient.
- Due to their varying nature and properties, the above two algorithms may detect different outliers from the same time series. However, they are complementary. So, by putting them together into our outlier detection system, most (if not all) outliers can be effectively detected and removed. The resulting time series (i.e., the processed financial data) can then be used as input to existing financial models (e.g., neural network architecture) for a more accurate forecast of future security prices, a more accurate prediction of future market behaviour, and more accurate computation of option prices. This demonstrates an alternative role of outlier detection technique (as a pre-processor than as a stand-alone tool for obtaining insight into data distribution).

This paper is organised as follows. The next section describes our outlier detection system for financial applications. Section 3 shows experimental results. Finally, conclusions are presented in Section 4.

2 Our Proposed Outlier Detection System

In this section, we start describing our proposed outlier detection system, which consists of two phases. In Phase I, we identify the missing prices in a given dataset (i.e., financial time series), and substitute them with a new price. The new price is calculated in such a way that it ensures consistency with its neighbouring prices. This is done to avoid the introduction/generation of "artificial" outliers (noise). In Phase II, we run both a *data mining algorithm* and a *statistical algorithm* to detect those short-lived sudden price changes (i.e., outliers) based on data mining (or more specifically, distance-based data mining) and statistical approaches, respectively. We note that the execution of the data mining algorithm does not depend on the result of the statistical algorithm, and vice versa. Hence, by combining these two underlying algorithms, we nullify their individual weaknesses and speed up the outlier detection process.

2.1 Phase I: Identifying Missing Data

The goal of Phase I of our system is to identify gaps (i.e., missing data/prices, such as those depicted in Fig. 1(a), in the financial time series) and to fill them with new prices. With this respect, we develop a gap-identification algorithm based on the following realistic assumptions: There are five business days every week, and data for each business day are available (regardless whether it is a holiday or not). The algorithm, which has a linear complexity with a single scan

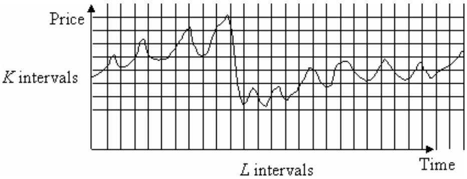


Fig. 2. The search space for the DetectOutliersDM algorithm

of the dataset, runs as follows. It scans the whole dataset once, and divides price items on a weekly basis. If any item for a certain weekday is missing, it is substituted with an item generated by a function, called *NewPrice*, which is used to obtain (i) an approximate value for the missing item or (ii) a new value of an item that is considered an outlier. The complexity of function *NewPrice* is linear with respect to the size of interval used for calculating the average. Here, we assume that items are normally distributed. Prices that are close (based on the date) are expected to influence each other to a greater extent than those prices that are far apart. So, in abstract terms, the new price can be computed based on the following equation:

$NewPrice = \left[\max\left(\sum_{i=1}^{n} I_{t-i}w_{ui}, 0\right)/2n\right] + \left[\max\left(\sum_{i=1}^{m} I_{t+i}w_{di}, 0\right)/2m\right]$, where I_t is an item in the time series, w_{ui} are the weights of the preceding (upstream) items, and w_{di} are the weights of the following (downstream) items; m and n are numbers of neighbouring data items in the downstream and upstream directions, respectively.

2.2 Phase II: Detecting Short-Lived Sudden Price Changes

Once the missing prices are identified, we can apply Phase II of our system to detect outlying prices from the financial time series. For this phase, we develop a data mining algorithm (*DetectOutliersDM*) and a statistical algorithm (*DetectOutliersStat*), and then effectively combine them into our system.

The Data Mining Algorithm. An algorithm in Phase II is a distance-based data mining algorithm called **DetectOutliersDM**. This algorithm is based on the FindAllOutsM algorithm [7], which uses a distance-based notion of outliers to detect outlying items. According to this notion, an item I_t in a dataset is an *outlier* if most items in the dataset lie at a distance greater than a user-defined threshold D from I_t.

The key idea of our DetectOutliersDM algorithm can be described as follows. The financial time series can be represented in the two-dimensional space with prices along the y-direction and time along the x-direction. Similarly, we divide the space between the maximum and minimum item values (i.e., price values) of the dataset into K equal intervals along the y-axis, where K is a user-specified constant and the size of each interval is (MaxPrice − MinPrice)$/K$. The space between the first and the last dates is divided into L intervals along the x-axis. We then map these L time intervals into the K price intervals so that each cell is a square and uses the same units of measurement. This scenario is depicted

```
DetectOutliersStat
(1) Calculate the sum and the sum of squares of the first $i$ items that follow the first item in the
    set (where $i$ is a constant or a user-defined value).
(2) For each item in the dataset:
    (a) Calculate the mean and the standard deviation for the current item based on the sum and
        the sum of squares. There are two means and two standard deviations for each item (based
        on items in the upstream and downstream directions).
    (b) Calculate item rankings, and compare them to some predefined threshold (which is either
        a constant or a user-defined value). An item is an outlier if both rankings are greater
        than the threshold. If the current item is marked as an outlier, replace it with a new item
        generated by NewPrice (as described in Section 2.1).
    (c) Update the sum and the sum of squares for the next item in the set.
```

Fig. 3. A Skeleton for the DetectOutliersStat algorithm

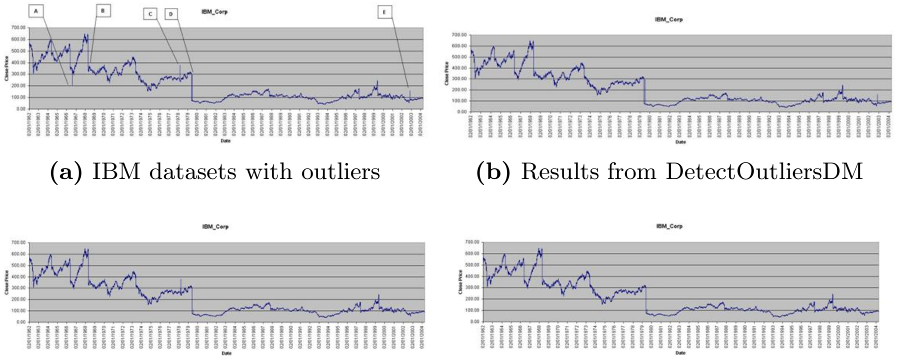
in Fig. 2. The distance that defines the neighbourhood for each item is then equal to:

$$D = (\text{MaxPrice} - \text{MinPrice})/K \times 2\sqrt{2}.$$

We use the Euclidean distance for calculation. With this setting, a price item is considered an *outlier* if it has insufficient number of neighbours within distance D (i.e., within a cell with low item density). For the financial time series (with each price quantised into a two-dimensional space), the complexity of this cell-based algorithm depends on the number of price items N in the dataset and the number of cells in the space. More precisely, the algorithm has a linear complexity with respect to N.

As a preview, we will show in Section 3 that our distance-based data mining algorithm is effective in detecting outliers (especially in detecting those short-lived sudden price changes that are lying away from the normal price range).

The Statistical Algorithm. The second algorithm in Phase II is a statistical based algorithm called **DetectOutliersStat**. Here, we make an assumption that items are normally distributed. Although it might not be true for the whole dataset, this assumption usually holds for many short continuous sub-groups in the dataset. The DetectOutliersStat algorithm is based on a statistical observation that most items are located within three standard deviations from the mean (or average). Thus, if an item is 10 standard deviations away from the mean, it is very likely to be an outlier. The distance that measured in standard deviations from the mean is defined as *item ranking*. Each item I_t has two such rankings: One ranking is based on the items that precede I_t (i.e., upstream) and another ranking is based on the items that follow I_t (i.e., downstream). In order to be considered an *outlier*, an item I_t needs to have both rankings greater than some specified thresholds. To improve efficiency, we avoid calculating the mean for each item from scratch; instead, we use the moving averages. Since the algorithm works in a sequential manner, the averages (means) for the current price item can be calculated by adjusting the averages for the upstream items. The complexity of this algorithm is linear, and it requires only a single scan of the entire dataset. The items that are marked as outliers are replaced with new items generated by the *NewPrice* function described in Section 2.1. Fig. 3 shows a skeleton of this statistical algorithm.



(a) IBM datasets with outliers

(b) Results from DetectOutliersDM

(c) Results from DetectOutliersStat

(d) Results from combined algorithm

Fig. 4. Experimental results on the IBM Corporation dataset

3 Experimental Results

We ran our proposed system over sets of real (historical) security price time series, which were originated from `www.yahoo.com`. To test the performance and effectiveness of our system, we added some "artificial" outliers to the time series. Regarding our system, the interface was built using Microsoft Excel with Visual Basic for Applications, and the underlying mining engine was implemented using Visual C++. As security datasets are processed one at a time, we assume that a dataset for a particular security will fit entirely into memory. This realistic assumption holds because in an extreme case, there are only about 26,000 prices for each security if day-to-day historical data are available for the past 100 years.

In the experiments, we tested our proposed system using various datasets (e.g., financial time series comprising of security prices for IBM Corporation, Boeing, Microsoft Corporation, etc.). The results were consistent. So, for lack of space, we only show the experimental results on the time series for IBM Corporation for the period 1962–2004. For this dataset (Fig. 4(a)), there are several important items that are worth special mentioning: A is a single outlier that is outside of the normal range of that interval in the time series. B is *not* an outlier; it is just a point where the stock price suddenly drops (due to some natural factors such as a change in economic situation or a stock split). C is a double-itemed outlier (where the two price items are very close to each other), which does not fall outside of the normal range of the dataset. D (which occurs not too far after the long-lived sudden price change) and E are single outliers that are within the normal range of the time series.

We first applied **DetectOutliersDM** to the IBM time series. Results in Fig. 4(b) show that our algorithm was able to successfully detect and remove outliers A and C, while leaving non-outlier B intact. An advantage of this algorithm is its effective removal of double-itemed (or multi-itemed) outliers. This stems from the fact that the algorithm relies on the number of neighbours of a given data item to determine an outlier. Having only a small number of neighbours would be a good indication of an outlier. However, the algorithm failed to identify outliers D and E, because these two outliers were within the normal range of the dataset and they had a large number of items in their surrounding neighbourhood; hence, the algorithm did not see them as outliers.

We then applied **DetectOutliersStat**, which calculates moving averages for each item. As shown in Fig. 4(c), our algorithm successfully detected and removed outliers A, D and E, while leaving non-outlier B intact. However, the algorithm failed to identify double-itemed outlier C. It is because this algorithm used statistical methods based on averages and standard deviations. When there were several outliers that are close to each other, they would influence each other's averages and standard deviation. This would lead to a lower ranking of the individual items in the time series, and hence failed to correctly identify outliers.

To summarise, the above experimental results show the strengths and weaknesses of DetectOutliersDM and DetectOutliersStat. Due to their varying nature and properties, the algorithms may not necessarily detect and remove the same outliers from the financial time series. Instead, they may detect different outliers caused by short-lived sudden price changes. For example, both algorithms were able to detect outlier A (a single outlier that is *outside* of the normal range) and leave non-outlier B intact. However, outlier C (a *multi-itemed* outlier) was only detected by DetectOutliersDM; outliers D and E (outliers that are *within* normal range) were only detected by DetectOutliersStat.

Observing the strengths and weaknesses of these two algorithms, we finally put the two together into our proposed system. Results in Fig. 4(d) indicate that our system comprising of both algorithms successfully detected outliers A, C, D and E while left non-outlier B intact. This shows the *effectiveness* of our system.

Next, let us then turn our attention to the *efficiency* issue. Experiments were conducted using the above IBM dataset on a single processor machine with 512 MB of operating memory. Results show that the execution times for both algorithms were short ($\approx$ 1 second) and approximately the same.

As the output from our system (the resulting/processed datasets) can be used as input for financial models used in forecasting future security prices or in predicting future market behaviour, we plan to conduct some experiments to study the improvement in the forecasting of future security prices and the computation of option prices.

4 Conclusions

We developed an efficient system to successfully detect outliers from financial time series. Our system consists of two phases: Phase I identifies the missing data, and replaces them with new prices that are consistent with their neighbouring prices; Phase II detects short-lived sudden price changes. We developed two algorithms for this second phase. Our data mining algorithm, called DetectOutliersDM, uses a distance-based approach to detect outliers (especially those items having insufficient neighbours and those multi-itemed outliers). Our statistical algorithm, called DetectOutliersStat, uses moving averages of each item to detect outliers (especially those outlying items that are within the normal range and those single-itemed outliers). While both of these two algorithms are effective in detecting most outliers, there exist some outliers that are detected by only one of the algorithms. We observed and understood the strengths and weaknesses of the two algorithms, and we put them together. Then, an item in the time series is considered as an outlier if it is detected by one of the algorithms. Consequently, more outliers can be effectively detected and removed. This, in turns, leads to cleaner time series (i.e., with less noise).

This paper shows a confluence of various disciplines—namely, data mining, statistics, and finance. It also shows an additional applicability of outlier detection techniques. To elaborate, many existing outlier detection algorithms are generally served as stand-alone tools for obtaining insight into data distribution. In contrast, our proposed outlier detection system can be served as a preprocessing step for other algorithms (e.g., financial models for forecasting future security prices, predicting future market behaviour, and/or pricing complex financial instruments such as derivatives).

Acknowledgement. This project is partially sponsored by Science and Engineering Research Canada (NSERC) and The University of Manitoba in the form of research grants.

References

1. Agrawal, R., Ghosh, S., Imielinski, T., Iyer, B., Swami, A.: An interval classifier for database mining applications. In: Proc. VLDB 1992. 560–573
2. Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., Verkamo, A.I.: Fast discovery of association rules. In: Advances in Knowledge Discovery and Data Mining (1996) ch. 12
3. Back, A.D., Weigend, A.S.: A first application of independent component analysis to extracting structure from stock returns. World Scientific - IJNS **8** (1997) 473–484
4. Ester, M., Kriegel, H.-P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proc. KDD 1996. 226–231
5. Jolliffe, I.: Principal Component Analysis, Springer-Verlag (1986)
6. Keogh, E., Lonardi, S., Chiu, B.Y.: Finding surprising patterns in a time series database in linear time and space. In: Proc. KDD 2002. 550–556
7. Knorr, E.M., Ng, R.T.: Algorithms for mining distance-based outliers in large datasets. In: Proc. VLDB 1998. 392–403
8. Lakshmanan, L.V.S., Leung, C.K.-S., Ng, R.T.: Dynamic mining of constrained frequent sets. ACM TODS **28** (2003) 337–389
9. Leung, C.K.-S.: Evaluation of data mining opportunities at Workers' Compensation Board. Research report, Workers' Compensation Board of BC, Canada (1998)
10. Leung, C.K.-S., Khan, Q.I., Hoque, T.: CanTree: a tree structure for efficient incremental mining of frequent patterns. In: Proc. ICDM 2005. 274–281
11. Omiecinski, E., Savasere, A.: Efficient mining of association rules in large dynamic databases. In: Proc. BNCOD 1998. 49–63

12. Park, J.: Modern portfolio theory and its application to hedge funds: Part II. MFA Reporter (Aug. 2001) 1–2, 4, 12–13
13. Schwertman, N.C., Owens, M.A., Adnan, R.: A simple more general boxplot method for identifying outliers. Elsevier - CSDA **47** (2004) 165–174
14. Srikumar, K., Bhasker, B., Tripathi, S.K.: MaxDomino: efficiently mining maximal sets. In: Proc. BNCOD 2003. 131–139
15. Utans, J.W., Holt, T., Refenes, A.N.: Principal components for modelling multi-currency portfolios. In: Proc. NNCM 1996. 359–368
16. Victoria-Feser, M.-P.: Robust portfolio selection. Working paper 2000.14, University of Geneva, Switzerland (2000)

Efficient Update of Data Warehouse Views with Generalised Referential Integrity Differential Files

Carson Kai-Sang Leung[1] and Wookey Lee[2]

[1] The University of Manitoba, Winnipeg, MB, Canada
kleung@cs.umanitoba.ca
[2] Sungkyul University, Anyang, Korea
wook@sungkyul.ac.kr

Abstract. Data warehouse (DW) views provide an efficient access to information integrated from source data. When changes are made to the source data, the corresponding views may be outdated. Thus, the maintenance of DW views is crucial for the currency of information. Recently, a method was proposed to use referential integrity differential files (RIDFs) to self-maintain DW views that contain select-project-joins over relations modelled in a star schema. However, it is not uncommon for applications to have relations that are modelled in other schemas such as a snowflake schema or a galaxy schema. In this paper, we generalise the concept of RIDFs; we propose a method that uses *generalised RIDFs* to self-maintain the DW views that contain joins over relations modelled in the star schema as well as non-star schemas. Our method computes new views by using only the old materialised views and files that keep the truly relevant tuples in the "delta". Consequently, it avoids accessing the underlying source data, and hence leads to efficient update of DW views.

Keywords: Data warehousing, view maintenance, referential integrity constraints, snowflake schema, galaxy schema, self-maintainability.

1 Introduction

A data warehouse (DW) is a subject-oriented, integrated, time-variant, and non-volatile collection of data organised in such a way that it supports the decision making process of management [8]. In general, DW views provide a fast access to integrated source data. As changes can be made to the source data, the corresponding views may be outdated. Thus, the maintenance of views is crucial for the currency of information. In other words, views need to be periodically refreshed so as to reflect those updates that have been made to the source data. In response to the changes to the source data, many existing DW views are refreshed by recomputing the contents from scratch (i.e., computing the new views from the updated source data), while some other views are incrementally maintained by accessing the source data. However, these approaches can be costly. Moreover, in many real-life situations, it is not uncommon that only a

D. Bell and J. Hong (Eds.): BNCOD 2006, LNCS 4042, pp. 199–211, 2006.

tiny fraction of some huge source data gets changed. The above approaches require an access to a huge amount of source data. Consequently, both CPU and I/O costs of these approaches can be extremely high. A better approach is to incrementally maintain views *without* accessing the source data. This calls for efficient view maintenance approaches.

Over the past decade, many approaches (e.g., [1-7, 9-13, 15-19]) have been proposed. However, some of these works focused mainly on conventional database views than DW views, and some did not fully exploit *referential integrity (RI) constraints* for view maintenance. For those that maintain DW views based on RI, they used auxiliary structures (e.g., auxiliary views [17], auxiliary relations [15], auxiliary data [9], complements [11]), which can be costly to build in some situations.

Recently, an efficient method, which incurred a lower cost, was proposed to exploit the RI constraints imposed on relations in the source data [13]. Specifically, the method used *referential integrity differential files (RIDFs)* to keep all and only those tuples that are relevant to the maintenance of views. By so doing, DW views can be self-maintained in the sense that the new views (reflecting the changes that were made to the source data) can be formed by using only the old materialised views, differential files (DFs—i.e., files containing the inserted or the deleted tuples), and RIDFs. In other words, the update of views avoided accessing the underlying databases. Such a method was designed for, and worked well on, self-maintaining views that contain a select-project-join (SPJ) over multiple relations modelled in a *star schema*. However, there exist many situations where relations are *not* modelled in the star schema (e.g., a snowflake schema in Example 1, a galaxy schema in Example 2).

Example 1. Consider a DW for shipment consists of the following tables:

- `Item (itemID, name, description)`;
- `Location (locCode, city, county, region)`;
- `Shipper (shipperID, locCode, shipperName)`
  `where locCode references Location`; and
- `Shipment (shipmentID, itemID, shipperID)`
  `where itemID references Item, shipperID references Shipper`.

Here, information about shippers and their locations is stored in two different dimension tables, namely `Shipper` and `Location`, due to normalisation. Relations/tables in this DW are modelled in a *snowflake schema*, where the fact table `Shipment` references dimension tables `Shipper` (which, in turn, references the dimension table `Location`) and `Item`. In this model, view $v_1 \equiv \pi_{\texttt{itemID,shipperName}} \sigma_{\texttt{city=Belfast}} (\texttt{Shipment} \bowtie \texttt{Shipper} \bowtie \texttt{Location})$, which finds IDs of items and names of shippers for those shippers located in Belfast, contains an SPJ over three relations modelled in a snowflake schema. □

Example 2. Let us add the following table to the above shipment DW:

- `Sales (invoiceID, itemID, locCode, price)`
  `where itemID references Item, locCode references Location`.

Then, in the resulting model, relations/tables are in a *galaxy schema* consisting of a collection of stars and snowflakes. Here, two fact tables `Sales` and `Shipment` share the dimension table `Item`. In this model, view $v_2 \equiv \pi_{\texttt{invoiceID,shipmentID}} \sigma_{\texttt{description=book}} (\texttt{Sales} \bowtie \texttt{Shipment} \bowtie \texttt{Item})$, which finds invoice IDs and shipment IDs for all books, contains an SPJ over three relations modelled in a galaxy schema. □

Given that there exist situations where relations can be modelled in a non-star schema, some natural questions are: How to handle these situations? Can we use RIDFs in these situations? Can RIDFs be helpful? Can we self-maintain the views using just the old materialised views, DFs and RIDFs? If not, what else is needed?

In this paper, we study these questions. Our ***key contribution*** is the development of a novel method, which exploits RI constraints and generalises the ideas of RIDFs, for self-maintaining DW views. Our method uses *generalised RIDFs (GRIDFs)*, which keep all and only those tuples that are (directly or indirectly) relevant to the updates of views. By so doing, the method efficiently self-maintains the views modelled in the star schema as well as non-star schemas (e.g., the snowflake schema, the galaxy schema). With our method, new views can be formed by using *only* (i) the old materialised views, (ii) DFs, (iii) RIDFs, and (iv) *GRIDFs*. In other words, the method avoids accessing any underlying databases to form the new views.

The outline of this paper is as follows. Section 2 gives background. Section 3 describes how we generalise the concepts of RIDFs for self-maintaining DW views involving joins over relations modelled in a snowflake or a galaxy schema. Section 4 discusses further generalisation and potential improvements of GRIDFs. Experimental results are given in Section 5. Finally, conclusions are presented in Section 6.

2 Background

In this section, we present some background materials about RI constraints and RIDFs, which are relevant to the rest of this paper.

2.1 Referential Integrity Constraints

A referential integrity (RI) constraint can be specified between relations in database and data warehousing environments; it is used to maintain consistency among tuples in the relations. Informally, the constraint states that a tuple r in a relation R (called the **referencing relation**) that refers to another relation S (called the **referenced relation**) must refer to an existing tuple s in S. More formally, the foreign key of R (denoted as $R.fk$) must "match" a candidate key of S (denoted as $S.ck$), that is, they must have the same domain and $R.fk = S.ck$.[1] Without loss of generality, we assume in this paper that all relations in the DW are "linked" by RI constraints.

Whenever there is a change to a relation in an underlying database, the corresponding views need to be updated to reflect the change. This can be done using either an immediate mode or a deferred mode. For the former, the views are

[1] A *candidate key* of a relation is a minimal set of attributes whose values uniquely identify each tuple in the relation. A *foreign key* is a set of attributes (in a referencing relation R) that either refers to a candidate key of the referenced relation S or is NULL.

refreshed immediately; for the latter, all the changes are first recorded in some *differential files (DFs)*, and the views are then updated periodically using these DFs. Whenever a tuple is inserted into, or deleted from, a referencing relation R or a referenced relation S, appropriate actions need to be taken as described below. (i) When a tuple r is **inserted into a referencing relation** R, a look-up in S is required to ensure the presence of a tuple $s \in S$ where $s.ck = r.fk$. If s is present, then r is inserted into R as well as the differential file $\mathit{\Delta} R$;[2] otherwise, RI is violated. (ii) When a tuple r is **deleted from a referencing relation** R, the tuple r is recorded in the differential file ∇R. (iii) When a tuple s is **inserted into a referenced relation** S, the tuple s is recorded in the differential file $\mathit{\Delta} S$. (iv) When a tuple s is **deleted from a referenced relation** S, a (reverse) look-up in R is required (for the default mode of "`on delete no action`") to ensure the absence of a tuple $r \in R$ satisfying $r.fk = s.ck$. If r is absent, then s is safely removed from S; otherwise (i.e., r exists in R), RI is violated and the deletion is rejected. It is interesting to note that the insertion into, or deletion from, one relation (R or S) does not affect another one.

2.2 Referential Integrity Differential Files

Let us consider view $v_3 \equiv \pi_{\texttt{itemID,city}} \sigma_{\texttt{price}>\texttt{50}}(\texttt{Sales} \bowtie \texttt{Item} \bowtie \texttt{Location})$, which finds item IDs and cities of those sales with price > \$50. This view contains an SPJ over three relations modelled in a *star schema*, where the fact table `Sales` references dimension tables `Item` and `Location`. This view can be expressed—in abstract terms—as $\pi_A \sigma_C(v_4)$, where $v_4 \equiv (F \bowtie D_1 \bowtie D_2)$ such that $F.fk_1$ references $D_1.ck$ and $F.fk_2$ references $D_2.ck$. In subsequent expressions, let us focus on how to efficiently update the join component because it dominates the SPJ operations.

When changes are made to the source data, a **naïve method** to update a DW view—whenever its underlying relations (e.g., F, D_1, D_2) of a view are updated—is to ignore the old view and to compute the new view from scratch (e.g., $v_4' = (F' \bowtie D_1' \bowtie D_2')$). However, this method can be very costly, especially when updates are made very frequently or when only a tiny fraction of underlying relations is updated.

It is well-known that an updated relation R' can be expressed as $R' = R - \nabla R \cup \mathit{\Delta} R$, where R is the old relation (e.g., F, D_1, or D_2), $\mathit{\Delta} R$ is its insertion, and ∇R is its deletion. So, an **improved method** is to obtain the new view v_4' from the old view $v_4 \equiv (F \bowtie D_1 \bowtie D_2)$, DFs (e.g., $\mathit{\Delta} F, \nabla D_1, \ldots$), and source relations (e.g., F, D_1, D_2), as follows:

$$\begin{aligned} v_4' &= (F - \nabla F \cup \mathit{\Delta} F) \bowtie (D_1 - \nabla D_1 \cup \mathit{\Delta} D_1) \bowtie (D_2 - \nabla D_2 \cup \mathit{\Delta} D_2) \\ &= (F \bowtie D_1 \bowtie D_2) - (F \bowtie D_1 \bowtie \nabla D_2) \cup (F \bowtie D_1 \bowtie \mathit{\Delta} D_2) \cup \ldots \\ &\quad - (\mathit{\Delta} F \bowtie \mathit{\Delta} D_1 \bowtie \nabla D_2) \cup (\mathit{\Delta} F \bowtie \mathit{\Delta} D_1 \bowtie \mathit{\Delta} D_2). \end{aligned} \tag{1}$$

[2] Since the views can be updated using the deferred mode, it is more precise to state the following. An insertion of a tuple r into R requires a look-up in the "current" referenced relation $(S - \nabla S \cup \mathit{\Delta} S)$. If there exists a tuple $s \in (S - \nabla S \cup \mathit{\Delta} S)$ such that $s.ck = r.fk$, then r is inserted into R as well as $\mathit{\Delta} R$.

Note that, among the $3^3 = 27$ terms in Equation (1), the first term $(F \bowtie D_1 \bowtie D_2)$ is the old view v_4. However, many of the remaining 26 terms (e.g., $(F \bowtie D_1 \bowtie \mathit{\Delta} D_2)$) involve source relations.

To avoid accessing the source relations and to efficiently self-maintain DW views, we proposed in BNCOD 2005 a **self-maintainable method with referential integrity differential files (RIDFs)** [13]. By exploiting the properties of RI constraints as well as the nature of the expression for view updates and by using RIDFs, Equation (1) can be simplified to become the following:

$$\begin{aligned} v_4' = v_4 &\cup (\mathit{\Delta} F \bowtie \mathit{RIDF}_F(D_1) \bowtie \mathit{RIDF}_F(D_2)) \\ &\cup (\mathit{\Delta} F \bowtie \mathit{RIDF}_F(D_1) \bowtie \mathit{\Delta} D_2) \cup (\mathit{\Delta} F \bowtie \mathit{\Delta} D_1 \bowtie \mathit{RIDF}_F(D_2)) \\ &\cup (\mathit{\Delta} F \bowtie \mathit{\Delta} D_1 \bowtie \mathit{\Delta} D_2) \ominus \nabla F \ominus \nabla D_1 \ominus \nabla D_2. \end{aligned} \tag{2}$$

Such a simplification is possible because of the following:

- The term $(F \bowtie D_1 \bowtie D_2)$ in Equation (1) represents the old view v_4.
- Any terms with $(F \bowtie \mathit{\Delta} D_1)$ give empty relations. Because of RI constraints, for all $f \in F$, there must exist $d \in D_1$ such that $d.ck = f.fk_1$. In other words, there does not exist a tuple $d' \in \mathit{\Delta} D_1$ satisfying $d'.ck = f.fk_1$. Similar comments apply for all terms with $(F \bowtie \mathit{\Delta} D_2)$.
- All the terms involving ∇F can be grouped together (denoted as $\ominus \nabla F$) as they basically represent the action that all the tuples containing $f \in \nabla F$ can be deleted. Similar comments apply for all terms involving ∇D_1 as well as ∇D_2.
- The term $(\mathit{\Delta} F \bowtie \mathit{\Delta} D_1 \bowtie \mathit{\Delta} D_2)$ involves only three differential files $(\mathit{\Delta} F, \mathit{\Delta} D_1$ and $\mathit{\Delta} D_2)$. In other words, no access to the source data is required.
- The remaining three terms—namely, $(\mathit{\Delta} F \bowtie \mathit{RIDF}_F(D_1) \bowtie \mathit{RIDF}_F(D_2))$, $(\mathit{\Delta} F \bowtie \mathit{RIDF}_F(D_1) \bowtie \mathit{\Delta} D_2)$ and $(\mathit{\Delta} F \bowtie \mathit{\Delta} D_1 \bowtie \mathit{RIDF}_F(D_2))$—all use RIDFs. Recall from Section 2.1 that when a tuple f is inserted into F, we check if there exists a tuple $d \in D_i$ such that $d.ck = f.fk$. If such d exists, the insertion is successful and f is then recorded in $\mathit{\Delta} F$. Given that the search and check has been performed, one can record the tuple f in a file called *RIDF*. By so doing, the RIDF contains all those tuples (d) that are related to the tuples in $\mathit{\Delta} F$. In other words, the RIDF contains all and only those tuples that could be joined with $\mathit{\Delta} F$ in the term $(\mathit{\Delta} F \bowtie D_i)$. Therefore, with the RIDF, the term $(\mathit{\Delta} F \bowtie D_i)$ can be rewritten as $(\mathit{\Delta} F \bowtie \mathit{RIDF}_F(D_i))$, which no longer requires an access to the source data.

While more details can be found in our BNCOD 2005 paper [13], it is important to note that the self-maintenance of DW views with RIDFs was designed and worked well on joins over relations that are modelled in a *star schema*. In the current BNCOD 2006 paper, we extend and generalise the RIDFs to handle situations where relations are modelled in a *non-star schema* (as well as in a star schema).

3 Our New Method: Self-maintenance of DW Views with Generalised RIDFs (GRIDFs)

In this section, we start describing our *new* method that *uses our generalised RIDFs to self-maintain DW views.* Like the method with RIDFs, this new one with GRIDFs also exploits referential integrity. However, unlike the method with RIDFs, the method with GRIDFs can be applicable for updating views involving relations that are modelled in non-star schemas (as well as in a star schema).

Here, we start with the base case where views involve an SPJ over three relations; then, in Section 4, we give the general case for k relations. For three relations R, S and T, there are various ways in which these relations reference others. For example, R may reference both S and T (e.g., views v_3, v_4 shown above). Alternatively, R may also reference S, which in turn references T (e.g., view v_1 in Example 1). As the third way, it may be the case where both R and S reference T (e.g., view v_2 in Example 2).

3.1 "Forward-Linked" Generalised RIDFs (fGRIDFs)

Let us consider view $v_1 \equiv \pi_{\texttt{itemID,shipperName}}\sigma_{\texttt{city=Belfast}}(\texttt{Shipment} \bowtie \texttt{Shipper} \bowtie \texttt{Location})$ in Example 1. How to self-maintain v_1? Or, a more general question is: How to compute a new view of the form $\pi_A\sigma_C(v_5)$, where $v_5 \equiv (F \bowtie D_1 \bowtie D_2)$ such that $F.fk$ references $D_1.ck$ and $D_1.fk$ references $D_2.ck$?

Learned from Section 2.2, we know the disadvantages of using the naïve method (i.e., start from scratch) and the improved method (i.e., using the old view, DFs, and source relations). Specifically, the former can be very costly, whereas the latter requires accesses to source relations. So, we exploit the RI constraints and obtain the following expression:

$$\begin{aligned} v_5' = v_5 &\cup (\Delta F \bowtie RIDF_F(D_1) \bowtie D_2) \cup (\Delta F \bowtie \Delta D_1 \bowtie RIDF_{D_1}(D_2)) \\ &\cup (\Delta F \bowtie \Delta D_1 \bowtie \Delta D_2) \ominus \nabla F \ominus \nabla D_1 \ominus \nabla D_2. \end{aligned} \tag{3}$$

Observed from Equation (3), we can easily spot that one of the terms (i.e., the term $(\Delta F \bowtie RIDF_F(D_1) \bowtie D_2)$) still involves D_2 (the underlying database). Hence, even if we could use RIDFs, the new view v_5' could not be computed without accessing the source data.

On the surface, it may appear that this is the best we could do. However, a careful study reveals that we could do better. Specifically, are we required to access D_2? Do we need to join with the entire D_2? The answer in each case is no. We can avoid accessing D_2 by using some "files" similar to RIDFs. The "files" store only truly relevant tuples. To elaborate, we extend and generalise the concept of RIDFs, and we come up with *generalised RIDFs (GRIDFs).* Specifically, our idea can be described as follows. When a tuple f is inserted into F, we check if there exists a tuple $d_1 \in D_1$ such that $d_1.ck = f.fk$. If such d_1 exists, the insertion is successful. Then, (i) f is recorded in ΔF and (ii) d_1 is recorded in $RIDF_F(D_1)$. Up to this point, the procedure sounds familiar as it is the same as the creation of the RIDF. However, next step is different: The insertion of d_1 into $RIDF_F(D_1)$ then triggers a look-up of $d_2 \in D_2$ such that $d_2.ck = d_1.fk$. Due to the RI constraint, we know that there exists such $d_2 \in D_2$. So, d_2 is then inserted into a "file" called the *"forward-linked" generalised RIDF (fGRIDF).* See the definition below.

Definition 1 ("Forward-linked" generalised referential integrity differential file (fGRIDF)). *Let (i) an SPJ view $\pi_A \sigma_C(F \bowtie D_1 \bowtie D_2)$ be created in terms of three relations F, D_1 and D_2; (ii) a RI constraint be imposed on F and D_1 such that $F.fk = D_1.ck$, where $F.fk$ denotes the foreign key of the referencing relation F and $D_1.ck$ denotes a candidate key of the referenced relation D_1; and (iii) a RI constraint be imposed on D_1 and D_2 such that $D_1.fk = D_2.ck$, where $D_1.fk$ denotes the foreign key of the referencing relation D_1 and $D_2.ck$ denotes a candidate key of the referenced relation D_2. (Note that D_1 plays two different roles: It is a referenced relation with respect to F, but a referencing relation with respect to D_2.) Then, when a tuple f is successfully inserted into F (i.e., f is put in ΔF), the corresponding tuple d_1 (where $d_1.ck = f.fk$) is then inserted into D_1. This triggers the insertion of d_2 into a* **"forward-linked" generalised referential integrity differential file***, denoted as $fGRIDF_{D_1}(D_2)$, which keeps all and only those tuples (in D_2) that are truly relevant to the update of the view. Precisely, for each tuple $d_1 \in RIDF_F(D_1)$, its corresponding $d_2 \in D_2$ (such that $d_2.ck = d_1.fk$) is kept in $fGRIDF_{D_1}(D_2)$.*

There are some nice properties of $fGRIDF_{D_1}(D_2)$. First, $fGRIDF_{D_1}(D_2)$ keeps all and only those tuples (in D_2) that are truly relevant to the join $(\Delta F \bowtie RIDF_F(D_1) \bowtie D_2)$. Thus, the number of tuples in $fGRIDF_{D_1}(D_2)$ is bounded above by the number of tuples in D_2 (i.e., $|fGRIDF_{D_1}(D_2)| \leq |D_2|$). Second, for each candidate key of D_2, the number of tuples in $fGRIDF_{D_1}(D_2)$ is bounded above by the number of tuples in $RIDF_F(D_1)$. This is due to RI constraints. More specifically, because $d_1.fk = d_2.ck$, many $d_1 \in RIDF_F(D_1)$ can reference one d_2 (but each d_1 can only reference one d_2). Hence, if D_2 only has one candidate key (which is quite common for dimension tables), then $|fGRIDF_{D_1}(D_2)| \leq |RIDF_F(D_1)|$. Third, since $|RIDF_F(D_1)| \leq |\Delta F|$, we have $|fGRIDF_{D_1}(D_2)| \leq |RIDF_F(D_1)| \leq |\Delta F|$. Therefore, by exploiting properties of RI constraints and using $fGRIDF_{D_1}(D_2)$, Equation (3) can be simplified to become the following (i.e., the new view can be computed as follows):

$$\begin{aligned} v_5' = v_5 &\cup (\Delta F \bowtie RIDF_F(D_1) \bowtie fGRIDF_{D_1}(D_2)) \\ &\cup (\Delta F \bowtie \Delta D_1 \bowtie RIDF_{D_1}(D_2)) \\ &\cup (\Delta F \bowtie \Delta D_1 \bowtie \Delta D_2) \ominus \nabla F \ominus \nabla D_1 \ominus \nabla D_2. \end{aligned} \tag{4}$$

It is important to note that, with this self-maintainable method with GRIDFs, we no longer require accesses to the source data. The new view v_5' can be computed using only (i) the old view v_5, (ii) DFs, (iii) RIDFs, and (iv) GRIDFs (i.e., fGRIDFs). See the following example.

Example 3. Let us consider the self-maintenance of view $v_6 \equiv$ (`Shipment` $\bowtie$ `Shipper` $\bowtie$ `Location`), which is modelled in a snowflake schema. (Note that view v_1 in Example 1 can be expressed as $\pi_{\texttt{itemID,shipperName}}\sigma_{\texttt{city=Belfast}}(v_6)$.) When the underlying databases are updated, the new view v_6' can be computed as follows:

$$\begin{aligned} v_6' = v_6 &\cup (\Delta\texttt{Shipment} \bowtie RIDF_{\texttt{Shipment}}(\texttt{Shipper}) \bowtie fGRIDF_{\texttt{Shipper}}(\texttt{Location})) \\ &\cup (\Delta\texttt{Shipment} \bowtie \Delta\texttt{Shipper} \bowtie RIDF_{\texttt{Shipper}}(\texttt{Location})) \\ &\cup (\Delta\texttt{Shipment} \bowtie \Delta\texttt{Shipper} \bowtie \Delta\texttt{Location}) \\ &\ominus \nabla\texttt{Shipment} \ominus \nabla\texttt{Shipper} \ominus \nabla\texttt{Location}. \end{aligned}$$ □

3.2 "Backward-Linked" Generalised RIDFs (bGRIDFs)

Next, let us consider view $v_2 \equiv \pi_{\texttt{invoiceID,shipmentID}}\,\sigma_{\texttt{description=book}}(\texttt{Sales} \bowtie \texttt{Shipment} \bowtie \texttt{Item})$ in Example 2. How to self-maintain v_2? Or, a more general question is: How to compute a new view of the form $\pi_A\sigma_C(v_7)$, where $v_7 \equiv (F_1 \bowtie F_2 \bowtie D)$ such that $F_1.fk$ references $D.ck_1$ and $F_2.fk$ references $D.ck_2$?

By applying the method with RIDFs [13] (or applying our proposed method with fGRIDFs described in Section 3.1), we obtain the following expression:

$$\begin{aligned} v_7' = v_7 &\cup (F_1 \bowtie \Delta F_2 \bowtie RIDF_{F_2}(D)) \cup (\Delta F_1 \bowtie F_2 \bowtie RIDF_{F_1}(D)) \\ &\cup (\Delta F_1 \bowtie \Delta F_2 \bowtie [RIDF_{F_1}(D) \cap RIDF_{F_2}(D)]) \\ &\cup (\Delta F_1 \bowtie \Delta F_2 \bowtie \Delta D) \ominus \nabla F_1 \ominus \nabla F_2 \ominus \nabla D. \end{aligned} \tag{5}$$

Observe that two of the terms (i.e., the second and the third terms) still involve those underlying databases (i.e., F_1 and F_2). So, what we need is another type of generalised RIDF, which we call the *"backward-linked" generalised RIDF (bGRIDF)*. Specifically, the idea can be described as follows. When a tuple f_i is inserted into F_i, we check if there exists a tuple $d \in D$ such that $d.ck = f_i.fk$. If such d exists, the insertion is successful. Then, (i) f_i is recorded in ΔF_i and (ii) d is recorded in $RIDF_{F_i}(D)$. Again, the extra/new step is as follows: The insertion of d into $RIDF_{F_i}(D)$ triggers a reverse look-up of $f_j \in F_j$ (where $j \neq i$) such that $f_j.fk = d.ck$. Due to the RI constraint, we know that there could be no such $f_j \in F_j$. However, if one exists, it is then inserted into a "backward-linked" generalised RIDF (i.e., $bGRIDF_D(F_j)$). See the definition below.

Definition 2 ("Backward-linked" generalised referential integrity differential file (bGRIDF)). *Let (i) an SPJ view $\pi_A\sigma_C(F_1 \bowtie F_2 \bowtie D)$ be created in terms of three relations F_1, F_2 and D; and (ii) a RI constraint be imposed on F_i and D such that $F_i.fk = D.ck$ where $F_i.fk$ denotes the foreign key of the referencing relation F_i (for $i = 1, 2$) and $D.ck$ denotes a candidate key of the referenced relation D. Then, when a tuple f_i is successfully inserted into F_i (i.e., f_i is put in ΔF_i), the corresponding tuple d (where $d.ck = f_i.fk$) is then inserted into D. This triggers the insertion of f_j (where $j \neq i$) into a* **"backward-linked" generalised referential integrity differential file**, *denoted as $bGRIDF_D(F_j)$, which keeps all and only those tuples (in F_j) that are truly relevant to the update of the view. Precisely, for each tuple*

$d \in RIDF_{F_i}(D)$, its corresponding $f_j \in F_j$ (such that $f_j.fk = d.ck$) is kept in $bGRIDF_D(F_j)$.

A nice property of the $bGRIDF_D(F_j)$ is that it keeps all and only those tuples (in F_j) that are truly relevant to the join ($\varDelta F_i \bowtie F_j \bowtie RIDF_{F_i}(D)$). Thus, the number of tuples in $bGRIDF_D(F_j)$ is bounded above by the number of tuples in F_j (i.e., $|bGRIDF_D(F_j)| \leq |F_j|$). Another property is that, for a given $d \in RIDF_{F_i}(D)$, there could be no $f_j \in F_j$ (where $f_j.fk = d.ck$) referencing it. This potentially reduces the size of $bGRIDF_D(F_j)$. Therefore, by exploiting properties of RI constraints and using $bGRIDF_D(F_i)$, Equation (5) can be simplified to become the following:

$$\begin{aligned} v_7' = v_7 &\cup (bGRIDF_D(F_1) \bowtie \varDelta F_2 \bowtie RIDF_{F_2}(D)) \\ &\cup (\varDelta F_1 \bowtie bGRIDF_D(F_2) \bowtie RIDF_{F_1}(D)) \\ &\cup (\varDelta F_1 \bowtie \varDelta F_2 \bowtie [RIDF_{F_1}(D) \cap RIDF_{F_2}(D)]) \\ &\cup (\varDelta F_1 \bowtie \varDelta F_2 \bowtie \varDelta D) \ominus \nabla F_1 \ominus \nabla F_2 \ominus \nabla D. \end{aligned} \tag{6}$$

It is important to note that, with this self-maintainable method with GRIDFs, we no longer require accesses to the source data. The new view v_7' can be computed using (i) the old view v_7, (ii) DFs, (iii) RIDFs, and (iv) GRIDFs (e.g., bGRIDFs). See the following example.

Example 4. Let us consider the self-maintenance of view $v_8 \equiv (\texttt{Sales} \bowtie \texttt{Shipment} \bowtie \texttt{Item})$, which is modelled in a galaxy schema. (Note that view v_2 in Example 2 can be expressed as $\pi_{\texttt{invoiceID,shipmentID}}\sigma_{\texttt{description=book}}(v_8)$.) When the underlying databases are updated, the new view v_8' can be computed as follows:

$$\begin{aligned} v_8' = v_8 &\cup (bGRIDF_{\texttt{Item}}(\texttt{Sales}) \bowtie \varDelta\texttt{Shipment} \bowtie RIDF_{\texttt{Shipment}}(\texttt{Item})) \\ &\cup (\varDelta\texttt{Sales} \bowtie bGRIDF_{\texttt{Item}}(\texttt{Shipment}) \bowtie RIDF_{\texttt{Sales}}(\texttt{Item})) \\ &\cup (\varDelta\texttt{Sales} \bowtie \varDelta\texttt{Shipment} \bowtie [RIDF_{\texttt{Sales}}(\texttt{Item}) \cap RIDF_{\texttt{Shipment}}(\texttt{Item})]) \\ &\cup (\varDelta\texttt{Sales} \bowtie \varDelta\texttt{Shipment} \bowtie \varDelta\texttt{Item}) \ominus \nabla\texttt{Sales} \ominus \nabla\texttt{Shipment} \ominus \nabla\texttt{Item}. \qquad \Box \end{aligned}$$

4 Discussion: Generalisation to Multiple Relations

So far, we have described and explained our proposed efficient method with GRIDFs for self-maintaining DW views involve an SPJ over three relations: (i) the use of fGRIDFs for an SPJ over a chain of one fact table and two dimension tables (Section 3.1), and (ii) the use of bGRIDFs for an SPJ over two fact tables that share a dimension table (Section 3.2). As expected, our method is not confined to just three relations. It can be further generalised to handle multiple relations by exploiting RI constraints. For example, a new view containing an SPJ over a chain of a fact table F and k levels of dimension tables $D_1, \ldots, D_k$ can be computed using $2k + 3$ terms as follows:

$$\begin{aligned} v' &= F' \bowtie D_1' \bowtie \cdots \bowtie D_k' \\ &= v \ \cup (\varDelta F \bowtie RIDF_F(R_1) \bowtie fGRIDF_{R_1}(R_2) \bowtie \cdots \bowtie fGRIDF_{R_1}(R_k)) \end{aligned}$$

$$
\begin{aligned}
&\cup \; [\bigcup_{j=2}^{k-1} (\Delta F \bowtie \Delta R_1 \bowtie \cdots \bowtie \Delta R_{j-1} \bowtie RIDF_{R_{j-1}}(R_j) \\
&\qquad\qquad \bowtie fGRIDF_{R_j}(R_{j+1}) \bowtie \cdots \bowtie fGRIDF_{R_j}(R_k))] \\
&\cup \; (\Delta F \bowtie \Delta R_1 \bowtie \cdots \bowtie \Delta R_{k-1} \bowtie RIDF_{R_{k-1}}(R_k)) \\
&\cup \; (\Delta F \bowtie \Delta R_1 \bowtie \cdots \bowtie \Delta R_k) \ominus \nabla F \ominus \nabla R_1 \cdots \ominus \nabla R_k. \qquad (7)
\end{aligned}
$$

Similarly, a new view containing an SPJ over k fact tables $F_1, \ldots, F_k$ that share a dimension table D can be computed using $2^k + k + 2$ terms as follows:

$$
\begin{aligned}
v' &= F_1' \bowtie \cdots F_k' \bowtie D' \\
&= v \;\cup \Big(\bigcup \Delta F_i \bowtie bGRIDF_D(F_j) \bowtie [\cap_i RIDF_{F_i}(D)]\Big) \\
&\quad \cup \; (\Delta F_1 \bowtie \cdots \bowtie \Delta F_k \bowtie \Delta D) \ominus \nabla F_1 \cdots \ominus \nabla F_k \ominus \nabla D, \qquad (8)
\end{aligned}
$$

where $1 \leq i, j \leq k$. With this generalisation, one would be able to efficiently compute the new DW views that contain an SPJ over different numbers of relations modelled in various schemas (e.g., star, snowflake, or galaxy schemas). One does not need to access the underlying databases during the update. All it needs is the old view, DFs, RIDFs, and our proposed GRIDFs (i.e., fGRIDFs and/or bGRIDFs).

Our proposed GRIDFs can further be improved by keeping only relevant *attributes* of the relevant tuples. Any attributes that do not contribute to the update of DW views can be discarded. Moreover, any tuples that do not contribute to the selection operator (of the SPJ) can also be discarded.

5 Experimental Results

We ran several experiments on various DWs. The results cited below are based on a DW that consists of a fact table (with 6,000,000 tuples) and multiple dimension tables (each with 800,000 tuples) that are modelled in a snowflake schema. In the experiments, we compared the results of the following four implemented methods:

- The **naïve method**, which recomputes new views from scratch.
- The **improved method**, which uses *old views, DFs*, and *source relations* to update the views.
- The self-maintainable **method with RIDFs** [13], which uses only old views, DFs, and *RIDFs.*
- Our efficient self-maintainable **method with GRIDFs**, which uses only old views, DFs, RIDFs, and *GRIDFs.* This method avoids accessing source relations even for those that are modelled in a non-star schema (e.g., the snowflake or galaxy schema).

In the first experiment, we fixed the number of dimension tables to 2 (i.e., the view $(F \bowtie D_1 \bowtie D_2)$ that contains an SPJ over a fact table F and two dimension tables D_1 & D_2. We varied the percentage of tuples being updated/changed

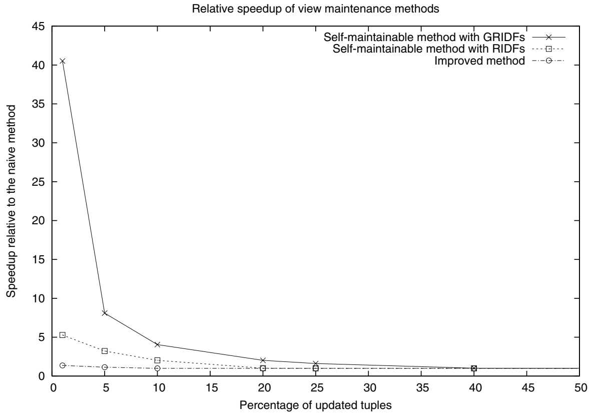


Fig. 1. Relative speedup of view maintenance methods

from 1% to 50%. The x-axis of Fig. 1 shows the percentage of updated tuples; the y-axis shows the speedup of the improved method, the method with RIDFs, and our self-maintainable method with GRIDFs against the naïve method. As observed from Fig. 1, the lower the percentage of updated tuples, the higher is the benefit of using our method. For example, the speedup of our method is above 40 times when 1% of tuples are updated. A much higher speedup is expected when the percentage of updated tuples is lower (e.g., 0.1%). Note that a low percentage of updated tuples is not uncommon. In many real-life applications, DW views need to be refreshed frequently (which usually leads to a low percentage of tuples being updated between each refresh) so as to facilitate accurate decision making.

While Fig. 1 shows the relative speedup, the table below gives some samples of the total runtime (i.e., both CPU and I/O time) for updating view.

% updated tuples	Naïve	Improved	RIDFs	GRIDFs
1%	714 mins	520 mins	135 mins	17.6 mins
10%	714 mins	621 mins	222 mins	88.1 mins

Note that our proposed method with GRIDFs requires a much shorter runtime than the other three methods. The reason is that our method uses GRIDFs; it does not need to access source relations. In contrast, the method with RIDFs, which uses DFs and RIDFs, needs to access the source relation D_2. The improved method, which uses DFs but not RIDFs, needs to access more source relations (both D_1 and D_2).

Next, we varied the number of dimension tables. The results show that increasing the number of dimension tables increases the speedup of our method and increases the runtime gaps. Note that when there are k dimension tables, the improved method and the method with RIDFs need to access k and $(k-1)$ source relations respectively. In contrast, our proposed method with GRIDFs does not need to access any source relations.

Then, let us count the numbers of tuples in the source relations, the "delta", RIDFs, and GRIDFs. It was observed that the number of tuples needed to be stored in a GRIDF is bounded above by the numbers of tuples in its corresponding source relations, RIDFs, and "delta" (e.g., $|fGRIDF_F(D_2)| \leq \min\{|D_2|, |RIDF_F(D_1)|, |\Delta F|\}$).

To summarise, the experimental results show the effectiveness of our proposed self-maintainable method with GRIDFs. Since the results on various DWs were consistent (and for lack of space), we do not show all the results here. For more details, please refer to our technical report [14].

6 Conclusions

Data warehouse (DW) views provide an efficient access to integrated data. As changes are made to the source data, the corresponding views may be outdated. Hence, the maintenance of views is crucial for the currency of information. In this paper, we proposed a novel method to efficiently self-maintain the DW views that contain a select-project-join (SPJ) over multiple relations. Specifically, we exploit the RI constraints imposed on the relations in the source data, and generalise the referential integrity differential files (RIDFs). The *generalised RIDFs (GRIDFs)*, proposed in this paper, keep the truly relevant tuples in the "delta"; they avoid accessing the underlying databases. Consequently, our method can update DW views by using only the old views, differential files (e.g., the insertion file ΔR and the deletion file ∇R), RIDFs, and *GRIDFs*. The method is applicable to the efficient self-maintenance of views that contain an SPJ over relations modelled in various schemas in data warehousing environments.

Acknowledgements. This project is partially sponsored by Science and Engineering Research Canada (NSERC) and The University of Manitoba, as well as Korea Science and Engineering Foundation (KOSEF) through Advanced Information Technology Research Centre (AITrc), in the form of research grants.

References

1. Blakeley, J.A., Larson, P.-Å., Tompa, F.W.: Efficiently updating materialized views. In: Proc. SIGMOD 1986. 61–71
2. Bruckner, R.M., Tjoa, A.M.: Managing time consistency for active data warehouse environments. In: Proc. DaWaK 2001. 254–263
3. Engström, H., Lings, B.: Evaluating maintenance policies for externally materialised multi-source views. In: Proc. BNCOD 2003. 140–156
4. Fišer, B., Onan, U., Elsayed, I., Brezany, P., Tjoa, A.M.: On-line analytical processing on large databases managed by computational grids. In: Proc. DEXA Workshop (GLOBE) 2004. 556–560
5. Griffin, T., Libkin, L., Trickey, H.: An improved algorithm for the incremental recomputation of active relational expressions. IEEE TKDE **9** (1997) 508–511
6. Gupta, H., Mumick, I.S.: Selection of views to materialize in a data warehouse. IEEE TKDE **17** (2005) 24–43

7. Hyun, N.: Multiple-view self-maintenance in data warehousing environments. In: Proc. VLDB 1997. 26–35
8. Inmon, W.H.: Building the Data Warehouse. John Wiley & Sons (1996)
9. Khan, S., Mott, P.: LeedsCQ: a scalable continual queries system. In: Proc. DEXA 2002. 607–617
10. Kotidis, Y., Roussopoulos, N.: A case for dynamic view management. ACM TODS **26** (2001) 388–423
11. Laurent, D., Lechtenbörger, J., Spyratos, N., Vossen, G.: Monotonic complements for independent data warehouses. VLDB Journal **10** (2001) 295–315
12. Lee, W.: On the independence of data warehouse from databases in maintaining join views. In: Proc. DaWaK 1999. 86–95
13. Leung, C.K.-S., Lee, W.: Exploitation of referential integrity constraints for efficient update of data warehouse views. In: Proc. BNCOD 2005. 98–110
14. Leung, C.K.-S., Lee, W.: GRIDFs: generalised referential integrity differential files for maintaining data warehouse views. Technical report TR 06/02, Department of Computer Science, The University of Manitoba, Canada (2006)
15. Mohania, M., Kambayashi, Y.: Making aggregate views self-maintainable. DKE **32** (2000) 87–109
16. Qian, X., Wiederhold, G.: Incremental recomputation of active relational expressions. IEEE TKDE **3** (1991) 337–341
17. Quass, D., Gupta, A., Mumick, I., Widom, J.: Making views self-maintainable for data warehousing. In: Proc. PDIS 1996. 158–169
18. Theodoratos, D., Xu, W.: Constructing search spaces for materialized view selection. In: Proc. DOLAP 2004. 112–121
19. Zhuge, Y., Garcia-Molina, H., Hammer, J., Widom, J.: View maintenance in a warehousing environment. In: Proc. SIGMOD 1995. 316–327

SAGA: A Combination of Genetic and Simulated Annealing Algorithms for Physical Data Warehouse Design

Ladjel Bellatreche[1], Kamel Boukhalfa[2], and Hassan Ismail Abdalla[3]

[1] LISI/ENSMA - Poitiers University France
bellatre@ensma.fr
[2] Laghouat University - Algeria
k.boukhalfa@mail.lagh-univ.dz
[3] Prince Sultan University - Saudi Arabia
habdalla@cis.psu.edu.sa

Abstract. Data partitioning is one of the physical data warehouse design techniques that accelerates OLAP queries and facilitates the warehouse manageability. To partition a relational warehouse, the best way consists in fragmenting dimension tables and then using their fragmentation schemas to partition the fact table. This type of fragmentation may dramatically increase the number of fragments of the fact table and makes their maintenance very costly. However, the search space for selecting an optimal fragmentation schema in the data warehouse context may be exponentially large. In this paper, the horizontal fragmentation selection problem is formalised as an optimisation problem with a maintenance constraint representing the number of fragments that the data warehouse administrator may manage. To deal with this problem, we present, SAGA, a hybrid method combining a genetic and a simulated annealing algorithms. We conduct several experimental studies using the APB-1 release II benchmark in order to validate our proposed algorithms.

1 Introduction

A data warehouse stores large amounts of consolidated and historical data. It is especially designed to support complex business decision queries. A relational data warehouse is usually modelled using a star schema (or a snow flake schema) which is characterised by one or more *very large* fact tables and a number of much smaller dimension tables. On the top of this schema, star queries are executed. The main characteristic of these queries is that they impose restrictions on the dimension values that are used for selecting specific facts; these facts are further grouped and aggregated according to the user demands. The major bottleneck in evaluating such queries has been the join of a large fact table with the surrounding dimension tables [8].

The horizontal data partitioning is an important aspect of physical database design [7,6]. It has a significant impact on performance of OLAP queries and

D. Bell and J. Hong (Eds.): BNCOD 2006, LNCS 4042, pp. 212–219, 2006.

manageability of the data warehouse. In context of relational warehouses, horizontal partitioning allows tables, indexes and materialised views to be partitioned into *disjoint sets* of rows that are physically stored and accessed separately [7]. Contrary to materialised views and indexes, data partitioning does not replicate data, thereby reducing space requirements and minimising the update overhead [6]. It can also be combined with others optimisation structures like indexes, materialised views, and parallel processing [8]. Several work and commercial systems show its utility and impact in optimising OLAP queries [7, **?**]. But *few studies have formalised the problem of selecting a horizontal partitioning schema to speed up a set of queries and proposed selection algorithms.*

In [1], we showed that the best way to partition a relational warehouse is to decompose the fact table based on the fragmentation schemas of dimension tables. Concretely, we (1) partition some/all dimension tables using their simple selection predicates, and then (2) partition using the fragmentation schemas of the fragmented dimension tables. The number of horizontal fragments of the fact table (denoted by N) generated by this partitioning procedure is given by: $\mathbf{N = \prod_{i=1}^{g} m_i}$, where m_i and g are the number of fragments of the dimension table D_i, and the number of dimension tables participating in the fragmentation process. This number may be very large [1]. Consequently, instead to manage one star schema, the data warehouse administrator (DWA) will manage N sub star schemas. It will be very difficult for him to *maintain* all these sub-star schemas. In [1], we developed a genetic algorithm to partition a data warehouse modelled by a star schema. *The main limitation of this algorithm is that the used fitness function did not take into account the maintenance constraint.* In this paper, we propose an approach, called, SAGA that combines genetic and simulated annealing algorithms. It penalises the fragmentation schemas that violate the maintenance constraint. A penalty function is incorporated in the fitness function used by both algorithms (genetic and simulated annealing).

This paper is divided in five sections: Section 2 formalises the fragmentation selection problem in data warehouses modelled using star schemas. Section 3 presents the components of the SAGA, where the GA and the SA are described in details. Section 4 gives the experimental results using benchmark APB-1 release II benchmark. The Section 5 concludes the paper by summarising the main results and suggesting future work.

2 Horizontal Partitioning Selection Problem

A formulation of the horizontal partitioning problem is the following: given (1) a set of dimension tables $D = \{D_1, D_2, ..., D_d\}$ and a fact table F, (2) a set of OLAP queries $Q = \{Q_1, Q_2, ..., Q_m\}$, where each query Q_i $(1 \leq i \leq m)$ has an access frequency, and (3) a *threshold* (W fixed by the DWA) representing the maximal number of fragments that he can maintain. The satisfaction of this constraint avoids an explosion of the number of the fact fragments. The horizontal partitioning problem consists in determining a set of dimension tables $D' \subseteq D$ to be partitioned and using their fragmentation schemas to partition the fact table F into a set of horizontal fragments (called fact fragments) such that: the *sum of the query cost when executed on top of the partitioned star schema is minimised, and the maintenance constraint is satisfied* ($N \leq W$).

Our approach, $SAGA$, uses first a genetic algorithm (GA) [3] and then a simulated annealing algorithm (SA) [5]. GAs have been used in the database physical design [9, 4]. Given a well-defined search space they apply three different genetic search operations, namely, *selection*, *crossover*, and *mutation*, to transform an initial population of chromosomes, with the objective to improve their quality. Note that GA is not guaranteed to find the global optima. One of the main reasons is the problem of premature convergence of the GAs. To avoid this problem, we use a SA. It is an iterative improvement scheme with the hill-climbing ability, which allows it to reject inferior local solutions and find more globally *near optimal solutions*. It starts with a solution candidate obtained by the GA, then repeatedly attempts to find a better solution by moving to a neighbour with higher fitness, until it finds a solution, where none of its neighbours has a higher fitness. To avoid getting trapped in poor local optima, SA allows occasionally an uphill moves to solutions with lower fitness by using a temperature parameter to control the acceptance of the moves (with a probability). These two algorithms will be described in details in next sections.

3 Th Description of SAGA

3.1 Genetic Algorithm Setting

Each chromosome (fragmentation schema) can be represented by a multidimensional array [1], where each cell i represents a a sub domain of the whole domain of the fragmentation attribute A_i. This multidimensional array representation may be used to generate all possible fragmentation schemas (an exhaustive search), given by: $2^{(\sum_{i=1}^{K} n_i)}$, where K and n_i represent the number of fragmentation attributes and the number of sub domains of fragmentation attribute A_i. Selection, crossover and mutation operations of our GA are detailled in [1].

Constraint Handing: Penalty Ranking. Each fragmentation schema FS_i generated by our GA is evaluated using a fitness function representing by a cost model which calculates the sum of page accesses (inputs/outputs) incurred by each query executed on the fragmented data warehouse. Our cost model supposes that all dimension tables are stored in the main memory. Let $D^{sel} = \{D_1^{sel}, ..., D_k^{sel}\}$ be the set of dimension tables having selection predicates, where each selection predicate p_j (defined on a dimension table D_i) has a selectivity factor denoted by $Sel_{D_i}^{p_j}$ ($Sel_{D_i}^{p_j} \in [0, 1]$). For each predicate p_j, we define its selectivity factor on the fact table, denoted by $Sel_F^{p_j}$ ($Sel_{D_i}^{p_j} \neq Sel_F^{p_j}$) [1]. In this study, the selectivity factors are chosen using an uniform distribution (UD) and a non uniform distribution (NUD).

To compute our fitness function, each query Q_k is executed over each partitioned schema FS_i $\{S_1, S_2, ..., S_{N_i}\}$. In order to identify relevant sub star schemas used by this query, we introduce a boolean variable (denoted by

$V(Q_k, S_i))$ defined as follows: $V(Q_k, S_i) = 1$ if the sub star schema S_i is needed for Q_k, 0 otherwise. Therefore, the cost in terms of number of inputs outputs needed by this query is $IOC(Q_k, FS_j) = (\sum_{j=1}^{N_i} V(Q_k, FS_j) \prod_{i=1}^{M_j} Sel_F^{p_i} \times \left\lceil \frac{||F|| \times L)}{PS} \right\rceil)$, where, M_j, F, L and PS represent the number of selection predicates defining the fact fragment of the sub star schema S_j, the cardinality of the fact table (number of tuples) F, the width, in bytes, of a tuple of a table F and the page size of the file system (in bytes), respectively. The total cost of executing all queries Q is given by: $TIOC(Q, FS_i) = \sum_{k=1}^{m} IOC(Q_k, FS_i)$. Our horizontal partitioning schema problem can be formulated as follows:

Maximize $Eval(Q, FS_i) = (TIOC(Q, \phi) - TIOC(Q, FS_i))$ *subject to* $N_i \leq W$, where $TIOC(Q, \phi)$ is the cost of executing the set of queries on the no partitioned schema. Note that our GA may generate fragmentation schemas that violate the maintenance constraint (see Section 2). In order to penalise these schemas, a penalty value is introduced as a part of the fitness function. In our study we used a linear penalty [9] defined as: $Pen(Q, FS_i) = \alpha \times (N_i - W)$, where α is the penalty factor (a constant > 0). Our final fitness function follows the *divide mode* [9] and defined as follows:

$$F(Q, FS_i) = \frac{Eval(Q,FS_i)}{Pen(Q,FS_i)}, \text{ if } Pen(Q, FS_i) > 1,\ Eval(Q, FS_i); \text{ otherwise.}$$

3.2 Simulated Annealing Algorithm Setting

The SA is applied on the final solution obtained by the GA. This means that the initial state of SA is the fragmentation schema generated by the GA. Random moves used by the SA are applied on the final multidimensional array of GA. In order to facilitate their implementation, the initial array is transformed into one dimensional array, by concatenating its all rows. This is gives a new representation of the fragmentation schema of fact table. The random moves generate a new problem, called the *validation of a solution*. This is due to the fact that our SA consists in modifying cell values of the array by incrementing or decrementing them. This may cause an overflow of domain values of cells. To solve this problem, we have developed a function that checks the validity of each solution generated by SA. The fitness of each solution is calculated using the cost model (TC) developed in Section 3.1. The steps of our SA are shown in Algorithm 1.

4 Experimental Studies

For our study, we use the dataset from the APB1 benchmark [2]. The star schema of this benchmark has one fact table Actvars and four dimension tables: *Actvars* (24 786 000 tuples), *Prodlevel* (9 000 tuples), *Custlevel* (900 tuples), *Timelevel* (24 tuples), and *Chanlevel* (9 tuples). This warehouse has been populated using the generation module of APB1. Our simulation software was built using Visual C performed under a Pentium IV 1,5 Ghz microcomputer (with a memory of 256 Mo).

Algorithm 1. Simulated Annealing Algorithm

```
Input: initial_state (the fragmentation schema generated by GA), initial_temperature;
Output: minstate;
begin
    minstate:= initial_state; cost := TC(initial_state); mincost := cost;
    temp := initial temperature;
    repeat
      repeat
          newstate:=after random move; validation_check(newstate); newcost:= TC(newstate);
        if (newcost $\leq$ cost) then state := newstate; cost := newcost
        else with probability $e^{(\frac{newcost-cost}{temp})}$
          state := newstate; cost := newcost
        end;
        if ($cost < mincost$) then minstate := state; mincost := cost
        end
        until equilibrium not reached;
        reduce temperature
      until not frozen;
      return minstate
end
```

Experimental Setup. We have considered 15 queries. Each query has selection predicates, where each one has its selectivity factor. The page size (PS) is 65536 bytes. We considered 9 fragmentation attributes. The number of sub domains generated by these attributes is 40. An exhaustive algorithm should generate 2^{40} fragmentation schemas to get the optimal solution. Due to this large number we did not consider an exhaustive search. Our experiments were conducted as follows: firstly, we conduct experimentation to identify the good parameters of our GA: (1) number of generations, (2) number of chromosomes, (3) crossover rate, (4) mutation rate, secondly, using these parameters, we run our GA and then the SA.

The relevant parameters obtained by the first experimentation is [1]: 500 generations (40 chromosomes per generation), crossover and mutation rates are 70% and 30%, respectively, in the beginning. After several generations, the mutation rate of 6% was used to ovoid a redundant search.

We have conducted experiments to get the best value of penalty factor α. To do so, we have chosen several values of α and for each one we execute our GA. The results show that the penalty factor has a great impact of the final solution. Figure 1 shows that when α is small, the performance of queries is good. The logic behind this result is that when α is small, the maintenance constraint will be relaxed, therefore our GA explores more solutions and does not ignore the penalised solutions. Figure 2 confirms this result by showing the explosion of the number of fragments when α is less than 0.5 for non uniform distribution (NUD). For the rest of experiments, we choose $\alpha = 0.01$.

[1] For lack of space these experiments are not presented.

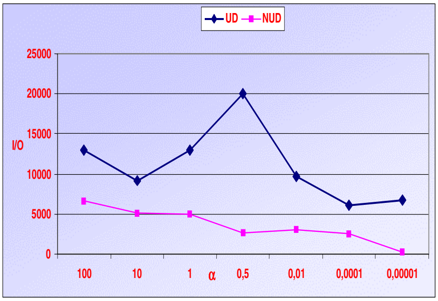


Fig. 1. The effect of the penalty factor on the quality of fragmentation schemas

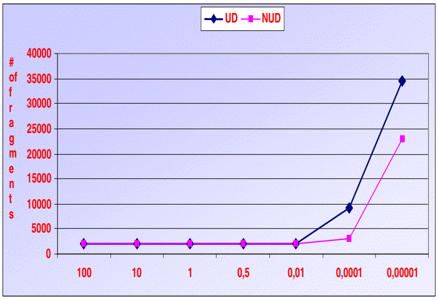


Fig. 2. Impact of penalty factor on the number of generated fragments

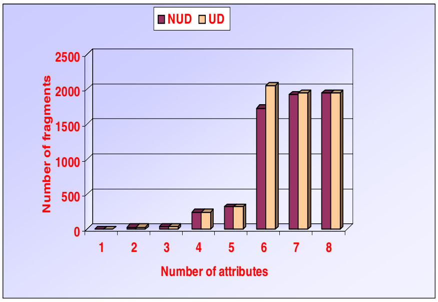


Fig. 3. The impact of the number of fragmentation attributes on number of final fragments

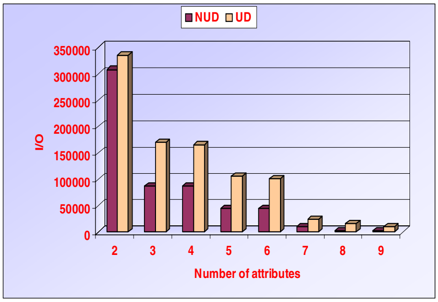


Fig. 4. The impact of the number of fragmentation attributes on total IO

Experimentation of GA. In order to evaluate the impact of the used fragmentation attributes, we did an experimentation that varies the number of these attributes. It starts from one to nine attributes, and for each step, it calculates the number of generated fragments, the number of inputs outputs for executing the 15 queries. The results show that the number of fragments and IOs are proportional with the number of used fragmentation attributes (Figures 3 and 4). The main guideline that a DWA can consider is that if he wants to speed up his queries, he should use a large number of attributes covering all dimension tables. This is because OLAP queries are executed on the whole schema. Figure 5 studied the effect of the dimension tables participating on the fragmentation process. The performance of OLAP queries is proportional with the number of these tables. In Figure 6, we realised that the rate between the returned fragments by our GA and the possible ones varies according the number of the used fragmentation attributes. When only six attributes are used, this rate is high. Starting from six attributes, this rate becomes small (less than 1.5%). This is due to the augmentation of the number of possible fragments.

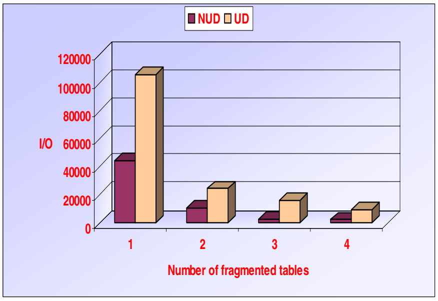


Fig. 5. Number of dimension tables participating in the fragmentation process and their impact on performance

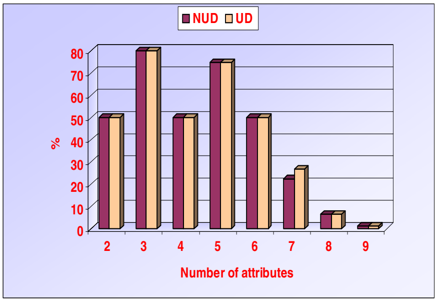


Fig. 6. The rate between the number of generated fragments and possible fragments

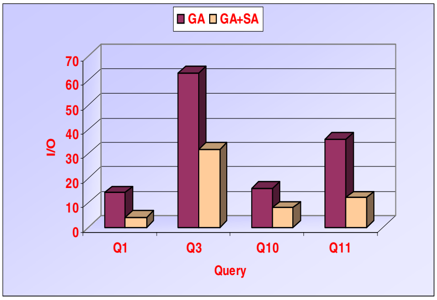


Fig. 7. Profitable queries

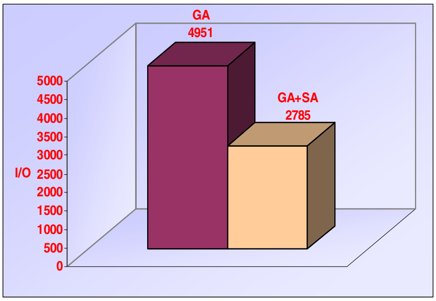


Fig. 8. Query reduction after SA

Experimentation of the SA. Parameters used in our SA are: initial temperature was 400 which is decremented by 2 each 100 iterations, threshold was 2000, α was 0.01, and the number of generations with improvement was fixed at 21000. We have studied the effect of GA and SA on different queries. The reduction is significantly when using the SA. We realised that among the initial set of queries (15), some queries do not get benefit from the application of SA. This is because that their selection predicates do not match with the fragmentation predicates generated by SA (see Figure 7). Finally, Figure 8 shows the impact of SA on global query processing cost reduction. It was reduced by 44% after applying the SA. We can conclude that the combination of GA and SA gets a good result (SA is complementary to GA).

5 Conclusion

In this paper, we concentrate on horizontal partitioning of relational warehouses. The number of fact fragments generated by our partitioning methodology can be very huge. Therefore, it will be difficult for the data warehouse administrator to maintain all fragments. To solve this problem, firstly, we formalised it as an optimisation problem. Secondly, we proposed a hybrid method combining genetic and simulated annealing algorithms. The use of SAs avoids the problem of premature convergence inherent to GAs by allowing uphill moves to solutions with a worse fitness. This fitness is characterised by a cost model that evaluates the cost of executing a set of most frequently queries performed on a top of the partitioned warehouse schema. This cost model takes into account the penalty function. Finally, we implement the proposed solution using the APB-1 benchmark. Our experimental studies go through three steps: (1) identification of relevant parameters that will be used for our GA, (2) execution of GA using these parameters, and (3) execution of our SA based on the solution obtained by our GA. The experimental results are encouraging and show the feasibility of SAGA.

Interesting extension of this work concerns the application of the proposed algorithms to select join indexes due to the similarity between star join index and fact fragments.

References

1. L. Bellatreche and Boukhalfa K. An evolutionary approach to schema partitioning selection in a data warehouse environment. *Proceeding of the International Conference on Data Warehousing and Knowledge Discovery (DAWAK'2005)*, pages 115–125, August 2005.
2. OLAP Council. Apb-1 olap benchmark, release ii. http://www.olapcouncil.org/research/bmarkly.htm, 1998.
3. J. H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, Michigan, 1975.
4. Y. Ioannidis and Y. Kang. Randomized algorithms algorithms for optimizing large join queries. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 9–22, 1990.
5. S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.
6. S. Papadomanolakis and A. Ailamaki. Autopart: Automating schema design for large scientific databases using data partitioning. *Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM 2004)*, pages 383–392, June 2004.
7. A. Sanjay, V. R. Narasayya, and B. Yang. Integrating vertical and horizontal partitioning into automated physical database design. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 359–370, June 2004.
8. T. Stöhr, H. Märtens, and E. Rahm. Multi-dimensional database allocation for parallel data warehouses. *Proceedings of the International Conference on Very Large Databases*, pages 273–284, 2000.
9. J. X. Yu, C-H. Choi, and G. Gou. Materialized view selection as constrained evolution optimization. *IEEE Transactions On Systems, Man, and Cybernetics, Part 3*, 33(4):458–467, November 2004.

Scheduling Strategies and Their Evaluation in a Data Stream Management System*

Sharma Chakravarthy and Vamshi Pajjuri

Information Technology Laboratory and Department of Computer Science and Engineering
The University of Texas at Arlington
sharma@cse.uta.edu

Abstract. MavStream, a Data Stream Management System (DSMS), has been developed for processing stream data from applications such as network monitoring, sensor monitoring and traffic management systems that require near-real time results and have to process unbounded streams of data. In order to be useful, a result produced by MavStream has to meet certain Quality of Service (QoS) requirements on tuple latency, memory usage, and throughput. Strategies used for scheduling the operators of continuous query (CQ) significantly affect the QoS metrics and hence are of interest. This paper discusses scheduling strategies used in MavStream, their design, implementation, and evaluation. Scheduling is done in MavStream at the operator level. The scheduler maintains a ready queue of operators and decides on the operators to be scheduled based on the scheduling strategy. We first introduce the path capacity scheduling strategy with the goal of minimizing tuple latency by scheduling operator paths with maximum processing capacity. Later we discuss segment-scheduling strategy that aims at minimization of total memory requirement by scheduling operator segments with maximum memory release capacity. We then discuss simplified segment strategy, which splits operator path into just two segments providing better tuple latency performance than segment scheduling strategy and lower memory utilization than path capacity scheduling strategy. Extensive set of experiments have been designed and performed to evaluate the proposed scheduling strategies by simulating real time streams. The performance metrics of average tuple latency, memory utilization and throughput are compared with each other for different strategies and with round robin strategy to validate the analytical conclusions.

1 Introduction

Traditional database management systems (DBMSs), consisting of a set of persistent relations, a set of well-defined operations, and a highly optimized query processing engine, have been researched for over 25 years and are widely used in applications that require persistent storage and processing of ad hoc queries to manage and process large volume of data. DBMSs are designed for managing large amounts of data and

* This work was supported, in part, by the following NSF grants: IIS-0326505, EIA-0216500, MRI 0421282, and IIS 0534611.

D. Bell and J. Hong (Eds.): BNCOD 2006, LNCS 4042, pp. 220 – 231, 2006.

for ad hoc querying and generating reports. They support features such as consistency, concurrency and recovery over well-defined operations over relations such as *insertion, deletion, and update* and is efficient for supporting large volume of transactions over persistent data.

However, the past few years have witnessed a large class of data-intensive applications that produce data at high update rates (e.g., stock markets, sensor applications, and pervasive environments). Also these applications produce data continuously and the data is typically presented in the form of a *stream*. As a result, the volume of a data stream is huge. These applications need processing capability for data arriving continuously which may be used for monitoring interesting changes or patterns over the data in a timely manner. An example would be to find traffic accidents on a highway. These applications are different from DBMS applications in terms of their data sources and computation requirements.

It is clear that these applications do not fit the traditional DBMS processing model and its querying paradigm. DBMSs are not designed to load data in the form of streams and to provide continuous computation, expressed as *continuous queries*, over this data. The techniques developed for DBMSs cannot be directly applied to these applications as they were not designed for real-time applications and to support quality of service (QoS) requirements. A re-examination of the processing model and query processing techniques are needed to match the requirements of an increasing number of stream-based applications. The systems that are used to process data streams and satisfy the needs of those stream-based applications are termed Data Stream Management Systems (DSMSs).

MavStream is a generalized DSMS designed for supporting continuous query processing on stream data and to support QoS specifications associated with queries. In MavStream [1], user can give a query from a GUI, which is instantiated, scheduled and executed. Queries can be ad-hoc or continuous [3]. Continuous streams are given as input and results are computed in real-time without storing the data in secondary memory. This paper mainly concentrates on scheduling alternatives to support the predefined Quality of Service (QoS) requirements for a continuous query. The continuous streams are unbounded and can be thought of as a relation with infinite tuples. The result produced has to be within a specified average tuple latency range since most of these applications need to react in a timely manner. This could significantly be affected by the way a query is scheduled. As the size of the main memory and the CPU speed is limited and DSMS requires real time results, optimization of memory usage is critical for DSMSs. Maximum memory utilization should be bounded as the queries are executed in main memory. Thus, we address the two QoS requirements: average tuple latency and maximum memory utilization, in this paper. The user specifies a threshold for the average tuple latency and the memory utilization. This QoS for a query can be achieved by selecting the appropriate strategy for scheduling the operators of the query and hence is a major area of interest and the main focus of this paper. In this paper the scheduling strategies are extended and then a comparison is made to evaluate their effectiveness. A single scheduling strategy would not be able to satisfy both QoS measures, as there are tradeoffs among the performance metrics and the usage of limited resources. So we have introduced several scheduling strategies [7].

1. Path Capacity scheduling to achieve minimum tuple latency.
2. The segment scheduling strategy to achieve the minimal memory requirements.
3. The simplified segment strategy, which requires slightly more memory but gives better tuple latency than the segment strategy.

In Section 2 we describe related work in this field. In section 3 we describe the architecture of MavStream while section 4 discusses the design of the scheduling strategies presented in MavStream and their limitations. In section 5, the experiments and results are given to evaluate the strategies. We conclude with our contributions and a summary of directions for future work in Section 6.

2 Related Work

Aurora [5] is a data flow system that uses the primitive box and arrow representation. Tuples flow from source to destination through the operational boxes. The scheduler is responsible for multiplexing the processor usage to multiple queries. To reduce the scheduling and operator overhead, Aurora relies on batching. Aurora scheduler performs dynamic scheduling-plan construction and QoS-aware scheduling. Aurora employs two level [8] scheduling approach to address the execution of multiple simultaneous queries. The first level handles the scheduling of super boxes, which is a set of operators, and the second level decides how to schedule a box within a super box. They have discussed their strategy to decrease the average tuple latency based on superbox traversal. However, we have used a different approach and have proved that our PCS is optimal in terms of overall tuple latency. Although their tuple batching - termed train processing, uses a similar concept used in the Chain strategy and our segment strategies, no algorithm has been provided to construct those superboxes.

STREAM [3] is a prototype implementation of a complete data stream management system being developed at Stanford. STREAM has a central scheduler that has the responsibility for scheduling operators. The scheduler dynamically determines the time quantum of execution for each operator. Period of execution may be based on time, or on the number of tuples consumed or produced. STREAM uses the chain scheduling strategy [9] with the goal of minimizing the total internal queue size. It supports continuous queries but has not addressed the issue of ad-hoc queries. As a complement to this, path capacity scheduling proposed in this paper minimizes the tuple latency.

In Rate Based Optimization [10] the fundamental statistics used are estimated from the rate of the streams in the query evaluation tree. They estimate the output rates of various operators as a function of the rates of their input and use these estimates to optimize queries. The Rate Based Optimization aims at maximizing the throughput of the query. They do not consider tuple latency and memory requirement, which are the prominent QoS measures.

Eddies [11] support dynamic re-optimization in decentralized dataflow query processing. It relies on simple dynamic local decisions to get a good global strategy. Eddies have flexible prioritization scheme to process tuples from its priority queue. Lottery Scheduling is utilized to do local scheduling based on selectivity. By adding a feedback to the control path, it effectively favors the low-selectivity operators. Eddy module directs the flow of tuples from the inputs through the various operators to the output, providing the flexibility to allow each tuple to be routed individually through the operators. Eddy's tuple buffer is implemented as a priority queue with a flexible prioritization scheme. They use a router to schedule that also continuously monitors the system status. The large amount of status information associated with the each tuple affects its scalability.

3 MavStream Overview

MavStream is modeled as client-server architecture in which client accepts input from the user and sends the transformed input to the server over predefined protocols. The various components of MavStream are shown in Figure 1. The web-based client provides a graphical user interface to accept queries. It constructs a query plan object, which is a tree of operator nodes that represents a single complete query and includes QoS requirements from user specifications. MavStream server is a TCP Server, which listens on a predefined port. It is responsible for executing user requests, and producing desired output. It accepts commands and requests from a client that describes the task to be carried out. It provides integration and interaction of various modules. It provides details of available streams and schema definitions to clients so that they can pose relevant queries to the system. It initializes and instantiates operators constituting a query and schedules them. It also handles the starting and stopping of a query.

The query plan object and stream definitions are sent to the server. The Instantiator has the responsibility of initializing and instantiating streaming operators and their associated buffers on accepting the query plan object from the client. Instantiator traverses the query tree in a bottom-up manner to ensure that child operators are instantiated prior to parent operators, to respect query semantics as data flows from leaves to root. It also associates input and output buffers (or queues) with desired parameters to operators for consuming and producing tuples.

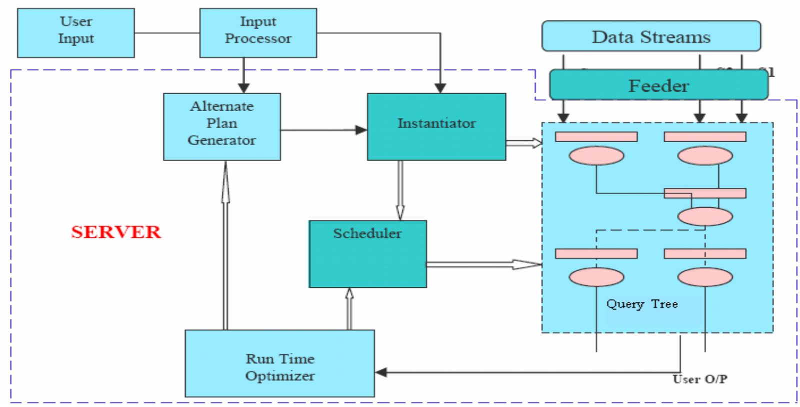


Fig. 1. MavStream Architecture

The Alternate Plan Generator module takes the plan given by the user as input and gives alternate plans that are more efficient than the input plan. The best alternate plan (locally optimal) of a query tree may not be the optimal with respect to a global plan. An alternate plan is considered the best when most of its operators are merged with the existing global plan or the global plan satisfies user-specified QoS specifications.

Run time optimizer uses alternate plan generator in order to dynamically select an alternate plan when the result produced by the previous plan does not satisfy the quality of service requirements. To ensure QoS, optimizer monitors output and may ask the scheduler to change the scheduling policy or increase the priority of the operators or assign different time quantum for specific operators which need more time.

The scheduler is one of the critical components in MavStream. The scheduler maintains a ready queue, which maintains the order in which operators are scheduled. The server initially populates this queue. Operators must be in a ready state in order to be scheduled. The rest of the paper discusses the scheduling strategies used in MavStream.

4 Scheduling Strategies

Operating systems provide a coarse control over thread scheduling and are not useful when scheduling needs some prioritization derived from application or query semantics. In MavStream, we do operator level scheduling as it provides better control and less overhead. Design of a scheduler and the scheduling strategies used have a significant impact on meeting the QoS specifications. A good scheduling strategy should be able to handle unexpected overload situation and should have low scheduling overhead. Unfortunately, all of the QoS requirements listed above cannot be satisfied by a single scheduling scheme. Hence we have implemented three scheduling strategies [7, 13] to deal with the different QoS metrics.

1. Path capacity scheduling strategy (PCS) to minimize tuple latency.
2. Segment scheduling to minimize memory utilization.
3. Simplified segment strategy, which is hybrid of the above two strategies.

MavStream supports relational operators such as select, split, project and windowed versions of operators [4] such as aggregate and join. User query is made up of these operators where the tuples are pushed from the bottom node (leaf node) to the root. The following notations are used in the description of scheduling all the strategies.

i) Operator selectivity σ_i: The ratio of the number of tuples going out to the number of tuples coming in.

ii) Operator processing capacity $C^P_{O_i}$: The number of tuples that can be processed in one time unit at operator O_i.

iii) Operator memory release capacity $C^M_{O_i}$: The number of memory units that can be released within one time unit by operator O_i.

iv) Operator path processing capacity $C^P_{P_i}$: The number of tuples that can be processed within one time unit by operator path P_i.

v) Operator service time ($1/ C^{P}Oi$): The number of time units needed to process one tuple at this operator Oi.

vi) Path memory release capacity $C^{M}Pi$: The number of memory units released within one time unit by the operator path Pi.

4.1 Path Capacity Scheduling Strategy

The path from the leaf node to the root node is called an operator path. The number of operator paths is equal to the number of leaf nodes. The total time spent by a tuple to reach the root node for an operator path with k operators with selectivity σ_i (where i=1…k-1) is $1/ C^{P}O_1 + \sigma_1/ C^{P}O_2 + ...+ \sigma_1*...*\sigma_{k-1}/ C^{P}O_k$. The time taken by one tuple to go from the leaf node to the root node is the service time of the operator path. Processing capacity of the operator path is the inverse of service time of the operator path.

Path Capacity Scheduling (PCS) schedules the operator path with maximum processing capacity as long as there are tuples present in the input buffer of the operator path. Once the operator path is chosen for scheduling, a bottom-up approach is used to schedule the operators along the operator path. It is a static scheduling strategy as the processing capacity of operator completely depends on the system and the selectivity of the operator, which can be found by monitoring the output of the operator.

PCS minimizes tuple latency and has less scheduling overhead as the number of operator paths is less than the number of operators. In round robin and other operator scheduling strategies, the operators of two operator paths are scheduled in an interleaving manner. The tuples are buffered in the intermediate queues that increase the tuple latency. Path capacity strategy has starvation problem as most of the resources are given to the operator path having maximum processing capacity. As each operator is implemented as a single thread, context-switching overhead for both scheduling strategies are same.

4.2 Segment Scheduling Strategy

Segment strategy is developed with the goal of minimizing the total internal queue size. Segment construction involves partitioning the operator path into segments and then pruning the segment list. Consider the operator path with k operators O1, O2… Ok, with O1 being the leaf operator and Ok the root operator. Segment creation starts by adding O1 to the segment and then adding to the segment the next operator if it has a higher memory release capacity. This process is continued until the next operator in the sequence has lower memory release capacity. Then a new segment is started and the process continues until the root operator is added.

Memory release capacity of a segment with k operators is C^{P}_{S} (*InputTupleSize - OutputTupleSize* * $\sigma 1 \sigma 2 .... \sigma k$). The scheduler chooses the segment with maximum memory release capacity as long as there are tuples present in the input buffer of the segment or there exists another segment with more memory release capacity than the current segment. Once the segment is chosen for scheduling, a bottom-up approach is used to schedule the operators along the segment. It is a static scheduling strategy.

In the query plan construction, we generally push the lower selectivity operators down the tree, due to which there will be less number of tuples present in the system.

Segment scheduling takes advantage of lower selectivity and higher processing rates of the bottom side operators by scheduling the segment with maximum memory release capacity more often. In the segment scheduling, we buffer partially processed tuples at the start of a segment that contributes towards higher tuple latency than the path capacity strategy. Segment scheduling has less scheduling overhead than the round robin as the number of segments is less than the number of operators in a query plan.

4.3 Simplified Segment Scheduling Strategy

Simplified segment scheduling strategy is a variant of the segment scheduling strategy. The operator path is partitioned into at most two segments. The first segment includes the leaf operator and its adjacent operators such that the ratio of the memory release capacity of the next operator to the current operator is more than the constant factor γ, which is less than one to reduce the number of segments. All the other operators along the operator path will form the second segment. In the previous segment construction algorithm, γ value used is 1.

Memory requirement is slightly more than the segment scheduling strategy because the first segment releases more memory. The tuple latency is less than the segment scheduling strategy because the tuples are buffered at most two times in the operator path. The scheduling overhead is less than the segment scheduling because the number of segments in a segment strategy is more than the number of simplified segments in a query plan.

4.4 Design Issues

The scheduler generally maintains a ready queue (containing the operators) that provides the order in which operators are scheduled. When a query plan and their operators are instantiated by the instantiator, server populates the ready queue. The traditional approach of maintaining a single ready queue does not allow switching between scheduling strategies depending on QoS requirements. Also, it is not appropriate to be generating segments at run time as it interferes with processing and causes increased tuple latency. Figure 2 shows the new design, which will allow us to switch from one scheduling strategy to other dynamically at run time. Instead of a single ready queue, we have a sequence of ready queues each of which contains paths/segments for a scheduling strategy. These are sorted with respect to their processing or memory release capacities. The processing capacity of operator paths and the memory release capacity of the segments depend on the selectivity of each operator. If QoS requirements are not met, a different scheduling strategy is taken. All of the described scheduling constructs are created when a query plan is instantiated. The paths are revised on server during the course of query execution, as the selectivity of each operator is monitored and updated. The query given by the user is a tree of operator nodes that has complete information about the flow of data. The starting selectivity of each operator is assumed by the system and the paths are computed from the query tree before operators are instantiated.

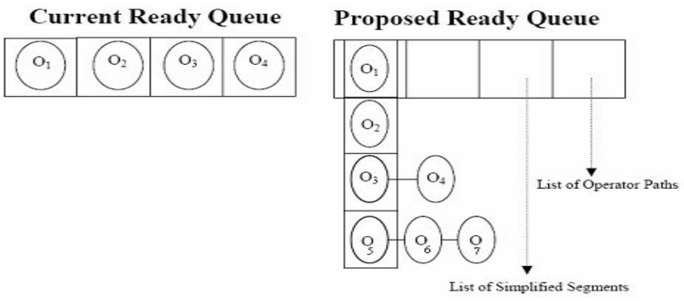


Fig. 2. Ready Queues

During query execution as the selectivity of operators is monitored, the values of the selectivity might change and because of this the priorities of the operator paths might change as well. To handle this problem, we sort the ready queue of the scheduler at the end of iteration. This approach introduces an overhead of sorting. As the number of paths in a ready queue is quite small, the overhead introduced is minimal.

The use of the above scheduling strategies can lead to starvation of tuples present in input buffers of the non-scheduled paths. To avoid this we schedule paths using a minimum threshold (τ) number of tuples present in the input queue. The threshold value is chosen based on the system load. If we choose a small value then the scheduling overhead increases, and a large value might result in increased tuple latency.

5 Experimental Evaluation

All experiments were run on an unloaded machine with 2 Xeon processors, 2.4GHz each, 2GB RAM and Red Hat Linux 8.0 as the operating system. The data set for performance evaluation is obtained from MavHome [6]. The continuous stream collected from MavHome is stored in a database. A feeder module generates input data streams which are fed to the MavStream server. The feeder module can be configured such that the delays between tuples follow a Poisson distribution with a specified arrival rate.

The performance metrics considered are: average tuple latency, memory utilization and throughput. To calculate the average tuple latency, we timestamp each tuple when entering and leaving the system and use these timestamps to calculate the actual tuple latency. To calculate memory utilization, we monitor each buffer every second and determine the number of tuples present in it. Multiplying number of tuples by the average size of tuple in that buffer gives memory utilization. To calculate throughput, we monitor the output buffer every second and determine the number of tuples coming out and convert it into an average for a period.

5.1 Effect on Varying Data Rate on Average Tuple Latency and Maximum Memory

In this experiment, the effect of varying data rate on average tuple latency is observed for various scheduling strategies. It is run using a single query with six operators in

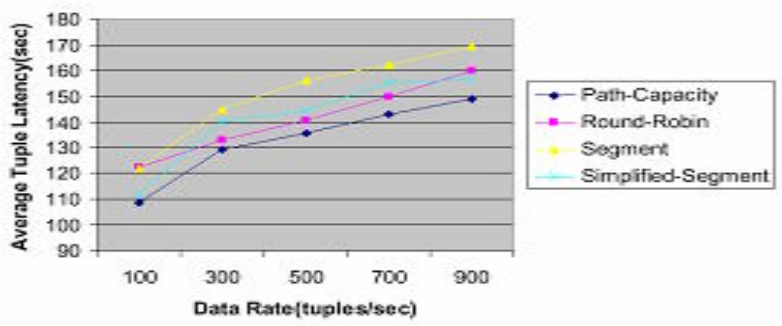


Fig. 3. Data Rate vs. Avg Tuple Latency

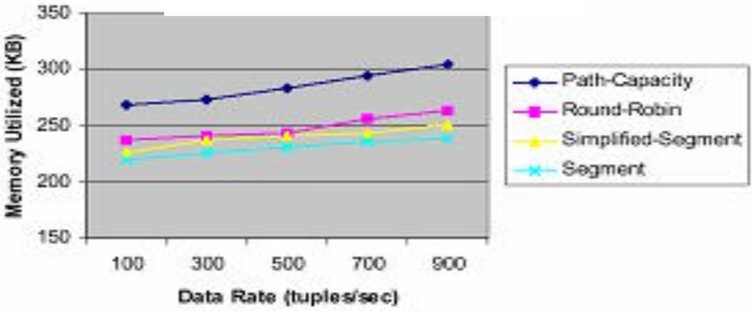


Fig. 4. Data Rate vs. Memory Utilization

the system. This experiment is done in main memory (with the buffer manager off). The data rate is varied from 100 tuples/sec to 900 tuples/sec. The data set is fixed with 1000 tuples/window with 5 windows. It can be observed from the Figure 3 that as the data rate increases the “Average Tuple Latency” increases. This is due to the increase in waiting time in the buffer which is proportional to the data rate.

In the path capacity scheduling strategy, tuples are processed without buffering in the intermediate queues, whereas in the other scheduling strategies, tuples are buffered in the intermediate queues and the waiting time as well as the tuple latency increases. Simplified segment strategy is better than the segment strategy as the number of times a tuple gets buffered is at most two. In this experiment, the effect of varying data rate on Maximum Memory Utilization is also computed for various scheduling strategies. It can be observed from Figure 4 that, as the data rate increases the “Maximum Memory Utilization” also increases. In the path capacity scheduling strategy, unprocessed tuples are buffered in the input queues of base operators. Segment scheduling strategy takes advantage of lower selectivity and processing capacity of the bottom operator. Segment scheduling strategy picks out the particular segment that is most effective at reducing memory usage and schedules it repeatedly as long as there are tuples present in its input queue, reducing memory utilization. On the other hand, round robin executes the best operator less frequently, increasing the memory usage.

5.2 Throughput of the System During the Query Execution

In this experiment, the number of tuples given out by the system is monitored during the course of execution of the query in various scheduling strategies. It is run using a single query with six operators in the system for different input data rates. This experiment is done in the main memory. This is done so as to study the throughput. The data set is fixed at 1000 tuples/window with 5 windows.

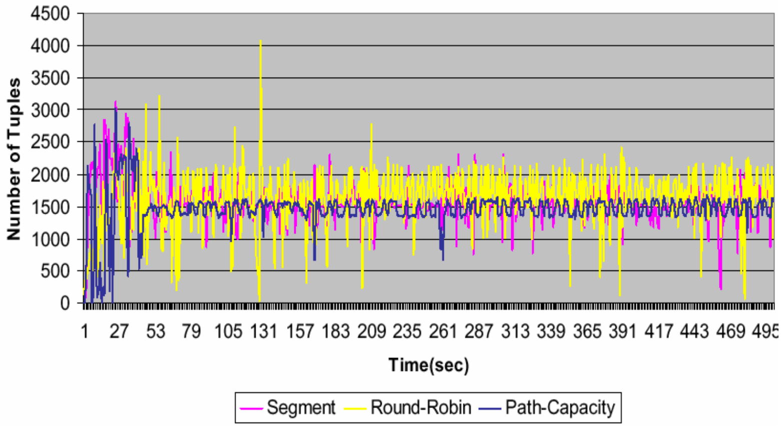


Fig. 5. Throughput vs. Time

It is observed from Figure 5 (simplified scheduling strategy is not shown for improved legibility) that the output pattern of the path capacity scheduling strategy is much smoother than the segment scheduling strategy as well as the round robin scheduling strategy.

5.3 Memory Utilization by the System During the Query Executions

In this experiment, the memory utilized by the system is monitored during the course of execution of the query in various scheduling strategies. It is run using a single query with six operators in the system. This experiment is done in the main memory. The data rate is 40 tuples/sec for the first 150 seconds and then it is increased to 80 tuples/sec until 200 seconds. The data rate is again decreased to the normal rate of 40 tuples/second until 300 seconds and then increased to 80 tuples/sec until 350 seconds and brought down to 40 tuples/second and fixed until the end of execution. This is done so as to study the memory utilization when the input rates change and to observe the effectiveness of scheduling algorithms on memory utilization during bursty input. The data set is fixed at 2000 tuples/window with 10 windows.

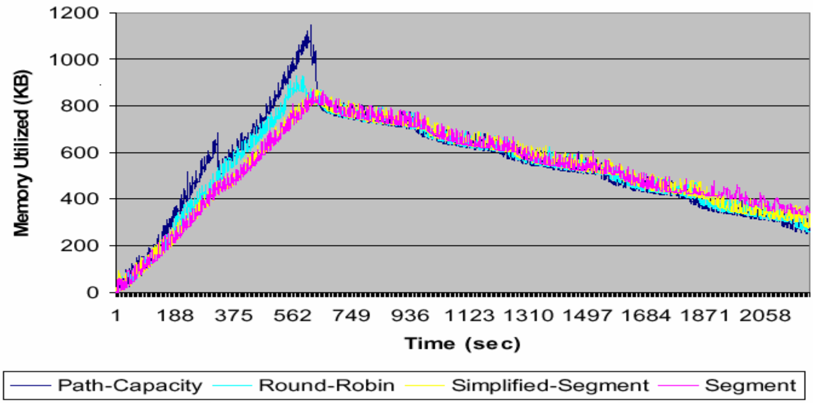


Fig. 6. Memory Utilization Vs Time

It can be observed from the Figure 6 that the memory utilized by the segment scheduling strategy is less than the path capacity scheduling. As the bottom operators have low selectivity, they can release more memory and segment scheduling takes advantage of these operators and schedules these operators more often to release more memory. Simplified segment take slightly more memory than the segment strategy as it divides the operator path into two segments. Path capacity strategy has some bursty nature in the memory consumed at the times when there is an increase in the input rate. As the tuples are fed to the base buffers the memory utilization increases and when tuples are no longer fed to base buffers the memory utilization decreases.

6 Conclusion and Future Work

In this paper, we have discussed several scheduling strategies for MavStream that will help address the QoS requirements. All the strategies described in this paper have been implemented as part of the MavStream system. Path capacity scheduling strategy minimizes the average tuple latency of the stream processing system. Segment scheduling is implemented to minimize the maximum memory required by the system by prioritizing the segment, which can release more memory. Similarly, simplified segment scheduling takes slightly more memory than segment strategy and gives better tuple latency. We have evaluated these scheduling strategies for various performance metrics such as tuple latency, memory utilization and throughput. MavStream server is designed to construct the operator paths, segments and simplified segments from the query tree. It also helps in calculating the performance parameters used by the scheduler such as processing capacity of operator path, memory release capacity by the segments and simplified segments.

A number of extensions to the Mavstream system are underway. Shedding [12] of tuples (termed load shedding) need to be employed when QoS requirements are not met even after changing scheduling to the best possible strategy. This may involve sacrificing some accuracy [2] of the results. A run-time optimizer is being implemented to monitor QoS of the system and match it against user-specified values to heuristically change scheduling and load shedding to maximize the performance of the system. Currently, we are combining event processing with stream processing in a synergistic manner to provide an end-to-end system for advanced monitoring applications.

Acknowledgments

The authors would like to thank Vihang Garg, Raman Adaikkalavan, and Balakumar Kendai on their help in revising the paper and formatting it.

References

1. A. Gilani. Design and Implementation of Stream Operators, Query Instantiator and Stream Buffer Manager. MS Thesis, CSE Dept.. The University of Texas at Arlington, 2003. [online] http://www.cse.uta.edu/research/publications/Downloads/CSE-2003-37.pdf
2. R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. S. Manku, C. Olston, J. Rosenstein, and R. Varma. Query processing, approximation, and resource management in a data stream management system. In Proc of CIDR, Jan 2003: pages 245-256.
3. B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In Proc of ACM PODS, June 2002: pages 1-16.
4. S. Sonune. Design and Implementation of Windowed Operators and Scheduler for Stream Data. MS Thesis CSE Department. The University of Texas at Arlington, 2003. [online] http://www.cse.uta.edu/research/publications/Downloads/CSE-2003-38.pdf
5. D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. Zdonik. Monitoring streams - a new class of data management applications. In Proc of the VLDB, 2002.
6. Cook, D., et al., MavHome: An Agent-Based Smart Home. In Proc of the Conference on Pervasive Computing, 2003. http://mavhome.uta.edu
7. Jiang Q., Chakravarthy S., Scheduling strategies for Processing Continuous Queries over Streams In Proc of 21st BNCOD (2004), pages: 16-30, July 2004.
8. D. Carney, U. Cetintemel, A. Rasin, S. Zdonik, M. Cherniack, and M. Stonebraker. Operator scheduling in a data stream manager. In Proc. Of the VLDB, 2003.
9. B Babcock, S. Babu, M. Datar, R. Motwani, Chain: Operators Scheduling for Memory Minimization in Stream Systems. In Proc of the ACM SIGMOD, 2003.
10. S. Viglas, J. Naughton, Rate-based Query Optimization for Streaming Information Sources. In Proc of the ACM SIGMOD, 2002.
11. R. Avnur, J. Hellerstein, Eddies: Continuously adaptive query processing. In Proc of the ACM SIGMOD, 2000: pages 261-272.
12. N. Tatbul, U. Cetintemel, S. Zdonik, M. Cherniack, and M. Stonebraker, Load Shedding in a Data Stream Manager. In proc of the VLDB, 2003.
13. V. Pajjuri. Design and implementation of scheduling strategies and their evaluation in MavStream. , MS Thesis CSE Department. The University of Texas at Arlington, 2004. [online] http://itlab.uta.edu/ITLABWEB/Students/sharma/theses/Vamshi.pdf.

The Anatomy of a Stream Processing System*

Altaf Gilani, Satyajeet Sonune, Balakumar Kendai, and Sharma Chakravarthy

Information Technology Laboratory
and Department of Computer Science and Engineering
The University of Texas at Arlington
sharma@cse.uta.edu

Abstract. Data intensive applications such as network monitoring, financial applications; sensor-based applications etc. need to be supported by general-purpose systems rather than customized implementations. They have a continuous, unpredictable and unbounded flow of data as input, referred as streams. The fact that data comes as a stream with varying input rates (instead of accessing data stored on a disk in a predictable way) and that quality of service (QoS) requirements are stringent for these applications warrants a re-examination of the fundamental architecture of a DBMS. This paper describes the basic processing model and architecture of MavStream - a new Data Stream Management System (DSMS) being developed at UT Arlington. The architecture of MavStream is the primary focus of this paper. The user can give a continuous query from a graphical user interface (GUI), which is instantiated, scheduled, and executed by the MavStream server. We first provide an overview of the basic model and architecture and then describe some of the components of the system. We provide some experimental results to demonstrate the utility of the system and the effect of different scheduling strategies and buffer sizes on the performance and output.

1 Introduction

Traditional database management systems (DBMSs) are geared towards commercial data processing. They are not designed to support the needs of stream-based applications. Streaming applications [1, 2] have their own characteristics such as unbounded input, and arrival rates that are unpredictable (bursty). In a data stream management system (or DSMS), stored (continuous) queries [3] are applied to stream data. The approach taken by a DBMS (store the data and apply queries) is not applicable as near real-time responses are expected. First, some of the operators used in a DBMS are blocking in nature when applied to non-finite data. Second, DBMS has no provision for supporting quality of service (QoS) requirements of applications. A DBMS assumes data elements to be synchronized but most of the stream-oriented applications are asynchronous in nature. A DBMS produces exact and complete answers where as

* This work was supported, in part, by the following NSF grants: IIS-0326505, EIA-0216500, MRI 0421282, and IIS 0534611.

D. Bell and J. Hong (Eds.): BNCOD 2006, LNCS 4042, pp. 232 – 239, 2006.

DSMS applications need to produce incremental outputs and these applications can tolerate approximate answers (when load shedding is used).

The novel requirements of stream-processing applications motivated us to design and develop MavStream [13, 14] to support continuous queries over data streams. Some of the application areas include security, telecommunications data management, manufacturing, pervasive environments, click stream analysis, and military applications (such as monitoring soldier characteristics in a battle field).

The remainder of the paper is organized as follows. Section 2 describes related work. In section 3, we discuss the system architecture, operators, buffer management, and scheduling. This section also includes the implementation status. Section 4 presents experimental results. Conclusions are in section 5.

2 Related Work

Most of the ongoing research in data streams including MavStream has adapted the relational operators for a stream-processing environment. But systems such as Telegraph [4] does not rely upon a traditional query plan, but instead, constructs a query plan that contains adaptive routing modules, which are able to re-optimize [5,6] the plan on a continuous basis while a query is running. In Cougar [7] queries determine the data extracted from sensors thereby reducing the volume of raw data transmitted. NiagaraCQ [8] is a system that mainly focuses on supporting continuous query processing over multiple, distributed XML files. Aurora's [9] query algebra supports several primitive operations. Quality of service is associated with the output. Scheduler [10] is designed to cater to the needs of a large-scale system with real time response requirements. Stream [11] uses a modified version of SQL (CQL) as the query interface. Operators can produce approximate answers based on the available memory [12].

3 MavStream Architecture

MavStream [13, 14] uses a client-server architecture in which the client accepts input queries from the user, transforms it and sends the request to the server. The various components of MavStream are show in Fig. 1. The client provides a graphical user interface to accept queries. It constructs a plan object data structure which is a tree of operator nodes where every node describes the operator completely. The plan object is sent to the server using a pre-defined set of protocols.

MavStream server is responsible for instantiating the plan object and executing it to produce desired output. It consists of a number of modules such as instantiator, actual operators, queues associated with each operator to hold unprocessed tuples, buffer manager and a scheduler, and handles interaction between them through shared data structures and synchronization. It provides details of available streams and schema definitions to clients so that they can pose relevant queries to the system. It also allows new streams to be registered with the system. It initializes and instantiates

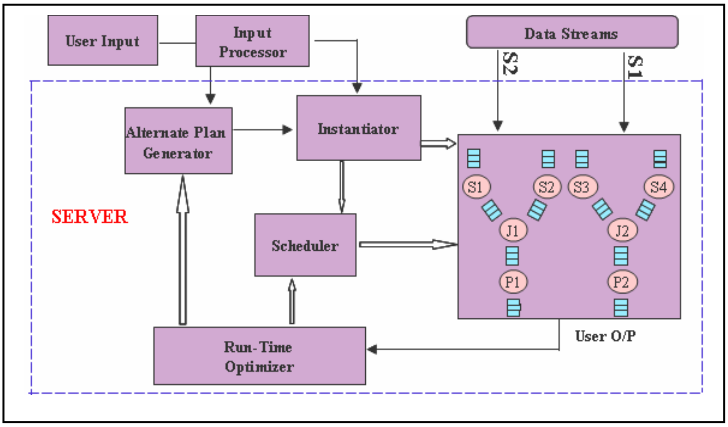


Fig. 1. Server Architecture

operators constituting a query and schedules them. It also stops a query on receiving command for query termination or when the life span of the query is over.

The instantiator module in MavStream has the responsibility of initializing and instantiating stream processing operators (such as select, project, windowed join and aggregate operators) and their associated buffers before a continuous query can be executed. The query plan is traversed in post order by the instantiator to ensure that child operators are instantiated prior to the parent operator in order to properly create the query tree (or plan). Instantiator extracts the information from the plan object data structure and uses it to create an executable operator.

The scheduler [14] initiates the execution of a query plan and sends the result back to the user through the client. The other modules in this architecture are: alternate plan generator and a runtime optimizer. The Alternate plan generator can be used for generating equivalent alternative plans for a given query. The gain may be magnified when it comes to optimizing a global plan by selecting one of the alternate plans in which most number of operators in an alternate plan merges with the existing operators in the global plan to share memory and computation. Run time optimizer is responsible for using the output and deciding changes (a different scheduling strategy, load shedding) that need to be carried out dynamically to satisfy user quality of service (QoS) requirements (such as minimizing tuple latency or memory).

3.1 Stream Operators

The operators are designed to produce continuous, real-time output as a result of processing queries. Blocking operators such as aggregates and join may block forever on their input, as streams are potentially unbounded. Hence, operators are broadly classified into: **Non-blocking Operators** and **blocking operators.** Non-blocking operators [13] process each tuple independently and hence the size of the stream is not an issue. Blocking Operators, on the other hand, require all tuples for their computation. However, in a streaming environment that is equivalent to waiting and not producing any output until the stream input is over. Since streams are assumed to be unbounded, this is not possible. Hence, the concept of a window (either a time interval or number of tuples) is used to compute blocking operators. Output is produced at the end of each window resulting in a continuous output that is also a stream.

3.2 Operator Model

The varying nature of an input stream is handled with data flow architecture for processing input tuples. Hence, it requires that the operators perform computation only when data is available. An operator processes tuples in its input buffers when it is allocated time by the scheduler. Further the operator should suspend itself when all the tuples in the input buffer are processed to maximize the time spent for computation. All operators interact frequently with other modules such as the buffer manager and the scheduler. Therefore it is necessary to present an abstraction of all operators to other modules of MavStream.

Each operator [13, 14] in MavStream is modeled as a separate thread. This allows one to schedule each operator explicitly and provides fine-grain control over their scheduling. Every operator in MavStream is modeled as a generic operator. This abstraction consists of properties common to all operators such as state, priority, output queues, and also provides an interface for other modules to start, stop, suspend and resume the operator. Through its lifetime an operator can be in one of the four states: ready, running, suspended, and stopped depending on the availability of data and the scheduling policy. All operators get a specified time quantum which is dependent on the scheduling policy and the priority of the operator.

Each operator reads a tuple from its input buffer processes it and delivers the result to the output buffer. Further window-based operators maintain an internal synopsis of tuples based on window definition. For join operator the synopsis will contain the input tuples that belong to a window from the both the streams. The join operator consumes a tuple from the input buffer, joins the tuple with all the tuples of other stream maintained in the synopsis that belong to current window and satisfy the join condition. The aggregate operators also computes the results incrementally as tuples are read from input buffer and output is given at the end of the window.

3.3 Window Specifications

The blocking nature of *join* and *aggregate* operators necessitates the use of windows, which allows the operator to not block entirely and provide continuous output. Windows [13] are classified as physical or logical based on how they are defined. Physical windows are defined using physical time while logical windows are defined using number of tuples in a window. Both physical and logical windows are further characterized based on how window endpoints are defined. A *Snapshot Window* is a single fixed window, typically used by a DBMS. A *Landmark Window* has a fixed starting point and the end point moves. As a result, windows get larger and larger. A *Sliding Window* has both of its end points moving in the same direction and by the same amount. Landmark and sliding window can be further classified based on the direction in which the window moves. Sliding windows can also be *overlapping* or *disjoint* based on the starting point of the next window.

In order to represent stream-based queries, the representation should clearly handle all of the above window specifications. SQL does not support window-based operations. Hence an extension has been added to support window-based queries. Also, this representation should support both physical and logical windows. The following four clauses are used in MavStream for specifying windows. *Begin Window* defines the starting point for the first window. *End Window* defines the ending point for the first window. *Hop Size* represents the amount by which the window will be moving in either direction. It is the hop size that determines the type of query window. Positive (negative) hop sizes moves the window forward (backward).*Start and End time* define the start and end query times.

3.4 Buffer Management

In a DSMS, between any two operators, there is a queue to hold the tuples that are yet to be processed by the operator. These queue sizes will vary during processing depending upon the input rate and the scheduling strategy. Storing all partially processed tuples may require an unbounded amount of memory. In reality as memory is bounded, we have implemented a pure main-memory alternative (buffer) and a disk-based alternative [13]. This can be specified as a parameter in the configuration file of the server. If the disk-based option is chosen, the tuples are written into indexed files after the buffer reaches a specified limit. The information that a tuple is stored or retrieved from main memory or a file is totally transparent to the operators.

The other feature of the buffer is that there can be more than one operator reading from a buffer at different consumption rates. In order to support these different rates of tuple consumption, buffer maintains a pointer (currentUnReadElement) for *each operator* associated with the buffer. This pointer points to the next tuple in the buffer that has not been read by the operator. Initially, each consumer operator has to register with the buffer. Buffer maintains a table, which contains the operators registered with the buffer and its currentUnReadElement pointer. This pointer needs to be adjusted when elements are de-queued from the buffer.

Purging tuples from a buffer has to be handled carefully as more than one operator is registered with the buffer. Before dequeuing a tuple from the buffer, purging logic needs to make sure that the element is not of use to *any* of its consumers; that is, all operators have read the element. For this, the purging logic makes use of currentUnReadElement pointer of each buffer. The minimum value of these pointer values is used for purging tuples.

Another important decision is to determine when to call the purging logic. Purging logic is called whenever an operator reads a tuple. Other operations on buffers do not invoke purging. This is done so that tuples are emptied out of the buffers at the earliest, creating more space for new tuples.

3.5 Implementation Status

MavStream has been implemented in Java. All modules except the alternate plan generator and the run-time optimizer has been implemented and tested. Split, select, joins and aggregate operators (sum, count, average, maximum, and minimum) have been implemented. For joins, both nested-loop and hash-based join algorithms have been implemented. Further, incremental and non-incremental (recomputed) versions of join have been implemented. All forms of windows including overlapping windows are supported.

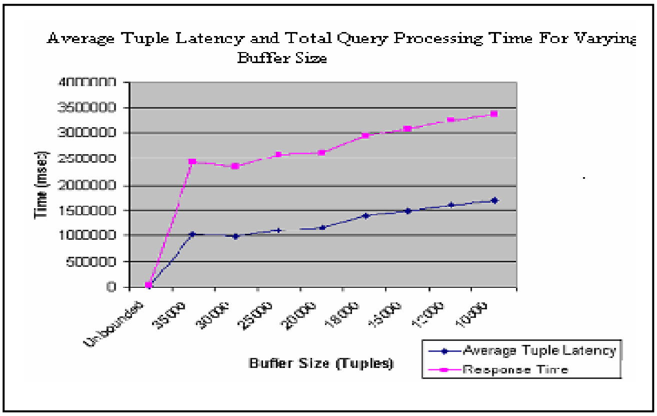


Fig. 2. Average tuple latency /Response time vs. Buffer size

Currently, round robin, weighted round robin, and flow-based scheduling (first-in, first-out) [14] has been implemented. A number of instrumentation has been incorporated to measure various usages: tuple latency, memory used, throughput etc.

4 Experimental Evaluation

All the experiments were performed on an unloaded machine with 2 Xeon processors, 2.4GHz, and 2GB RAM and Red Hat Linux 8.0 as the operating system. The data set for performance evaluation has been collected from smart home application [15] over a period of time. The live feed is stored in our database and is used as a stream to this system. Tuples are generated with a poison distribution to simulate real data set.

In the first experiment we observe the effect of buffer size on processing and response time. This experiment was run on five windows each consisting of 5000 tuples with a query having six operators. From Fig.2 it can be seen that when only main memory buffer is used (unbounded in the figure) it takes very negligible amount of time. As the buffer size is restricted, more tuples are stored on the disk and retrieved. This introduces additional I/O cost for reading and writing tuples. Hence both tuple latency and response times increase. Tuples stay longer the queues because of read and write to secondary storage.

In the second experiment different scheduling algorithms (round robin and weighted round robin) are studied to understand the behavior of *average tuple latency* with respect to availability of main memory. A single query with four operators is used for this experiment. The buffer assigned to each operator can contain at the most 1000 tuples. The data rate is fixed at 70 tuples/second. The data set is varied from 500 tuples per window to 1500 tuples per window.

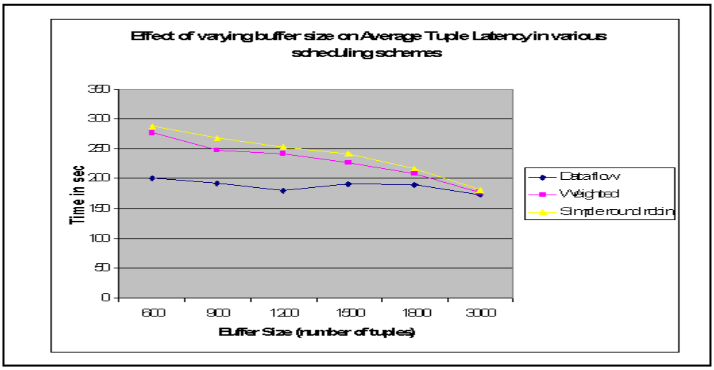


Fig. 3. Buffer size vs. Tuple latency

As expected the average tuple latency is inversely proportional to available memory. Higher the memory available to operators, the lower is the average tuple latency since no/few disk operations are involved. As the buffer sizes associated with operators are reduced, tuples that cannot be accommodated in main memory buffer are persisted on disk thus increasing the average tuple latency. This effect was observed by varying the scheduling schemes. It is observed again that data flow (or greedy approach) outperforms the other two scheduling strategies. Also weighted scheduling outperforms simple scheduling in all the cases. *Join* which is more complex and time consuming than *select* is assigned a higher priority. Since *select* operators are closer to data source in a query tree, they are assigned higher priority than *project* as they need to cope up with high input rates. This meaningful distribution of priorities to operators generates better result as observed in the Fig. 3. Simple scheduling scheme assign fixed priority to operators, hence cannot be used effectively to meet QoS requirements.

5 Conclusions

This paper briefly discusses the components of a stream data processing system and the issues that are specific to stream processing as compared to database query processing. Different types of streaming queries are identified and a general-purpose query representation is proposed. This representation covers all types of windows required for representing continuous queries.

A generic streaming operator is proposed which satisfies the query processing requirements of stream data. This provides a base model for implementing future operators. Effective buffer management strategies have been proposed to reduce the main memory utilized by the tuples and using secondary storages transparently at times of high load. We further analyze the effect of various scheduling strategies on varying the buffer size and the dataset. This work is being enhanced to include alternate plan generator and run time optimizer.

References

1. D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, S. Zdonik. Monitoring Streams- A new class of data management applications. In Proc of the 28th Intl. Conf. on VLDB, 2002.
2. B. Babcock, S. Babu, M. Datar, R. Motwani, J. Widom. Models and Issues in Data Stream Systems. Invited paper in Proc. of PODS 2002.
3. S. Chandrasekharan, M.J. Franklin. Streaming Queries over Streaming Data. In Proc of the 28th Intl. Conf VLDB, 2002: 203-214.
4. S. Chandrasekharan, O. Cooper, A. Deshpande, M.J. Franklin, J.M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, M.A. Shah. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In Proc. of CIDR 2003.
5. R. Avnur, J.M. Hellerstein. Eddies: Continuously adaptive query processing. In Proceedings of ACM SIGMOD 2000: 261-272.
6. S. Madden, M.J. Franklin. Fjording the stream: An architecture for queries over streaming sensor data. In Proc of the Intl. Conf. on Data Engineering, 2002: 555-566.
7. P. Bonnet, J. Gehrke, P. Seshadri. Towards sensor database systems. In Proc. Of 2nd Intl. Conf. on Mobile Data Management, 2001.
8. J. Chen, D.J. DeWitt, F. Tian, Y. Wang. NiagaraCQ: A scalable continuous query system for internet databases. In proc of ACM SIGMOD, 2000.
9. D. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, S. Zdonik. Aurora: A New Model and Architecture for Data Stream Management. In VLDB Journal (12)2: 120-139, August 2003.
10. D. Carney, U. Çetintemel, A. Rasin, S. Zdonik, M. Cherniack, M. Stonebraker. Operator Scheduling in a Data Stream Manager. In Proc of VLDB, 2003.
11. The STREAM Group. STREAM: The Stanford Stream Data Manager. IEEE Data Engineering Bulletin, March 2003
12. R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, R. Varma. Query Processing Resource Management and Approximation in a Data Stream Management System. In Proc. of CIDR 2003.
13. A. Gilani. Design and implementation of stream operators, query instantiator and stream buffer manager. MS Thesis, CSE Department, UT Arlington, 2003. [online] http://www.cse.uta.edu/research/publications/Downloads/CSE-2003-37.pdf
14. S. Sonune. Design and implementation of windowed operators and scheduler for stream data. MS Thesis, CSE Department, University of Texas at Arlington, 2003. [online] http://www.cse.uta.edu/research/publications/Downloads/CSE-2003-38.pdf
15. Cook, D.J., et al., MavHome: An Agent-Based Smart Home. In Proc of the Conference on Pervasive Computing, 2003.

Analyzing the Genetic Operations of an Evolutionary Query Optimizer

Victor Muntés-Mulero[1], Josep Aguilar-Saborit[1], Calisto Zuzarte[2],
Volker Markl[3], and Josep-L. Larriba-Pey[1]

[1] DAMA-UPC, Computer Architecture Dept., Universitat Politècnica de Catalunya,
Campus Nord UPC, C/Jordi Girona Módul D6 Despatx 117 08034 Barcelona, Spain
vmuntes, jaguilar, larri@ac.upc.edu
http://www.dama.upc.edu

[2] IBM Canada Ltd, IBM Toronto Lab.,
8200 Warden Ave., Markham, Ontario, Canada L6G1C7
calisto@ca.ibm.com

[3] IBM Almaden Research Center, 650 Harry Road,
K55/B1, San Jose, CA, 95139 USA
marklv@us.ibm.com

Abstract. In this paper we analyze the importance of the operations in a genetic programming-based optimizer. Among the several conclusions, we show that crossover operations have a larger impact on the quality of the best obtained execution plan than mutation operations.

1 Introduction

Genetic programming applied to query optimization was first proposed by Stillger et al. in [6]. Stillger presents a first crossover operation which handles QEPs represented by tree structures instead of strings of integers. The idea is further developed in [4], presenting The Carquinyoli Genetic Optimizer, which is a sound genetic optimizer that is able to handle acyclic query graphs.

In this paper we show how genetic operations oriented to query optimization improve the competitiveness of a genetic optimizer. Our study remarks the importance of crossover operations compared to mutation operations, although it also shows that mutation operations are essential to grant quality in the results.

This paper is organized as follows. Section 2 introduces genetic optimization and CGO. In Section 3, we analyze the evolution cost using the different genetic operations in CGO. In sections 4 and 5, we present related work and draw some conclusions.

2 Genetic Programming in Query Optimization

Inspired by the principles of natural selection, genetic programming performs operations on the members of a population. Each member in the population represents a path to achieve a specific objective and has an associated cost.

D. Bell and J. Hong (Eds.): BNCOD 2006, LNCS 4042, pp. 240–244, 2006.

Starting with an initial population containing a known number of members, usually created from scratch, two operations are used to produce new members in the population: *crossover operations*, which combine properties of the existing members in the population and *mutation operations*, which introduce new properties into the population. In order to keep the size of the population constant, a third operation, usually referred as *selection*, is used to discard the worst fitted members, using a fitness function. This process generates a new population, also called generation, that includes both the old and the new members that have survived to the selection operation. This is repeated iteratively until a *stop condition* ends the execution. Once the stop criterion is met, we take the best solution from the final population. Query optimization can be reduced to a search problem where the DBMS needs to find the optimum query execution plan (QEP) in a vast search space. Each QEP can be considered as a possible solution for the problem of finding a good access path to retrieve the required data. Therefore, every member in the population is a valid QEP. In this paper we use CGO since, in our understanding, it is the most complete and tested genetic programming-based optimizer.

CGO assumes that a QEP is the typical directed data flow graph. The physical implementations of the operations used in the QEP are called plan operations. CGO includes the basic plan operations, which are typically used in most commercial DBMSs: (i) Scan Operations: *sequential scan* and *index scan*; (ii) Join Operations: CGO allows for the three basic join implementations: *Hash Join*, *Nested-Loop Join* and *Merge-Scan Join*; (iii) Other Operations: Besides scan and join operations, CGO allows for two more operations: Sorting and materializing (Temp Operator). All these QEP operations implicitly include their corresponding selection and projection operations.

Regarding genetic operations, crossover operations randomly choose two QEPs of the population and produce two new trees preserving two subtrees from the parent plans. CGO includes four different kinds of mutation: (i) **Swap (S):** A join operation is randomly selected and its input relations are swaped. S is specially useful if we take into account that join operations are not symmetric (i.e. the *hybrid hash join*); (ii) **Change Scan (CS):** CGO randomly chooses a scan operation and changes the scan method if indexes are available; (iii) **Change Join (CJ):** The optimizer randomly chooses a join operation and it changes its implementation to one of the other two available implementations; (iv) **Random Subtree (RS):** A subtree S from a QEP chosen at random is selected. The remaining join operations, not included in S, are selected in random order until we have inserted all of them, and therefore, a new and complete QEP is created. Further details of CGO can be found in [4].

3 CGO Analysis

This section studies the evolution of the best cost obtained. We use 200 members in the populations and 100 generations per execution. We perform 26 crossover

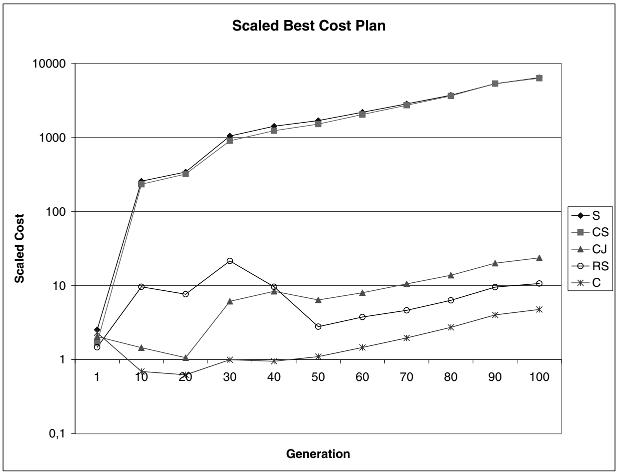


Fig. 1. Scaled best cost evolution vs. the combination of all the genetic operations

operations and 48 mutation operations (12 of each type) per generation, thus generating 100 new QEPs per iteration. We analyze the cost evolution for the best QEP in each generation. We use CGO to optimize each query using 6 different policies. The first five policies consist in applying only one type of mutation or crossover operation during the whole optimization. The last policy combines all the genetic operations. We perform 150 executions involving the random creation of 5 different databases, creating a random query for each one and executing the 6 policies 5 times with queries involving 20 relations. In [3], the experiment is repeated for 30 and 50 relations to analyze the impact of the number of relations involved in the query.

Figure 1 shows, for each of the first five policies, the average best cost divided by the average best cost obtained with the combined use of the five policies. The combination of all the genetic operations always leads in average to QEPs with associated costs several times lower than the other approaches. By nature, CS and S cannot solve critical structural shortcomings such as inefficient join order. Therefore, their application without being combined with other genetic operations results in very high-costed QEPs. Using CJ the best cost usually decreases very fast during the first generations, typically discarding *Nested-Loop Join* operations not using indexes, but it quickly converges yielding QEPs with associated costs which usually are about one order of magnitude higher than the combination of operations. RS outperforms CJ although, after the first generations, it presents a slow convergence compared to other approaches. Crossover operations on their own (C) show a fast convergence, although after the first generations the quality of the results is generally several times higher than the cost of the best QEP yielded by the combination of all the genetic operations.

4 Related Work

The first genetic technique, applied to query optimization, was proposed in [1], where the amount of information per plan was very limited since plans were transformed to chromosomes. Stillger et al. [6] proposed an optimizer based on genetic programming that directly uses QEP as the members in the population. The idea is further developed in [4], where CGO is presented, introducing the ability to handle cyclic query graphs and mutation operations that also use trees as data structures. CGO is validated against the DB2 UDB optimizer, proving that, for large join queries, it can outperform the greedy algorithm used by the commercial DBMS when the memory resources are exhausted due to the size of the search space. Other analysis of CGO can be found in [3]. Also, several variants of *random walk* algorithms have been proposed in [2, 5, 7].

5 Conclusions

We present an analysis of the cost evolution of the QEPs generated by different genetic operations in CGO. Our main conclusions are as follows. First, in the multidimensional search space, each type of mutation usually exploits one dimension. The combined use of all the mutations opens the genetic optimizer to the traversal of the whole search space. Second, the percentage of operations of each type used has an effect on the quality of the best plan. Although crossover operations are in general more powerful, the use of mutation operations is necessary to improve the quality of the optimizer as a whole. As a last conclusion and most important, some dimensions of the search space are better than others to obtain execution plans close to optimum. The combination of some mutations exploring orthogonal dimensions does not necessarily lead to improvements which are the addition of their independent gains, although they may be better.

References

1. Kristin Bennett, Michael C. Ferris, and Yannis E. Ioannidis. A genetic algorithm for database query optimization. In Rick Belew and Lashon Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 400–407, San Mateo, CA, 1991. Morgan Kaufman.
2. Y. E. Ioannidis and Y. Kang. Randomized algorithms for optimizing large join queries. In *SIGMOD '90: Proc. of the 1990 ACM SIGMOD international conference on Management of data*, pages 312–321, New York, NY, USA, 1990. ACM Press.
3. Victor Muntes, Josep Aguilar, Calisto Zuzarte, Volker Markl, and Josep Lluis Larriba. Genetic evolution in query optimization: a complete analysis of a genetic optimizer. Technical Report UPC-DAC-RR-2005-21, Dept. d'Arqu. de Comp. Universitat Politecnica de Catalunya (http://www.dama.upc.edu), 2005.
4. V. Muntes-Mulero, J. Aguilar-Saborit, C. Zuzarte, and J-L. Larriba-Pey. Cgo: a sound genetic optimizer for cyclic query graphs. In *Proceedings of the International Conference on Computer Science (To be published)*, May 2006.

5. Michael Steinbrunn, Guido Moerkotte, and Alfons Kemper. Heuristic and randomized optimization for the join ordering problem. *VLDB Journal: Very Large Data Bases*, 6(3):191–208, 1997.
6. M. Stillger and M. Spiliopoulou. Genetic programming in database query optimization. In J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 388–393, Stanford University, CA, USA, 28–31 July 1996. MIT Press.
7. Arun Swami and Anoop Gupta. Optimization of large join queries. In *SIGMOD '88: Proceedings of the 1988 ACM SIGMOD international conference on Management of data*, pages 8–17, New York, NY, USA, 1988. ACM Press.

An Evidential Approach to Integrating Semantically Heterogeneous Distributed Databases

Xin Hong, Sally McClean, Bryan Scotney, and Philip Morrow

School of Computing and Information Engineering, Faculty of Engineering,
University of Ulster, Cromore Road, Coleraine, BT52 1SA, Northern Ireland
{x.hong, si.mcclean, bw.scotney, pj.morrow}@ulster.ac.uk

Abstract. We present an evidential scheme for integrating semantically heterogeneous aggregates, stored as summary tables, of distributed databases. The ontology of discourse at which to carry out the integration of aggregates is the common ontology derived from local ontologies in the form of classification schemes. The distributed aggregation combination operator *DAggCom* is developed to combine aggregates from local databases and requires minimal communication with local data sources.

1 Introduction

In this paper we are concerned with the integration of aggregate views of distributed databases that use heterogeneous classification schemes, where local ontologies in the form of classification schemes, may be mapped onto a global ontology. Aggregates stored as summary tables are commonly used for summarizing information held in very large databases [4]. At the raw data level, the integration of aggregates may be achieved by simply appending the data sets to each other, aggregating and recalculating. However, it is cumbersome and costly, particularly because of the inaccessibility, or expense of communicating the original data.

This paper extends our previous work [3, 6] regarding aggregation of imprecise and uncertain datasets, to integration of aggregates from distributed databases. We propose mass function models for representing aggregate views using different local classification schemes and extend the mass function models onto the common ontology derived from the local ontologies. A new aggregate combination operator *DAggCom* is developed to integrate semantically heterogeneous aggregate views from distributed sources. *DAggCom* also takes into account weights of different datasets, where each site is weighted proportional to the "credibility" of its beliefs. The combined aggregate presents the summary of the distributed database on the common ontology. Distributed data sites need only pass on aggregates and weights to the operational site, which means that the communication between local data sources and the operational site is kept minimal; aggregation computation is therefore efficient.

2 Semantically Heterogeneous Aggregate Views

Notation 1. A set of all possible values that an attribute A can be assigned forms its *domain*, $D = \{v_1, \ldots, v_k\}$. The elements of D are grouped to give categories (classes)

D. Bell and J. Hong (Eds.): BNCOD 2006, LNCS 4042, pp. 245 – 249, 2006.

of the *classification schemes*. For *aggregate data views* V^r, r = 1, …, m, D is partitioned into categories $\{c_1^r, ..., c_{T_r}^r\}$, with the *cardinality* of category c_t^r, denoted by n_t^r, t = 1,…, T_r. Here T_r is the number of categories in the classification for attribute A in the aggregate view V^r. N^r is the cardinality of A in V^r, $N^r = \sum_{t=1}^{T_r} n_t^r$.

Definition 1. An *ontology* is defined as the Cartesian-product of a number of attributes, along with their corresponding classifications. An ontology, in the form of classification schemes, is a set of classes, $C = \{c_1, …, c_m\}$.

In a distributed database environment, ontologies of local databases may be mapped to a global ontology.

Definition 2. Q local databases in a distributed database produce Q aggregate views based on Q different classification schemes, $C_q = \{c_1^q, ..., c_{I_q}^q\}$, q = 1, …, Q. Denoting the global ontology by G = $\{g_1, …, g_n\}$, the *correspondence table* between Q local ontologies and the global ontology is defined as

$$CT(G, C_1, ..., C_Q) = \{< g_k, c_{i_1}^1, ..., c_{i_Q}^Q >\}$$

where $g_k \in c_{i_q}^q$; k = 1, …, n; $c_{i_q}^q \in C_q$; i_q = 1, …, I_q; q =1, …, Q.

Definition 3. From Q local ontologies, $C_q = \{c_1^q, ..., c_{I_q}^q\}$ (q = 1, …, Q), the *common ontology* denoted C, is derived and defined by the union of C_q, q = 1, …, Q. The common ontology C in the form of classification schemes is the set of classes,

$$C = \{c_j, j = 1, …, J\}$$

where (a) if $i \neq j$, then $c_i \neq c_j$, i, j = 1, …, J; (b) $C_q \subseteq C$, q = 1, …, Q; (c) $1 \leq J \leq \sum_{q=1}^{Q} I_q$.

The common ontology is the ontology of discourse at which to carry out the integration of aggregate views.

3 Evidential Models of Aggregate Views

The Dempster-Shafer theory of evidence (DS theory) [1, 2, 5, 7] is a generalization of traditional probability theory, in which mass functions are used to quantify our belief degree in various propositions. We denote the domain of an attribute by Θ. The mass function is defined on subsets of Θ as follows.

Mass function: A mass function m is a mapping: $2^\Theta \rightarrow [0, 1]$, satisfying:

1. $m(\phi) = 0$, where ϕ is an empty set
2. $\sum_{A \subseteq \Theta} m(A) = 1$.

In what follows, we propose mass function models of aggregate views. In a distributed database consisting of Q local databases, the qth aggregate view (associated with attribute A) uses the classification scheme $C_q = \{c_1^q, ..., c_{I_q}^q\}$ with associated cardinality n_i^q of c_i^q, $i = 1, …, I_q$, and the cardinality N^q of A, for $q = 1, …, Q$. With each class c_i^q is associated a mass function m_i^q, $m_i^q = \frac{n_i^q}{N^q}$. We represent the qth aggregate view as the set S_q, $S_q = \{< c_i^q, m_i^q >, i = 1, ..., I_q\}$.

Suppose that the common ontology, $C = \{c_1, …, c_J\}$, is derived from the Q local ontologies. For each aggregate view, we introduce the trivial extension of the set S_q

$$\tilde{S}_q = \{< c_j, \tilde{m}_j^q >, j = 1, ..., J\},$$

where

$$\tilde{m}_j^q = \begin{cases} m_i^q & \text{if } \exists i \in \{1, ..., I_q\} \text{ for which } c_j = c_i^q. \\ 0 & \text{otherwise.} \end{cases}$$

It can obviously be seen that $\tilde{m}^q$ is also a mass function.

4 Integrating Semantically Heterogeneous Aggregate Views

We now develop a new aggregation combination operator for distributed databases, for integrating semantically heterogeneous aggregate views to which the mass distributions have already been assigned. We integrate the aggregates to have the mass distribution on the common ontology derived from the local databases.

Consider that a distributed database contains Q local databases, each of which uses a different ontology $C_q = \{c_1^q, ..., c_{I_q}^q\}$ for $q = 1,…, Q$. The aggregate view of a local database is represented by $S_q = \{< c_i^q, m_i^q >, i = 1, ..., I_q\}$ with associated weight W_q. The common ontology derived from the Q local databases is denoted by $C = \{c_j, j = 1, …, J\}$. $\tilde{S}_q = \{< c_j, \tilde{m}_j^q >, j = 1, ..., J\}$ is the extension of S_q on the common ontology. The Q aggregate views in the extension form may then be combined using the aggregation combination operator *DAggCom* defined as follows:

$$DAggCom(\tilde{S}_q, q = 1, ..., Q) = \{< c_j, m_j >, j = 1, ..., J\}$$

where $m_j = \sum_{q=1}^{Q} (\frac{W_q}{\sum_{q=1}^{Q} W_q} \times \tilde{m}_j^q)$.

The output of the *DAggCom* operator is a set of the common ontology categories along with the associated mass functions.

As an illustrative example, the EU Labour Force Survey data for the *job status* attribute is considered. The global ontology is given as a set of possible attribute values, {*full-time self-employed; full-time employees; part-time employment; on government training scheme; unemployed (receiving benefits); in full-time education; economically inactive but not in full-time education*}. Two local classification schemes along with aggregates are produced from two local databases.

Scheme 1: {<full-time employment, 217>, <part-time employment including government training schemes, 73>, <unemployed, 14>, <economically inactive, 196>}

Scheme 2: {<in employment, 540>, <unemployed, 33>, <in full-time education, 370>, <economically inactive but not in full-time education, 30>, <on government training scheme, 27>}

From the two local ontologies, the common ontology can be derived as follows:

{c_1: full-time employment; c_2: part-time employment, including government training schemes; c_3: unemployed; c_4: economically inactive; c_5: in employment; c_6: in full-time education; c_7: economically inactive but not in full-time education; c_8: on government training scheme}.

Table 1. The common classification scheme for the *job status* attribute

Common ontology categories	$\tilde{m}^1$	$\tilde{m}^2$	m	Common ontology categories	$\tilde{m}^1$	$\tilde{m}^2$	m
c_1	0.434	0.000	0.145	c_5	0.000	0.540	0.360
c_2	0.146	0.000	0.049	c_6	0.000	0.370	0.247
c_3	0.028	0.033	0.031	c_7	0.000	0.030	0.020
c_4	0.392	0.000	0.131	c_8	0.000	0.027	0.018

Table 1 displays the extended mass function distributions of the two aggregate views for the *job status* attribute, $\tilde{m}^1$ and $\tilde{m}^2$. Taking the cardinality as the weight, using *DAggCom* the mass distribution m associated with the integrated aggregate on the common ontology is obtained and shown in Table 1.

Since we only need aggregate views and cardinalities from distributed data sites, aggregate combination, which requires minimal communication between local data sources and the operational site, is therefore computationally efficient. This work will be extended further to integration of aggregate views that may include summary data on non-partitions of attribute domain.

References

1. Dempster, A. P.: A Generalisation of Bayesian Inference. J. Royal Statistical Soc., Series B, Vol. 30 (1968) 205 – 247.
2. Guan, J. W., Bell, D. A.: Evidence Theory and Its Applications: Vol 1. Studies in Computer Science and Artificial Intelligence, Elsevier, North-Holland (1991).
3. McClean, S., Scotney, B.: Using Evidence Theory for the Integration of Distributed Databases. Int'l J. Intelligent Systems, Vol.12 (1997) 763 – 776.

4. McClean, S., Scotney, B., Greer, K.: A Scalable Approach to Integrating Heterogeneous Aggregate Views of Distributed Databases. IEEE Trans. Knowledge and Data Eng., Vol. 15 No. 1 (2003) 232 – 236.
5. McClean, S., Scotney, B., Shapcott, M.: Using Background Knowledge in the Aggregation of Imprecise Evidence in Databases. Data & Knowledge Engineering, Vol. 32 (2000) 131 – 143.
6. Scotney, B., McClean, S.: Database Aggregation of Imprecise and Uncertain Evidence. Int'l J. Information Sciences, Vol. 155 (2003) 245 – 263.
7. Shafer, G.: The Mathematical Theory of Evidence. Princeton University Press (1976).

Interoperability and Integration of Independent Heterogeneous Distributed Databases over the Internet

Bryan Scotney, Sally McClean, and Shuai Zhang

[1] School of Computing and Information Engineering, University of Ulster,
Coleraine, Northern Ireland, UK
{bw.scotney, si.mcclean, zhang-s1}@ulster.ac.uk

Abstract. A key challenge for independent databases that are distributed over the Internet is to provide mechanisms for interoperability to facilitate resource discovery, access, distributed processing and integration. Additional challenges arise when there is semantic heterogeneity of the underlying ontologies associated with the databases. This paper develops a mechanism that facilitates interoperability of independent distributed database applications that is useful in situations where databases have evolved separately but where post hoc semantic mappings between schema are possible. Such functionality can be obtained by storing metadata in a common format, such as RDF, thus allowing the databases to be discovered and accessed by browsing registries that contain details of the database objects. Once interoperability has been achieved, we describe a flexible user-centric method for integration and knowledge discovery from semantically heterogeneous data, based on the specification of ontology mappings from distributed local data sources to a derived ontology of discourse.

1 Background

Improved interoperability of Web-based information systems and increased automation in information processing were the original motivations for the development of the Semantic Web [4]. The Semantic Web brings, in particular, challenges to the integration of independent heterogeneous distributed databases over the Internet. The process of data integration [1] can be used to make explicit the often rich latent information in the data, and hence provides benefits in terms of knowledge acquisition. Thus, the provision of efficient automated integration facilities is an important aspect of using distributed database technology for knowledge discovery [5]. For large-scale data resources it is necessary that the contributing databases can be accessed and data retrieved in a decentralised way according to their own constraints, and that heterogeneity in ontologies and content is reconciled. Also, user-centric integration algorithms are desirable for matching users' query requirements.

In order to facilitate development of interoperability and automation, this paper develops mechanisms that facilitate discovery, access, integration and processing of independent heterogeneous distributed database applications over the Internet. A conceptual data and metadata model (including ontology mappings) is introduced into the logical layer of the three-tier information architecture system so as to facilitate the interoperability. The key to the model development is that the conceptual model is

D. Bell and J. Hong (Eds.): BNCOD 2006, LNCS 4042, pp. 250 – 253, 2006.

flexible enough to accommodate data and metadata stored in a range of physical layers that can facilitate users to extract components in a variety of formats for presentation and publication. Registries form an essential part of the logical layer and are used for indexing data and metadata objects, enabling them to be discovered, accessed and extracted [3]. Once interoperability has been achieved, we describe a flexible method for data integration and knowledge discovery.

2 Interoperability Mechanisms for Information Exchange

The first step towards achieving interoperability between database application systems is to transform the users' required data and metadata that various systems serve into objects that are described using a common format (e.g., RDF). For the successful exchange of metadata, mappings are needed from the terminologies used by each individual database system to the common format. We will deal principally with data objects of database tables, metadata objects of classifications, and mappings between classifications. However, other metadata objects are required, relating to logistical information about data providers and data access.

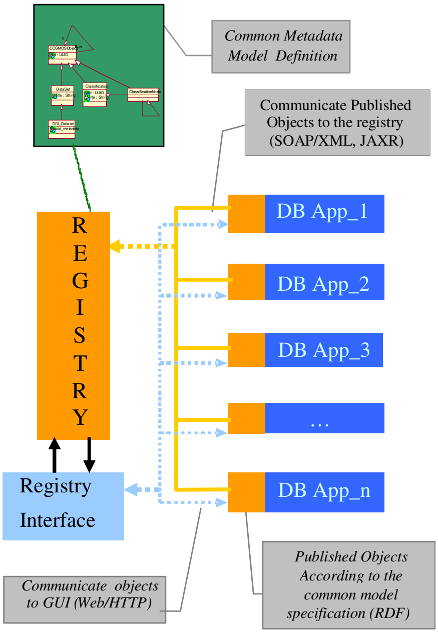

1
0..n
COSMOSObject
Id : UUID
DataSet
Title : String
Classification
Id : UUID
Title : String
ClassificationNode
DDI_Dataset
uuid_metadata
Common Metadata Model Definition
Communicate Published Objects to the registry (SOAP/XML, JAXR)
R E G I S T R Y
DB App_1
DB App_2
DB App_3
...
DB App_n
Registry Interface
Communicate objects to GUI (Web/HTTP)
Published Objects According to the common model specification (RDF)

Fig. 1. System architecture to enable keyword searches

The common browser facility should be in a position to keep track of which system has exported the metadata objects, and once queries are posed, it should thus be able to re-direct the requests for the data to the appropriate systems. Thus, the user will be unaware of the fact that, although metadata are browsed in a single application, the queries on the associated data are answered by (potentially) different distributed systems. In order to achieve efficient browsing and searching capabilities, an intermediate indexing level consisting of registries is required, and browsing may thus be achieved through a common application (e.g., ebXML Registry browser). Registries are used to maintain information related to issues such as provision of, type of, and access to datasets. Registries are central to interoperability of distributed database systems, being required for registration of data publishers, published information, querying, and information exchange.

To enable registry browsing, it is necessary to establish classification schemes so that the data and metadata objects can be classified. The classification of data and metadata objects is the principal mechanism by which the registry can be queried. Classification is carried out by a cataloguing service. Any data or metadata object must be classified by the following three mandatory classification schemes.

(1) ObjectType scheme: the way of identifying object types in an ebXML registry.
(2) DataFormat scheme: to identify the format of the object as published on the web.
(3) Publisher scheme: this identifies the database application system that has published the object.

However, an object may be classified by additional optional schemes.

Figure 1 shows the architecture of a system that is able to perform keyword searches by using individual database application systems as repositories of data/metadata. In the system, each of the individual applications has two main tasks:

(i) to make its data and metadata objects known to the registry by publishing them in a predefined manner according to the common metadata model specification;

(ii) to transfer these data and metadata objects among the applications, or between the applications and the Registry GUI application.

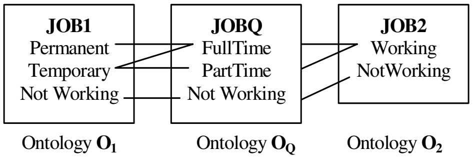
Ideally, all of the individual applications should communicate with each other only through the common metadata model and registry, using their own graphical GUIs. New data and metadata objects can be created from old ones through a sequence of events for exchange and processing of information between publishers.

3 Data Integration and Knowledge Discovery

Based on the mechanisms for interoperability of web systems discussed above, in this section we develop mechanisms for integrating and processing data from independent heterogeneous distributed databases on the Web. We focus on the semantic integration of the ontologies and corresponding data values (summaries in particular).

Definition 1. An ontology is defined as the Cartesian-product of a number of attributes along with their corresponding schema.
Definition 2. Two ontologies are defined to be semantically equivalent if there is a mapping between their respective schema.
Definition 3. An ontology O1 is said to be refined by another ontology O2 if O1 and O2 can be combined to provide a derivable classification finer than O1.

Firstly, the ontology of discourse (OOD) needs to be determined to provide a common ontology that specifies how the local semantics relate to the meaning of the query ontology adopted by the user. Our aim is then to integrate data that are at the finest possible aggregation of the OOD. The OOD is computed by refining the query ontology by the various database ontologies, using the *Ontint* refinement algorithm [5]. The computation may be regarded as metadata integration which is followed by integration of data. The approach is illustrated in the following example:



$$A_{Q1} = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, A_{Q2} = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

The correspondence matrices that relate the query ontology (O_Q) to ontologies O_1 and O_2, are given by A_{Q1} and A_{Q2} respectively. We first refine O_Q with O_1 which gives the new refined ontology O_{Q^*} {Permanent, Temporary∩FullTime, PartTime, NotWorking}; but there is no further possible refinement of O_{Q^*} by O_2. The OOD is therefore O_{Q^*} with a new derived value Temporary∩FullTime. Aggregates will be computed for this value, even though none was present in the original data.

Once metadata integration is complete, we integrate the aggregate data at a level of detail determined by OOD, using the integration algorithm *Aggint* [2]; this algorithm is based on minimisation of the Kullback-Leibler information divergence using the EM algorithm. Here, the respective probabilities for the OOD are $\{\pi_1, \pi_2, \pi_3, \pi_4\}$.

Table 1. Aggregates data (cardinalities) for the concept JOB

JOB1		**JOBQ**		**JOB2**	
Permanent	170	FullTime	60	Working	125
Temporary	37	PartTime	10	NotWorking	25
NotWorking	23	NotWorking	30		

$$A_{Q1^*} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, A_{Q2^*} = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

Using Table 1, and new correspondence matrices A_{Q1^*} A_{Q2^*}, the integrated summaries are obtained by iterative solution of the following equations:

$$\pi_1 = (170 + \frac{60 * \pi_1}{\pi_1 + \pi_2} + \frac{125 * \pi_1}{\pi_1 + \pi_2 + \pi_3}) / 480; \qquad \pi_2 = (\frac{37 * \pi_2}{\pi_2 + \pi_3} + \frac{60 * \pi_2}{\pi_1 + \pi_2} + \frac{125 * \pi_2}{\pi_1 + \pi_2 + \pi_3}) / 480;$$

$$\pi_3 = (10 + \frac{37 * \pi_3}{\pi_2 + \pi_3} + \frac{125 * \pi_3}{\pi_1 + \pi_2 + \pi_3}) / 480; \qquad \pi_4 = (23 + 30 + 25) / 480.$$

The converged solutions are π_1=0.688, π_2=0.030, π_3=0.120, π_4=0.162, with corresponding integrated cardinalities given by $N^*\pi_i$ (N=480 is the total cardinality).

4 Conclusion and Future Work

We have discussed interoperability mechanisms to facilitate discovery, access and processing of independent distributed databases and have described algorithms for integration of the corresponding ontologies and data. Further work will extensively evaluate the approach and develop new algorithms for user-centric integration.

References

1. Dao, S., Perry, B. Applying a data miner to heterogeneous schema integration. KDD-95, AAAI Press, Montreal, Canada, 1995, pp. 63-68.
2. McClean, S.I., Scotney, B.W., Greer, K.R.C. A Scalable Approach to Integrating Heterogeneous Aggregate Views of Distributed Databases. *IEEE Transactions on Knowledge and Data Engineering,* 2003, 15(1), pp. 232-236.
3. Nelson, C. Use of Metadata Registries for Searching for Statistical Data. 14th IEEE SSDBM, 2002, pp. 232-235.
4. Payne, T., Lassila, O. Semantic Web Services, IEEE Intelligent Systems, 2004, pp. 14-15.
5. Scotney, B.W., McClean, S.I. Knowledge Discovery from Databases on the Semantic Web. 16th IEEE SSDBM, 2004, pp. 333-336.

Trust Obstacle Mitigation for Database Systems

Victor Page[1], Robin Laney[2], Maurice Dixon[1], and Charles Haley[2]

[1] Department of Computing,
Communications Technology and Mathematics,
London Metropolitan University, 31 Jewry Street, London, EC3N 2EY
{Vic.Page, M.Dixon}@Londonmet.ac.uk
[2] Department of Computing
The Open University, Walton Hall, Milton Keynes, MK7 6AA
{R.C.Laney, C.B.Haley}@Open.ac.uk

Abstract. This paper introduces the Trust Obstacle Mitigation Model (TOMM), which uses the concept of trust assumptions to derive security obstacles, and the concept of misuse cases to model obstacles. The TOMM allows a development team to anticipate malicious behaviour with respect to the operational database application and to document a priori how this malicious behaviour should be mitigated.

1 Introduction

The Lowell Database Research Self-Assessment [1] discusses "trustworthy systems that safely store data, protect it from unauthorized disclosure, protect it from loss, and make it always available to authorized users". It also suggests that "the information management community should play a central role in addressing these needs and enhancing DBMSs with mechanisms to support these capabilities".

This work reports progress on addressing these needs at the application level without the need to enhance the DBMS. To do this we have derived a model, the Trust Obstacle Mitigation Model (TOMM), for analysing the detection and mitigation of security obstacles within a database system. An obstacle is something that, should it occur, will invalidate a trust assumption and result in a deviation between a use case and the realisation of the use case in the operational system. Security obstacles are caused by malicious agents, external to the system, that might destroy, reveal, modify, or block information assets. This is in line with the security requirements of confidentiality, integrity, availability, and authentication as presented in [2] and given the acronym CIAA.

The TOMM draws on three existing concepts, obstacle analysis [3, 4, 5], trust assumptions [6] and misuse cases [7, 8]. Trust assumptions are used to derive obstacles and misuse cases are used to provide a diagrammatic and textual representation of an obstacle. The contribution of this work is the bringing together of these concepts coupled with a 'traffic light' approach to ranking obstacles and their consequences.

The paper is structured as follows. Section 2 provides an overview of the TOMM. Finally conclusions are presented in Section 3 along with future work.

D. Bell and J. Hong (Eds.): BNCOD 2006, LNCS 4042, pp. 254 – 257, 2006.

2 The Trust Obstacle Mitigation Model

Fig. 1 presents an activity diagram for the TOMM. The swim lanes show the three phases of the TOMM Elicit Use Cases, Derive Obstacles, and Derive Mitigations. The swim lanes are present to show that each phase could be carried out in a different facilitated workshop.

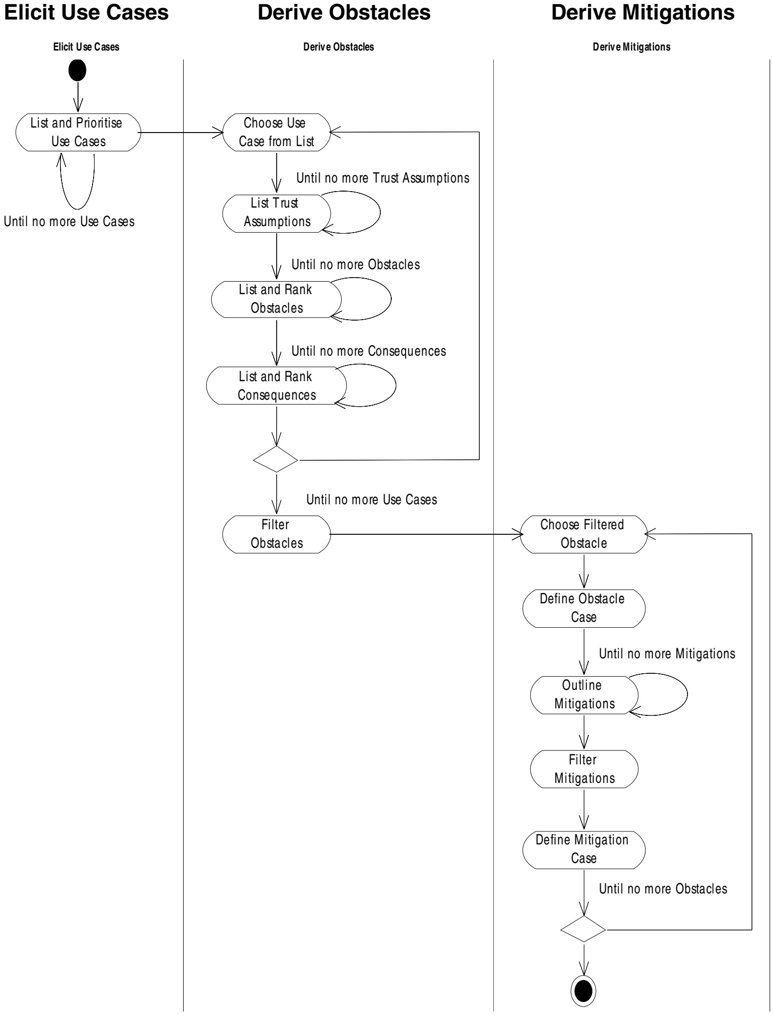


Fig. 1. Activity diagram of the TOMM

The above is not a strictly sequential process and there is an implied iteration in some of the activities. Specifically the development team may wish to re-prioritise the use case list either at the end of the Derive Obstacles phase, or at the end of the Derive Mitigations phase.

The phases and activities of the TOMM allow a development team to anticipate malicious behaviour with respect to the operational system and document a priori how this malicious behaviour should be mitigated. This is achieved by requiring the development team to reason about how the trust assumptions on which the database system will be built could be undermined.

The following lists alphabetically the definitions of the concepts and terminology that are used within the TOMM:

- **Consequence:** A consequence defines what would happen to the operational system if a trust assumption is invalidated due to the manifestation of an obstacle. Consequences are given a RAG code that signifies their severity to the operational system.
- **Filter:** Filter means to choose based on available data. For obstacles we choose those obstacles that need to be mitigated, based on the RAG codes. For mitigations we choose the most effective mitigation based on an overview of the mitigation along with its estimated cost and duration.
- **Mitigation:** A mitigation is something, that should it be implemented, will counter the effect of an obstacle.
- **Mitigation Case:** A Use Case that shows what should be done to counter the effect of malicious behaviour on the operational system.
- **Obstacle:** An obstacle is something that, should it occur, will invalidate a trust assumption and result in a deviation between a use case and the realisation of the use case in the operational system. Obstacles are given a RAG code that signifies the likelihood of them occurring.
- **Obstacle Case:** A Use Case that shows the effect of malicious behaviour on the operational system. The main function of the obstacle case is to decide and document a priori how the operational system would react to malicious use. This is based on the concept of a Misuse Case.
- **RAG Code:** RAG codes form an intuitive ‘traffic light’ approach to ranking obstacles and their consequences. RAG is an acronym for Red, Amber, and Green. The RAG codes are classified as follows;
  - **R:** signifies either a high likelihood of an obstacle occurring or a fatal consequence should an obstacle occur.
  - **A:** signifies either a medium likelihood of an obstacle occurring or a non-fatal consequence should an obstacle occur.
  - **G:** signifies either little likelihood of an obstacle occurring or low negative consequence should an obstacle occur.
- **Security Obstacle:** An obstacle that is caused by the malicious behaviour of an external agent (human or machine).
- **Trust Assumption:** Documents the way in which a use case, when realised in the operational system, can be trusted to have certain stated properties and/or behaviour.
- **Use Case:** A representation by diagram and text of a sub-set of the database system functionality.

3 Conclusions and Future Work

Use Case diagrams are at the heart of the Trust Obstacle Mitigation Model. They provide a simple yet powerful diagrammatic representation of database system requirements at a level of granularity appropriate for reasoning about security obstacles in facilitated workshops. The TOMM is visually intuitive and it can be adopted by projects where use case modeling would normally be applied. Also the TOMM provides an intuitively direct approach to ranking obstacles and their consequences – via the use of RAG codes.

The TOMM can be improved by deriving taxonomies of trust assumptions, obstacles, consequences and mitigations, along with heuristics to support their use. We have incorporated trust assumptions in the TOMM, which are assumptions by the development team that a requirement, when realised in the operational system, will cause that system to have certain stated properties and/or behaviour [6]. This suggests that the obstacles caused by this trust being misplaced can be classed as trust obstacles. The incorporation of the formal semantics described in UMLSec [9] will provably show that the model is consistent, correct and optimal for its purposes. Future work will focus on these improvements. A tool will also be developed to support the model.

References

1. Abiteboul, S., Agrawal, R., Bernstein, P., et al: The Lowell Database Research Self-Assessment. Communications of the ACM, Vol. 48. No. 5. (2005) 111-118
2. Stallings, W.: Business Data Communications 5th edn. Pearson Prentice Hall, Upper Saddle River USA (2005)
3. Page, V., Dixon, M., and Bielkowicz, P.: Object-Oriented Graceful Evolution Monitors: In: Konstantas, D., Leonard, M. and Patel, S. (eds): Proceedings of the Ninth International Conference on Object-Oriented Information Systems, University of Geneva Geneva Switzerland (2003) 46-59
4. Anton, A.: Goal Identification and Refinement in the Specification of Software-Based Information Systems. Ph.D. Thesis, College of Computing Georgia Institute of Technology (1997)
5. Lamsweerde, A., Letier, E.: Integrating Obstacles in Goal-Driven Requirements Engineering. ICSE'98 – 20th International Conference on Software Engineering, Kyoto Japan (1998) 53-62
6. Haley, C., Laney, R., Moffett, J., Nuseibeh, B.: (2004), The Effect of Trust Assumptions on the Elaboration of Security Requirements. Proceedings of the 12th International Requirements Engineering Conference (RE'04), Kyoto Japan (2004) 102-111
7. Alexander, I.: Misuse Cases: Use Cases with Hostile Intent. IEEE Software, Vol. 20, Issue 1. (2003) 58-66
8. Sindre, G., and Opdahl, A.: Eliciting Security Requirements by Misuse Cases. Proceedings of the 37th International Conference on Technology Object-Oriented Languages and Systems, Sydney Australia (2000) 120-131
9. Jürjens, J.: UMLsec: Extending UML for Secure Systems Development. In Jézéquel, J., Hussmann, H. and Cook, S. (eds): Proceedings of the 5th International Conference on The Unified Modeling Language, Dresden Germany (2002) 412-425

Towards a More Reasonable Generalization Cost Metric for K-Anonymization

Zude Li, Guoqiang Zhan, and Xiaojun Ye*

Institute of Information System and Engineering
School of Software, Tsinghua University, Beijing, 100084, China
{li-zd04, zhan-gq03, yexj}@mails.tsinghua.edu.cn

Abstract. A k-anonymity model contains an anonymity cost metric mechanism, which is critical for the whole k-anonymization process. The existing metrics cannot sufficiently identify the real cost on tabular microdata anonymization. We define a new cost metric that can be used for k-anonymization with the data generalization approach. The metric is more reasonable than the existing ones as it considers generalization range and range ratio rather than generalization height or height ratio, and the contribution of an attribute to the whole tuple rather than the amount of suppression cells. It can be used in most k-anonymity models for computing more precise anonymity costs.

1 Introduction

An individual represented as a record in a database might be re-identified by joining the released data with another publicly available database. To reduce the risk of this type of attacks, *k-anonymity* is proposed as a privacy protection model against individual re-identification in microdata publishing [9, 12]. Many instances illustrating such attacks are listed in literature such as [9, 12, 7, 4, 5, 2, 8], which are the motivations for most *k-anonymity* models introduced in the past several years. In general, *k-anonymity* means that one can only be certain that a value is associated with one of at least *k* values, or in a *k-anonymized* dataset, each record is indistinguishable from at least *k-1* other records with respect to certain identifying attributes [7].

Any *k-anonymity* model or mechanism contains an anonymity cost metric (i.e. generalization cost metric with the data generalization approach) as the preference criteria for data anonymization in a dataset (i.e. an original table). For instance, the precision metric *Prec* in [11], the cost function *Cost* in [2], etc. Such a preference criteria computing mechanism should be more important since it acts in the whole anonymization process as the base of defining "optimal" for *k-anonymity* solutions derived from the original table. Generally, a derived table with the minimal anonymity cost is always the "optimal" *k-anonymity* solution [9, 12, 11].

* Funded by National Basic Research Program of China, Project No.2002CB312006 and National Natural Science Foundation of China, Project No.60473077.

D. Bell and J. Hong (Eds.): BNCOD 2006, LNCS 4042, pp. 258–261, 2006.

The contributions of this paper are to analyze and categorize the existing anonymity cost metrics, and to define a more reasonable generalization cost metric taking into account several critical factors that can be used for most *k-anonymity* models. The paper is organized as follows: in Section 2 we briefly discuss and categorize several existing cost metric mechanisms. We then analyze some critical factors for anonymity cost metrics and define a more reasonable generalization cost metric in Section 3. Finally, we gives a short conclusion and a brief prospect for future work in Section 4.

2 Related Work

In a relational table, a set of identifying attributes is called a *Quasi-Identifier attribute set* ($\mathcal{QI}$), as they can be joined with external information to uniquely re-identify at least one individual in Ω with sufficiently high probability (Ω is a large population) [9, 5, 4]. A released table is said to adhere to *k-anonymity* if each released record has at least *k-1* other records whose values are indistinct over $\mathcal{QI}$. Suppose the approach to implement *k-anonymity* on $\mathcal{QI}$ attributes is data generalization. Each attribute in a table has a value domain containing all values for the attribute. Given a domain, it is possible to construct a more “general” domain in a variety of ways, such as the domain generalization hierarchy in [9, 11].

Current generalization cost metrics can be classified into four types: (a) on generalization hierarchy; (b) on the amount of suppression cells; (c) on partition, and (d) on entropy.

[**On generalization hierarchy**] Typically, the *Prec* calculation in [11] is one of this type, as $Prec(RT) = 1 - \frac{\sum_{i=1}^{n}\sum_{j=1}^{m}\frac{\mathcal{H}(A_{ij},\ A'_j)}{\mathcal{H}_{A_j}}}{n \times m}$, where $\mathcal{H}$ returns the *height* of a generalization hierarchy or a generalization relation, *Prec*(RT) refers to the generalization cost on all m $\mathcal{QI}$ attributes in n tuples. As it is defined, a generalization hierarchies is constructed as part of the preference criteria. The *Cost* function in [2] is also based on the generalization hierarchies of attributes, which on the whole table is the sum of the ratio of the generalization level on an attribute in a tuple to the hierarchy height on the attribute, i.e. $Cost(RT)$ $= \sum_i \sum_j Cost(i,\ j)\ /\ l_j$, where RT denotes the derived table, $Cost(i,\ j)$ denotes the cost of generalizing the jth attribute's value in the ith tuple to the form in RT, l_j indicates the generalization hierarchy height on the jth attribute.

[**On the amount of suppression cells**] In [1], the cost is based on the *Hamming distance* among tuples. For example, the Hamming distance on <82-02-57, M, 14890> and <11-22-42, M, 13092> is 2 since the values of 2 attributes in two corresponding tuples are different. The *diameter* merit in [8] is similar, as $d(S)$ $= Max_{u,v \in S}\ d(u,\ v)$, where S is a set and $d(S)$ denotes the maximal amount of different coordinates between any two elements. Both belong to this type.

[**On partition**] In [6], the cost metric is identified as a classification metric, which is computed as the sum of the individual penalties for each tuple in the table (with n tuples: tup_1, $\cdots$, tup_n): $CM = \frac{\sum_{i=1}^{n} penalty(tup_i)}{n}$, where *penalty* function returns 1 if the tuple (parameter) is suppressed or its class label is not the majority class, others *penalty* returns 0. The *discernibility* metric and the *classification* metric in [3] are also in this type. The *discernibility* metric assigns a penalty (the penalty reflects the fact that a suppressed tuple cannot be distinguished) to each tuple based on how many tuples in the transformed dataset are indistinguishable from it. The *classification* metric assigns no penalty to an unsuppressed tuple if it belongs to the majority class within its induced equivalence class. The normalized average equivalence class size metric C_{AVG} in [4] is similar to the above two metrics.

[**On entropy**] The generalization cost based on entropy computation is a classical measure used in information theory to characterize the purity of data, which is used in the Datafly system [10]. It is described as the anonymity levels between 0 and 1 specifying the minimum *bin* size (similar to k in *k-anonymity*) for every attribute. Attribute values need to be generalized to attain the minimum bin size. The metric for *ℓ-diversity* on sensitive attribute clusters [7], i.e. the condition for a cluster satisfying the *ℓ-diversity* requirement is defined as below: $-\sum_{s \in S} \frac{n(q,s)}{\sum_{s' \in S} n(q,s')} log(\frac{n(q,s)}{\sum_{s' \in S} n(q,s')}) \geq log(\ell)$, where $n(q,s)$ denotes the appearing amount of element s in cluster q. The larger the left side entropy is, the more difficult it is to infer a right element in the cluster q.

3 A More Reasonable Generalization Cost Metric

In the semantics of data generalization for *k-anonymity*, the generalization *range* and the corresponding *range ratio* should be more important than the discussed generalization height and height ratio, because the generalization range expresses the exact generalization loss and the range ratio describes the relative generalization degree of the attribute in the tuple. Two generalization relations with the same generalization height ratio may have very different generalization range ratios. The generalization range ratio expresses the generalization degree on an attribute, which is more closely related to the precise generalization loss metric than the simple generalization range. Furthermore, in a tuple the ratio of the generalization range on an attribute to the whole generalization range presents the influence of the generalization on the attribute to the whole tuple for generalization, which is called *generalization range contribution ratio*. This is a method to express the influence of generalization range on an attribute to the cost metric in the whole tuple through the ranges of other attributes.

Based on the above factors, we define a new generalization cost $\mathcal{GC}$ on a cluster S with sz elements of $m1$ $\mathcal{QI}$ attributes and $m2$ sensitive attributes (sz = $m1$ + $m2$). It can capture the above critical factors for expressing the more precise generalization cost on S. The $\mathcal{GC}$ value on S is calculated as follows.

$$\mathcal{GC} = (\sum_{j=1}^{sz}\sum_{i=1}^{m1} c_i^j \times \frac{R(A_i^j, A_i^{'})}{R_{A_i}} \times \sqrt{\frac{R(A_i^j, A_i^{'})}{\prod_{k=1,k\neq i}^{m1} R(A_k^j, A_k^{'})}}) \times \frac{\delta}{En}$$

$$En = -\sum_{i=1}^{m2}\sum_{s\in S_i}\frac{n(q,s)}{\sum_{s'\in S_i} n(q,s')}\ log(\frac{n(q,s)}{\sum_{s'\in S_i} n(q,s')})$$

where $A_i^{'}$ represents the generalized form of all elements on the ith attribute in the cluster, q is the cluster of sensitive attributes tuples mapping to S_i (the ith sensitive attribute), c and δ are relative balance factors.

4 Conclusion

In this paper, we systemically summarize the existing generalization cost metrics. We then build a new cost metric that is more reasonable than the existing ones. This metric can be used in more general *k-anonymity* models with the data generalization approach for optimal or approximate-optimal *k-anonymization* solution derived from the original table. In the near future, we will put the new metric into several *k-anonymity* models to test its effects by experiments.

References

1. Gagan Aggarwal, Tomas Feder, and etc. Anonymizing tables for privacy protection. *Available: http://theory.standford.edu/ rajeev/privacy.html*, 2004.
2. Gagan Aggarwal, Tomas Feder, and etc. Approximation algorithms for k-anonymity. *Journal of Privacy Technology*, Nov. 2005.
3. Reberto J.Bayardo and Rakesh Agrawal. Data privacy through optimal k-anonymization. *ICDE'05*, 2005.
4. Kristen LeFevre, David J.DeWitt, and Raghu Ramakrishnan. Multidimensional k-anonymity. *Technical Report, Available: www.cs.wisc.edu/techreports/2005/.*
5. Kristen Lefevre, David J.DeWitt, and Raghu Ramakrishnan. Incognito: Efficient full-domain k-anonymity. *SIGMOD'05*, 2005.
6. Vijay S. Lyengar. Transforming data to satisfying privacy constraints. *SIGKDD'02*, 2002.
7. Ashwin Machanavajjhala, Johannes Gehrke, and Daniel Kifer. ℓ-diversity: Privacy beyond k-anonymity. *ICDE'06*, 2006.
8. Adam Meyerson and Ryan Williams. On the complexity of optimal k-anonymity. *PODS'04, France*, 2004.
9. Pierangela Samarati and Latanya Sweeney. Protecting privacy when disclosing information: K-anonymity and its enforcement through generalization and suppression. *Technical Report, SRI Computer Science Lab.*, 1998.
10. Latanya Sweeney. Guaranteeing anonymity when sharing medical data, the datafly system. *Journal of the American Medical Informatics Association*, 1997.
11. Latanya Sweeney. Achieving k-anonymity privacy protection using generalization and suppression. *In International Journal on Uncertaining, Fuzziness and Knowledge -based Systems*, 10(5):571–588, 2002.
12. Latanya Sweeney. K-anonymity: A model for protecting privacy. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10(5):557–570, 2002.

Verification Theories for XML Schema

Suad Alagić, Mark Royer, and David Briggs

Department of Computer Science
University of Southern Maine
Portland, ME 04104-9300
{alagic, mroyer, briggs}@cs.usm.maine.edu

Abstract. XML Schema types and structures are represented as theories of a verification system, PVS, for proving properties related to XML schemas. Type derivations by restriction and extension as defined in XML Schema are represented in the PVS type system using predicate subtyping. Availability of parametric polymorphism in PVS makes it possible to represent XML sequences and sets via PVS theories. Transaction verification methodology is based on declarative, logic-based specification of frame constraints and the actual transaction updates. XML applications, including constraints typical for XML schemas, such as keys and referential integrity, have been verified.

1 XML Schema Types as PVS Theories

We describe a theorem prover technology for verifying properties related to XML schemas. We chose the PVS (Prototype Verification System [11]) theorem prover for our work because its type system includes predicate subtyping and bounded parametric polymorphism along with very general, and even higher order, logic capabilities. This makes PVS a suitable tool for expressing the complexity of XML Schema.

A PVS specification consists of a collection of theories. A theory is a specification of type signatures (of functions in particular) along with constraints applicable to instances of the theory expressed in the chosen logic. Hence in our approach, types and structures of the XML Schema have been represented as a collection of PVS theories.

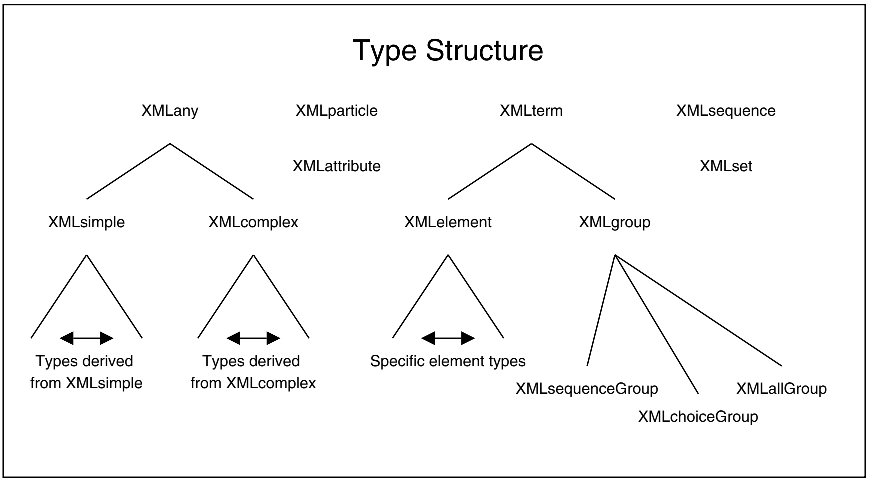
We claim several advantages for this approach. First, structural properties are expressed in a type system that conforms to well-established type systems of programming languages with subtyping and parametric polymorphism. Second, complex rules specified in the XML Schema documents in semi-formal English are now specified in PVS theories much more precisely and formally in a suitable logic. Likewise, specification of a variety of constraints in an application schema is now both required and possible in a more general formal framework. The most important advantage is that PVS allows automated reasoning about properties expressed in its theories, even application properties that are not expressible in XML Schema. Thus, reasoning and verification are supported in situations when XML data is processed by a transaction or a general purpose programming language.

D. Bell and J. Hong (Eds.): BNCOD 2006, LNCS 4042, pp. 262–265, 2006.

One particularly important application of a prover technology is verification that a transaction respects the integrity of a schema equipped with constraints. Our methodology for addressing this application requires explicit specification of frame constraints that the transaction does not affect, thus focusing the prover's attention on constraints that are at risk. The actual transaction update is specified in a declarative style as a binary predicate over pairs of database states, and the prover verifies that the update cannot violate the constraints. Although we use PVS to specify transaction updates, the methodology could be used with a variety of transaction languages.

2 Type Derivations in XML Schema

XML type anyType is the root of the XML type hierarchy [16, 17]. All other XML types are directly or indirectly derived from anyType by restriction or extension, and XML Schema has specific requirements on what derivations are valid. Two types that are derived directly from the type `XMLany` are `XMLsimple` and `XMLcomplex`. The subtyping relationships among XML types and our types specifying XML structures in PVS are represented in the following diagram:



A complex type is always derived from some other type, which may be either simple or complex. A complex XML type is equipped with a set of attributes and a content, which may be simple or complex. Complex content is specified via XML notions of elements, particles, groups, and group operators. Briefly, a complex content model determines a regular language of acceptable element instances, where the element tags are the symbols of the alphabet. A content derived by restriction will have a language that is a subset of the language of the base type, and a content derived by extension will have a language that is the concatenation of the base type's language with another language.

We define PVS theories for the XML constructs used in declaring XML complex types and predicates `extends` and `restricts` to formally capture the rules of XML Schema. A core idea behind type derivations in XML Schema is that an instance of a derived type may be viewed as a valid instance of its base type. This implies that all constraints associated with the base type are still valid when applied to an instance of a derived type. Our construction of PVS theories for XML Schema types and structures is governed by this basic requirement. This requirement corresponds to the notion of behavioral compatibility as presented in [4].

3 Related Research

The types as theories view is the basis of our previous results on generic data model management [1], semantics of objectified XML [3], and semantic compatibility problems for the object-oriented model [4].

A classical result on the application of theorem prover technology based on computational logic to the verification of transaction safety is [13]. Other results include usage of Isabelle/HOL [14] and PVS [2].

Results that address the problems of integration of a type system for XML with standard type systems [8, 9, 15] are confined to the problems of an integrated type system. These results do not address the issue of logic-based constraints, which is a distinctive feature of our work.

A variety of results are available on constraints for XML such as [7, 6, 10]. We consider XML constraints associated with a type system, and provide a prover technology to reason about constraints. This is probably the most distinctive feature of our work with respect to other related results.

4 Conclusions

Our experience with PVS had several lessons. First, intuitive techniques for verifying properties of transactions are inadequate. The PVS prover frequently exposed implicit assumptions we were making that were not logical consequences of the specifications. One advantage of using a prover tool is that it forces the developer to think more precisely and carefully about the application. Even when the goal theorem fails to prove, the prover gives valuable feedback to the developer. However, this feedback provided by PVS is not easy for a typical programmer to understand.

PVS does not check the consistency of a collection of axioms, and when possible such collections should not be used in writing specifications. We instead employ a definitional style which describes constraints as formulas, and then ask the PVS prover to show that the desired properties follow from the definitions.

A further conclusion is that tools such as PVS are not easy to use and require expertise and experience. A valid research goal is to develop proof strategies for particular tasks following the guidelines in [12, 5]. For a transaction verification proof strategy, a critical issue was separation of frame constraints from the logic-based specification of the actual updates. This strategy avoids expanding and rewriting the frame constraints and makes it possible to focus on the details of the proof of the active part of a transaction. In order to make these tools usable by typical programmers, a high-level user friendly interface based on suitable proof strategies is really required.

A major future research issue is extending this approach with reflective capabilities to allow the expression of XML features beyond conventional typing notions extended with constraints. In a separate piece of research we make use of a temporal logic specified by a suitable PVS theory in order to prove properties of object-oriented programs.

References

1. S. Alagić and P. A. Bernstein, A model theory for generic schema management, Proceedings of DBPL 2001, *Lecture Notes in Computer Science*, *2397*, pp. 228 - 246, 2002.
2. S. Alagić and J. Logan, Consistency of Java transactions, Proceedings of DBPL 2003, *Lecture Notes in Computer Science 2921*, pp. 71-89, Springer, 2004.
3. S. Alagić and D. Briggs, Semantics of Objectified XML, Proceedings of DBPL 2003, *Lecture Notes in Computer Science 2921*, pp. 147-165, Springer, 2004.
4. S. Alagić, S. Kouznetsova, Behavioral compatibility of self-typed theories, Proceedings of ECOOP 2002, *Lecture Notes in Computer Science 2374*, pp. 585-608, Springer, 2002.
5. M. Archer, B. Di Vito, and C. Munoz, Developing user strategies in PVS: A tutorial, Proceedings of STRATA 2003.
6. P. Buneman, S. Davidson, W. Fan, C. Hara and W-C. Tan, Reasoning about keys for XML, Proceedings of DBPL 2001, *Lecture Notes in Computer Science*, *2397*, pp.133 -148, 2002.
7. W. Fan and J. Simeon, Integrity constraints for XML, *Journal of Computer and System Sciences* 66, pp. 254-291, 2003.
8. H. Hosoya and B. Pierce, XDuce: A typed XML processing language, *ACM Transactions on Internet Technology*, 3(2), pp. 117-148, 2003.
9. H. Hosoya, A. Frisch, and G. Castagna, Parametric polymorphism for XML, Proceedings of POPL 2005, ACM, pp. 50-62.
10. G. M. Kuper and J. Simeon, Subsumption for XML types, Proceedings of ICDT, *Lecture Notes in Computer Science 1973*, pp. 331-345, Springer, 2001.
11. S. Owre, N. Shankar, J. M. Rushby, and D. W. J. Stringer-Clavert: PVS Language Reference, SRI International, Computer Science Laboratory, Menlo Park, California.
12. S. Owre and N. Shankar, Writing PVS proof strategies, Computer Science Laboratory, SRI International, http://www.csl.sri.com.
13. T. Sheard and D. Stemple, Automatic verification of database transaction safety, *ACM Transactions on Database Systems 14*, pp. 322-368, 1989.
14. D. Spelt and S. Even, A theorem prover-based analysis tool for object-oriented databases, *Lecture Notes in Computer Science 1579*, pp 375 - 389, Springer, 1999.
15. J. Simeon and P. Wadler, The Essence of XML, Proceedings of POPL 2003, ACM, pp. 1-13, 2003.
16. W3C: XML Schema Part 0: Primer, Second Edition, `http://www.w3.org/TR/xmlschema-0/`.
17. W3C: XML Schema Part 1: Structures, Second Edition, `http://www.w3.org/TR/xmlschema-1/`.

DTD-Driven Structure Preserving XML Compression

Stefan Böttcher and Rita Steinmetz

University of Paderborn (Germany), Computer Science
Fürstenallee 11, D-33102 Paderborn
{stb, rst}@uni-paderborn.de

Abstract. Whenever XML is used as the data format to exchange large amounts of data or even for data streams, the verbose behavior of XML is one of the bottlenecks. While compression of XML data seems to be a way out, it is essential for a variety of applications that the compression result still can be parsed, searched, transformed or modified efficiently. In order to support efficient search in compressed XML data, we have developed a compression technique that links two components: the first component uses the DTD to perform a structure-preserving compression of XML markup data, while the second uses a trie for the compression of text constants and attribute values.

1 Introduction

XML is a widespread standard for the exchange of data. Although XML has several advantages like the semi-structured nature of the data and the semantics that can be added by tag-names, XML bears the following major disadvantage. While the additional structure information added by the tags is an advantage when searching for sub-fragments that meet certain filter conditions, this markup makes the document rather verbose. Therefore, it is often argued that XML data is too verbose for the storage and exchange of huge data collections, and many practical applications today use relational databases or lists of comma-separated values instead of XML for data storage or data exchange. In comparison, we argue that a good compression algorithm should be able to compress not only most of the redundant markup, but also should compress the text or data values in such a way that the compressed data is more compact than a comma-separated list of values or a relational database table.

Another line of argumentation is that ordinary compression algorithms like zip can be used when the volume of the data to be stored or exchanged is critical. While this is true in principle, general compression algorithms will perform poorly, if an application requires access to only a very limited subset of the data. Whenever this subset of data is not known in advance, which is typical e.g. for ad-hoc querying applications to databases, it is difficult if not impossible to isolate a compressed fragment of the data that has to be decompressed in order to answer the query. In comparison, we prefer that queries can be applied to the compressed data without decompressing the whole data.

In Section 2 of this paper, we outline the requirements to a compression technique which is appropriate for very large XML documents and XML streams and which supports search on compressed data without decompression. In Section 3, we sketch a

D. Bell and J. Hong (Eds.): BNCOD 2006, LNCS 4042, pp. 266 – 269, 2006.

flexible architecture for implementing our approach, but due to space limitations, we defer some technical details to a forthcoming paper. In Section 4, we compare our approach with related work, and Section 5 outlines the summary and conclusions.

2 Requirements to the Compression of XML Documents

An algorithm for compression of XML documents should meet at least the following requirements:

The compression of an XML document should return a compressed XML (cXML) result which still allows answering queries without decompression of the complete cXML. As typical XML query languages like XQuery are based on XPath, at least the basic concepts of XPath like searching paths should be supported on the cXML data.

Additionally, transformations of XML documents, e.g., using XSLT or XQuery should be supported. Because these transformations use XPath as their access language to XML data, the support of XPath evaluation on compressed data is sufficient to meet this requirement too.

Furthermore, compression is especially useful when documents are very large, i.e., the compression method should support the compression of huge XML documents. When limited main memory is insufficient to store and compress large XML documents, the swapping of XML fragments to secondary storage may be considered a way out, but significantly slows down performance. Therefore, e.g. algorithms that are based on the Document Object Model (DOM) are expected to perform poorly. Instead, efficient compression of huge documents is required.

Finally, the compression method should be able to compress XML streaming data correctly. And XPath query execution on compressed streams should be supported as far as possible.

3 Design Decisions Meeting the Requirements to XML Compression

In order to meet the requirements, we have developed an XML compression system that is based on the following design decisions. We have designed a component-based XML compression system that combines a component for the compression of markup with a component for the compression of text constants and attribute values contained in the XML document. This allows us to find a suitable structure compression algorithm for markup independently of finding a suitable compression algorithm for text constants and attribute values, as long as both components are linked to each other. More precisely, there still has to be a pointer or identification mechanism which associates each text constant or attribute value with the markup that surrounds it. However, which technique is chosen for this association (pointers like in BPLEX [3], identifiers or simply an embedding of constants in place as e.g. suggested in [9]) is independent of the compression technique used for text constants and attribute values.

Inspired by ideas also found in [9], we use information found in the DTD to compress the structural information given in XML documents. However different from [9], we separate the storage and compression of structural information from the storage and compression of text constants and attribute values. We have linked both compression modules by the use of IDs, i.e., each ID uniquely represents a text constant or an attribute value used. For the compression of text constants and attribute values, we have implemented a trie [5] storing the constants in a prefix tree. Note however that our current implementation of the compression module for text constants can be replaced by any other approach for compressing text values, that assigns IDs to each value and that allows to compare the compressed values.

Finally, in order to enable the compression of very large documents and even XML streams, we have implemented a SAX-based parsing and compression method.

Due to space limitations, we delay the detailed description of our approach to a forthcoming publication.

4 Relation to Other Works

There exist several algorithms for XML compression. The first approach to XML compression was the XMill algorithm presented in [7]. It compresses the structural information, i.e., the tags, separately from the data, i.e., text constants and attribute values. The data is collected in several containers, where each container represents the data given for one kind of enclosing tags, and the whole container is compressed after data collection has been completed. This approach does not allow querying the compressed data and is not applicable to data streams.

The approaches XGrind [10], XPRESS [8] and XQueC [1] extend the XMill-approach in several aspects. Each of these approaches compresses the tag information using dictionaries and Huffman-encoding [6] and replaces the end tags by either a '/'-symbol or by parentheses. All three approaches allow querying the compressed data, and, although not explicitly mentioned, they all seem to be applicable to data streams. XQzip [4] and the approach presented in [2] compress the data structure of an XML document by combining bottom-up identical sub-trees. Afterwards, the data nodes are attached to the leaf nodes, i.e., one leaf node may point to several data nodes. The data is compressed by an arbitrary compression approach. These approaches allow querying compressed data, but they are not applicable to data streams.

An extension of [2] and [4] is the BPLEX algorithm presented in [3]. This approach does not only combine identical sub-trees, but also recognizes patterns that may span several levels of inner nodes within the XML tree, and therefore allows a higher degree of compression. As its predecessors, it allows querying the compressed data, but it is not applicable to data streams.

In contrast to all these approaches, our structure compression algorithm allows querying compressed data without fully decompressing it *and* is applicable to 'infinite' XML data streams, i.e., to continuous data streams and to data streams of previously unknown length. Our structure compression algorithm achieves an even higher level of compression than previous approaches supporting path queries, as the compressed data structure only contains compressed data plus some sparse structural information. As our compressed data structure contains nearly no structural information, only valid documents can be decompressed, i.e., our structure compression algorithm does not allow checking validity.

The approach presented in [9] follows the same basic ideas as our structure compression algorithm, i.e., to omit all information that is redundant because of the DTD. Whenever the document contains information not found in the DTD, this information is written to the compressed document. So the resulting compressed document generated by the approach of [9] is similar to our compressed structure except that e.g. text values and details of the compressed structure are encoded differently. However, the approach described in [9] to implement this compression idea is completely different from our structure compression algorithm. In [9], a DOM-tree is built and traversed simultaneously for the DTD and the XML document. In comparison, our approach is capable to sequentially compress 'infinite' data streams *and* to perform path queries on the compressed data stream, whereas the approach presented in [9] is limited by the size of main memory, i.e., only rather small documents can be compressed. For example, according to [9], they had problems with a file of 281 kB, as it exceeded their time threshold, whereas we were able to compress a document of more than 300 MB in a reasonable time.

5 Summary and Conclusions

The architecture for XML compression presented in this paper combines a technique that reduces the verbose structural parts of XML documents by removing information that is redundant when regarding the structure of a given DTD with a trie-based technique to compress text constants and attribute values. Both compression techniques are linked to each other by using unique IDs for text constants and attribute values that are stored in the trie. Although not described in this paper, our approach uses a SAX-based approach to compression which is capable to compress 'infinite' XML data streams, and which allows evaluating path queries on the compressed data without decompressing it. We are optimistic that our approach to compression and path query processing on compressed data is not only extensible to general XPath queries, but could also be applied to XQuery and XSLT.

References

[1] A. Arion, A. Bonifati, G. Costa, S. D'Aguanno, I. Manolescu, and A. Pugliese. XQueC: Pushing queries to compressed XML data. In Proc. VLDB, pages 1065–1068, 2003.

[2] Peter Buneman, Martin Grohe, Christoph Koch: Path Queries on Compressed XML. VLDB 2003: 141-152

[3] Giorgio Busatto, Markus Lohrey, and Sebastian Maneth, Efficient Memory Representation of XML Dokuments, DBPL 2005, LNCS 3774, S. 199–216, 2005.

[4] James Cheng, Wilfred Ng: XQzip: Querying Compressed XML Using Structural Indexing. EDBT 2004: 219-236

[5] E. Fredkin: Trie Memory. Communications of the ACM, 3(9):490-499, Sept. 1960

[6] D. Huffman, "A Method for Construction of Minimum-Redundancy Codes", Proc. of IRE, September 1952.

[7] H. Liefke and D. Suciu. XMill: An Efficient Compressor for XML Data, Proc. of ACM SIGMOD, May 2000.

[8] J. K. Min, M. J. Park, C. W. Chung. XPRESS: A Queriable Compression for XML Data. In Proceedings of SIGMOD, 2003.

[9] Neel Sundaresan, Reshad Moussa: Algorithms and programming models for efficient representation of XML for Internet applications. WWW 2001

[10] P.M. Tolani and J. R. Hartisa. XGRIND: A query-friendly XML compressor. In Proc. ICDE 2002, pages 225–234. IEEE Computer Society, 2002.

A Scalable Solution to XML View Materialization on the Web*

Dae Hyun Hwang[1], Hyunchul Kang[1], and Byeong-Soo Jeong[2]

[1] School of Computer Science and Engineering, Chung-Ang University, Korea
dhhwang@dblab.cse.cau.ac.kr, hckang@cau.ac.kr
[2] College of Electronics and Information, Kyung Hee University, Korea
jeong@khu.ac.kr

Abstract. Despite the practical importance of *XML materialized views* for XML database-backed Web applications and much attention to the issues for cache-answerability of XML queries with the static XML materialized views, the issues related to *incremental refresh* of XML materialized views against the relevant updates of the XML source have received little and limited attention. In this paper, we introduce our on-going project on XML view materialization on the Web. The main contribution of our solution is its *scalability* in the Web application environment. An overview of our design, the current status of implementation, and preliminary performance results are presented.

1 Introduction

In the past decades, much research was conducted on *view materialization* in relational database systems [3]. Would the same technology be needed and efficiently possible in the XML context? Since XML has now been established as a standard for data representation and exchange on the Web, materialization of XML views defined by XPath or XQuery expressions and their *incremental refresh* against the updates of XML source will be core techniques in efficiently supporting XML database-backed Web applications. Techniques for cache-answerability of XML queries which exploit middle-tier caching at the application server in the multi-tier architecture for XML database-backed Web applications (see Figure 1) have already received considerable attention. In those works, however, the XML materialized views are assumed to be static. As such, once they are materialized, no further maintenance is done. When they get obsolete due to the updates of their source, they might be recomputed from scratch.

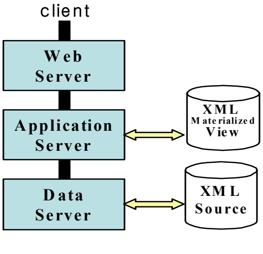


Fig. 1. Multi-tier Architecture for XML Database-backed Web Applications

To provide a full-fledged solution, the XML materialized views could be incrementally refreshed against the

* This work was supported by the Basic Research Program of the Korea Science & Engineering Foundation (grant No. R01-2003-000-10395-0).

D. Bell and J. Hong (Eds.): BNCOD 2006, LNCS 4042, pp. 270 – 273, 2006.

relevant updates of the source. So far, little work has been done for that. In [6], a technique for incrementally refreshing an XML materialized view defined by a subset of XQL was investigated. The key of this technique is maintenance of the auxiliary information called aggregate path index [1], which holds the collection of the source data objects relevant to the query pattern, in order to check the source update's relevance to the view. The main limitation of this technique, however, is that the size of the auxiliary information would be huge when the volume of XML source is large and/or when there are a number of materialized views. More general approach based on an XQuery algebra was proposed in [2]. However, it still suffers from possibly large amount of auxiliary information. A technique that utilizes the view correspondence assertions, which define the relationship between the schema of XML view and that of the source, in checking the source update's relevance to the view was proposed in [9]. This technique requires the existence of schema information. In [7], incremental maintenance of the path expression views defined by a subset of XPath 1.0 was investigated. The size of the auxiliary information that needs to be maintained with a view depends only on the expression size and on the answer size regardless of the source size. However, the path expression view dealt with is just a node set obtained as the result of evaluating an XPath expression rather than a fully materialized one.

In this paper, we introduce our on-going efforts for the development of XML materialized view technology, describing the main contribution compared with the previous work. In Section 2, the goal of our project along with the major design directions to realize it is described. Section 3 describes the current status of implementation and presents some preliminary performance results.

2 Design Overview

The main goal of our project is to provide a *scalable* solution to XML view materialization that is viable in the Web application environment. There are two aspects here. First, since XML views are defined against the XML source on the Web, the volume of view's source could be huge. Secondly, the number of materialized views could also be huge, maybe on the Internet scale. These led to the following major design decisions.

First, the storage structures of XML materialized views should be *disk-based* ones. If materialized views were maintained on main memory, the system would not scale at all. The issue of storage structures of XML materialized views was addressed by none of the previous work despite its practical importance. Only tree-based data structures that would fit in main memory were considered. The two major types of operations on a materialized view are retrieving the whole to provide the requested view and fixing just a small portion of it for incremental refresh. There exist performance tradeoffs between these two types of operations. As such, we need the storage structures of XML materialized views that compromise well between the two. We have designed three storage structures that respectively employ an *indexed log-structured text file*, an *indexed persistent DOM* (*PDOM* [5]) *file*, and an *RDBMS*. Due to space limitation, the details are omitted here.

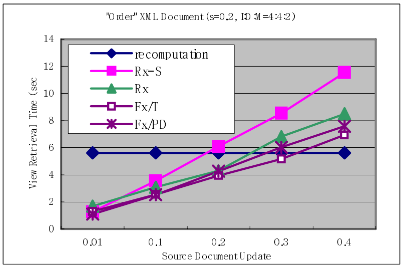


Fig. 2. View Retrieval Time

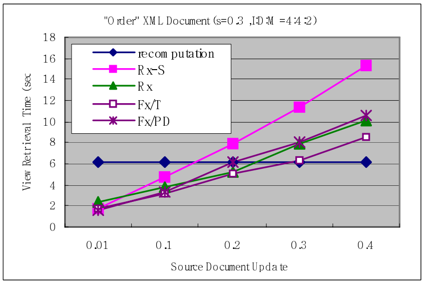


Fig. 3. View Retrieval Time

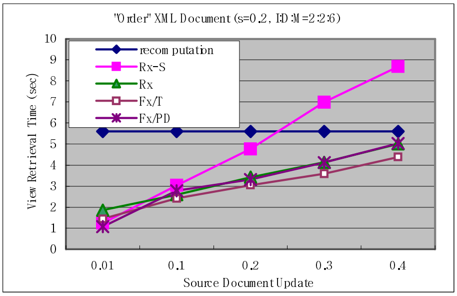


Fig. 4. View Retrieval Time

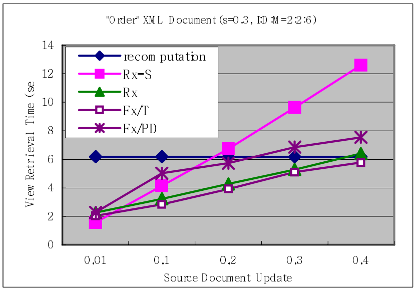


Fig. 5. View Retrieval Time

Secondly, *deferred* incremental refresh of XML materialized views, which requires *logging of updates* done to the XML source, should be the choice rather than its immediate counterpart to support a huge number of XML materialized views on the Web. With the deferred update propagation, the overhead incurred on XML update processing is just the update logging, which is negligible because the time it takes for update logging is dependent on the complexity of the update operation itself not on the number of materialized views supported. Another important advantage of deferred refresh policy is that it enables the *optimized* incremental refresh of a materialized view. By identifying the interrelationships among the logged updates, some can be cancelled with each other, merged into one, and/or just ignored. As for the space overhead incurred for update logging, it is regarded as small because the logged data is shared for all the materialized views, whose number may be enormous. Besides, it shall be fully amortized by the performance gain through view materialization.

Thirdly, the XML views considered should be the ones against *a huge set of XML documents* on the Web, rather than against a single document as assumed in the previous work. As such, our XML view is the result of *document filtering* as well as of *element retrieval* out of each filtered document.

3 Implementation and Performance Evaluation

We have implemented four schemes for maintaining XML materialized views in Java on Windows 2000 Server. They are called *Rx-S*, *Rx*, *Fx/T*, and *Fx/PD*. The first two employ an RDBMS (Oracle 9*i*) with JDBC connection. Fx/T employs an indexed log-structured text file whereas Fx/PD employs an indexed PDOM file. (For this, PDOM implementation of GMD-IPSI XQL Engine Version 1.0.2 [4] was used.) All except Rx-S perform optimized batch processing of update log. We have also implemented *recomputation* of the view. The performance of XML view retrieval by all these schemes was compared through a set of experiments.

The *order* documents in XML of TPC-W benchmark [8] were used as XML source, which was stored in an RDBMS (Oracle 9*i*). The experiments were conducted on a system of Celeron 2GHz PC with 512 MB main memory. Figure 2 through Figure 5 compare the view retrieval times as the ratio of source document updates increases. The parameter *s* denotes the selectivity of the view, and *I:D:M* the ratio among source insert: delete: modify. Our schemes considerably outperformed the recomputation as long as source update ratio was not too high. The efficiency of optimized batch processing of update log was also confirmed.

References

1. Chen, L., Rundensteiner, E.: Aggregate Path Index for Incremental Web View Maintenance. Proc. Int'l Workshop on Advanced Issues of EC and Web-based Info. Systems. (2000)
2. Dimitrova, K. et al.: Order-sensitive View Maintenance of Materialized XQuery Views. Tech. Rep. WPI-CS-TR-03-17. Comp. Sci. Dept. Worcester Polytechnic Institute. (2003)
3. Gupta, A., Mumick, I: Materialized Views: Techniques, Implementations, and Applications. MIT Press (1999)
4. Huck, G, Macherius, I.: GMD-IPSI XQL Engine. http://xml.darmstadt.gmd.de/xql/
5. Huck, G. et al.: PDOM: Lightweight Persistency Support for the Document Object Scheme. Proc. OOPSLA Workshop on Java and Databases: Persistence Options. (1999)
6. Quan, L. et al.: Argos: Efficient Refresh in an XQL-Based Web Caching System. Proc. Workshop on the Web and Databases. (2000) 23-28
7. Sawires, A. et al.: Incremental Maintenance of Path-Expression Views. Proc. ACM SIGMOD Int'l Conf. on Management of Data. (2005)
8. TPC-W: Transaction Processing Performance Council. http://www.tpc.org/tpcw/
9. Vidal, V., Casanova, M.: Efficient Maintenance of XML Views Using View Correspondence Assertions. Proc. Int'l Conf. on EC and Web Technologies. (2003) 281-291

A Rule-Based Data Warehouse Model

Cécile Favre, Fadila Bentayeb, and Omar Boussaid

ERIC Laboratory, University of Lyon 2,
5 av. Pierre Mendès-France, 69676 Bron Cedex, France
{cfavre, bentayeb}@eric.univ-lyon2.fr
omar.boussaid@univ-lyon2.fr

Abstract. A data warehouse is built by collecting data from external sources. Changes that occur have to be reflected in the data warehouse thanks to schema updating or versioning. However a data warehouse has also to evolve according to users' analysis needs. This evolution is rather driven by knowledge than by data. To take into account these changes, we propose a new Rule-based Data Warehouse (*R-DW*) model in which rules integrate users' knowledge to dynamically create dimension hierarchies. The *R-DW* model is composed of a fixed part which is a fact table related to its first level dimensions, and an evolving part which contains the rules. Our model allows analysis context evolution and increases interactions between users and the decision support system.

1 Introduction

A data warehouse constitute an effective support in managing an increasing mass of data from heterogeneous sources and in providing an answer to analysis needs. Exhaustively establishing the users' needs is a complex task. Sometimes, users have knowledge which is not represented in the data warehouse and which may be needed to have a complete analysis. We therefore have to make the data warehouse evolve to be able to take into account users' knowledge. But this kind of evolution is not easy to achieve in the traditional rigid data warehouse models.

Indeed, a data warehouse schema evolution is performed following two different ways, namely schema updating and schema versioning. The first approach consists in updating a schema and transforming data from an old schema into a new one [1]. The second approach, on the contrary, keeps track of all versions of a schema [2]. These two approaches constitute a solution to the dimensions evolution, when the latter is induced by the evolution of data. However, once the data warehouse made up, the users may only carry out analysis provided by the model. These approaches do not take into account new analysis needs driven by the expression of knowledge ; thus they are not flexible enough.

Works aiming at an increased flexibility in data warehouses generally use rules to either define the data warehouse schema from source schemas [3, 4] or support various types of constraints [5] in order to keep the data and the analysis coherent, or manage the exceptions during the aggregation process [6]. The rule-based languages contributed in making the data warehouse more flexible. We want to bring such a flexibility for the evolution of analysis needs driven by knowledge.

D. Bell and J. Hong (Eds.): BNCOD 2006, LNCS 4042, pp. 274–277, 2006.

Thus, we propose a new Rule-based Data Warehouse (*R-DW*) model, in which rules create new granularity levels in dimension hierarchies by integrating the users' knowledge. Our *R-DW* model has several advantages as compared to existing data warehouse models. It allows (1) to dynamically create hierarchies ; (2) to make analysis on evolving contexts ; (3) to increase the interaction between users and the information system since they can integrate their own knowledge.

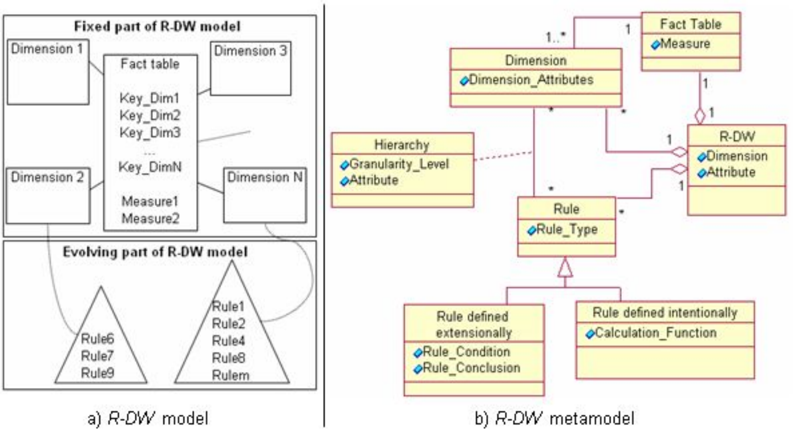
Section 2 explains principles of our *R-DW* model and an example is described in Section 3. Its implementation and a case study are presented in Section 4. Section 5 concludes this paper and provides future research directions.

2 The *R-DW* Model

The *R-DW* model is composed of a fixed part and an evolving part (Figure 1a). The fixed part is a star schema, it is composed of a fact table and first level dimensions. The evolving part is composed of rules which generate new granularity levels in dimension hierarchies based on the user's knowledge.

The metamodel described in Figure 1b is a generalization of the *R-DW* model. It contains a definition of the fixed part of the model represented by the *Fact table* and *Dimension* classes. The evolving part is represented by *Rule defined extensionally* and *Rule defined intentionally* classes. Rules are defined extensionally when they are based on well-known values that can be enumerated ; on contrary, they can be defined intentionally using functions.

Rules defined extensionally are "if-then" rules. The then-clause contains the definition of a higher granularity level. The if-clause contains conditions on the lower granularity levels. Rules defined intentionally allow inferences on the granularity level according to lower levels. For instance, to obtain the department of a customer, we just have to extract the first two characters of the postal code.



a) *R-DW* model

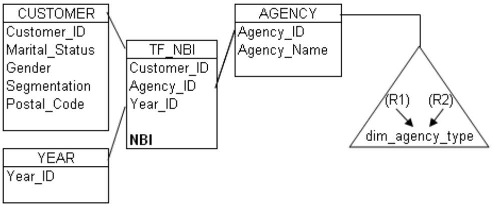
b) *R-DW* metamodel

Fig. 1. *R-DW* model and metamodel

Our model provides to the end-users a mean to define their own rules to determine new dimensions hierarchies. Then users can therefore analyze data according to the new granularity levels. The data warehouse model becomes thus more flexible for analysis.

3 Example

As an example, we use the case study of Le Crédit Lyonnais (LCL). The annual Net Banking Income (NBI) is the profit obtained from the management of customers account. This is a measure studied according to customers, agencies and years. The students portfolio manager of LCL knows that some agencies have students as only customers. Our *R-DW* model (Figure 2) can integrate this knowledge to carry out an analysis taking into account the student dedicated agencies. The fixed part of the model is made up of the fact table *TF_NBI* and the dimension tables *CUSTOMER*, *YEAR* and *AGENCY*. To the fixed part we add the evolving part containing rules that describe the manager's knowledge on student agencies (R1 and R2).The induced model makes it possible to carry out new analysis based on the user's knowledge. A new granularity level has been added in the hierarchy. Therefore our *R-DW* model allows us to build aggregates, by considering that the facts to aggregate concern a student agency (R1), or a classical agency (R2).



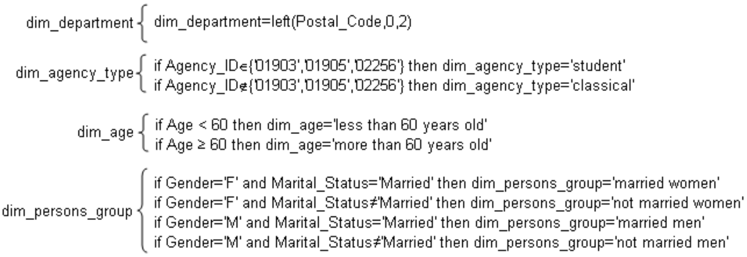
(R1): if Agency_ID∈{'01903','01905','02256'} then dim_agency_type='student'
(R2): if Agency_ID∉{'01903','01905','02256'} then dim_agency_type='classical'

Fig. 2. Rule-based Data Warehouse model for the NBI analysis

4 Implementation and Case Study

We developed a Web platform (HTML/PHP) behind Oracle DBMS used to store the fact table and dimension tables. Two additional tables contain the rules defined extensionally and intentionally. The Web platform allows the user to define the rules that generate the analysis axes. We initially restricted ourselves to decisionnal queries with an aggregation according to one granularity level. This aggregation is computed by a PL/SQL stored procedure.

The *R-DW* model for the NBI analysis previously presented is enriched by the granularity levels which are created by the evolving part of the Figure 3. From this model, the user will be able to run NBI analyses, not only by considering first level dimensions, but also by considering new dimension levels like agency type, department, age classes.



```
dim_department     { dim_department=left(Postal_Code,0,2)

dim_agency_type    { if Agency_ID∈{'01903','01905','02256'} then dim_agency_type='student'
                   { if Agency_ID∉{'01903','01905','02256'} then dim_agency_type='classical'

dim_age            { if Age < 60 then dim_age='less than 60 years old'
                   { if Age ≥ 60 then dim_age='more than 60 years old'

dim_persons_group  { if Gender='F' and Marital_Status='Married' then dim_persons_group='married women'
                   { if Gender='F' and Marital_Status≠'Married' then dim_persons_group='not married women'
                   { if Gender='M' and Marital_Status='Married' then dim_persons_group='married men'
                   { if Gender='M' and Marital_Status≠'Married' then dim_persons_group='not married men'
```

Fig. 3. Evolving part of the *R-DW* data warehouse for the NBI analysis

5 Conclusion

We proposed a rule-based data warehouse model. Rules introduce users' knowledge into the data warehouse for new analysis purposes. Our *R-DW* model is composed of two parts: a fixed part that contains a fact table and first level dimensions ; and an evolving part that is defined by rules defining granularity levels of dimension hierarchies. The *R-DW* model presents the advantage of being able to dynamically create dimension hierarchies, according to users' knowledge therefore satisfying their analysis needs. The implementation we developed was applied on the LCL case study and gave some promising results.

The perspectives opened by this study are numerous. First, we have to extend the analysis possibilities of our implementation. Then we intend to measure the performance of our approach in terms of storage space and response time. Furthermore we plan to define constraints on rules and a language that allows us to validate these rules. Moreover, we think it would be interesting to use non supervised learning methods for discovering new rules.

References

1. Blaschka, M., Sapia, C., Höfling, G.: On Schema Evolution in Multidimensional Databases. In: DaWaK'99: 1st International Conference on Data Warehousing and Knowledge Discovery. (1999) 153–164
2. Eder, J., Koncilia, C.: Changes of dimension data in temporal data warehouses. In: DaWaK'01: 3rd International Conference on Data Warehousing and Knowledge Discovery. (2001) 284–293
3. Kim, H.J., Lee, T.H., Lee, S.G., Chun, J.: Automated Data Warehousing for Rule-Based CRM Systems. In: 14th Australasian Database Conference on Database Technologies. (2003) 67–73
4. Peralta, V., Illarze, A., Ruggia, R.: On the Applicability of Rules to Automate Data Warehouse Logical Design. In: CAiSE Workshops. (2003)
5. Carpani, F., Ruggia, R.: An Integrity Constraints Language for a Conceptual Multidimensional Data Model. In: SEKE'01: XIII International Conference on Software Engineering & Knowledge Engineering. (2001)
6. Espil, M.M., Vaisman, A.A.: Efficient Intensional Redefinition of Aggregation Hierarchies in Multidimensional Databases. In: DOLAP'01: 4th ACM International Workshop on Data Warehousing and OLAP. (2001)

Enriching Data Warehouse Dimension Hierarchies by Using Semantic Relations*

Jose-Norberto Mazón and Juan Trujillo

Dept. of Software and Computing Systems
University of Alicante, Spain
{jnmazon, jtrujillo}@dlsi.ua.es

Abstract. Data warehouse dimension hierarchies are of paramount importance in OLAP (On-Line Analytical Processing) tools to support the decision-making process, since they allow the analysis of data at different levels of detail (i.e. levels of aggregation). This is why it is crucial to capture adequate hierarchies in the requirement analysis stage. However, operational sources may not be able to supply enough data to construct every level of these hierarchies. In this paper, we propose the application of semantic relations among WordNet concepts to enrich hierarchies by adding the required levels of aggregation. Decision makers will thus be able to achieve their information needs for analysis.

1 Introduction

In the early nineties, Inmon [4] coined the term data warehouse (DW) as "a subject oriented, integrated, non-volatile, and time-variant data collection in support of management's decision". It is widely accepted that data warehouses (DWs) are based on multidimensional (MD) modelling which structures information into facts and dimensions. A fact contains useful measures of a business process (sales, deliveries, etc.), whereas a dimension represents the context for analysing a fact (product, customer, time, etc.) by means of hierarchically organized dimension attributes [1, 9]. These hierarchies are of a crucial importance when OLAP (On-Line Analytical Processing) tools support the decision-making process, since they are used in such operations as roll-up or drill-down to analyse the large amount of data stored in the DW.

Lately, we have been developing an approach [5, 6], based on UML (Unified Modelling Language) [8] and the i* notation [10] for the conceptual modelling of DWs from user requirements. With this approach, once user requirements are correctly captured, we obtain the corresponding MD schema (i.e. required MD schema). The required MD schema is then conformed to the operational sources that will populate the DW. Nevertheless, in this conformation process we found

* Work supported by the projects: TIN2004-00779 from the Spanish Government, GV05/220 from the Valencia Government, and PBC-05-012-2 from the Castilla-La Mancha Government. Jose-Norberto Mazón is funded by the Spanish Ministry of Science and Education under a FPU grant (AP2005-1360).

D. Bell and J. Hong (Eds.): BNCOD 2006, LNCS 4042, pp. 278–281, 2006.

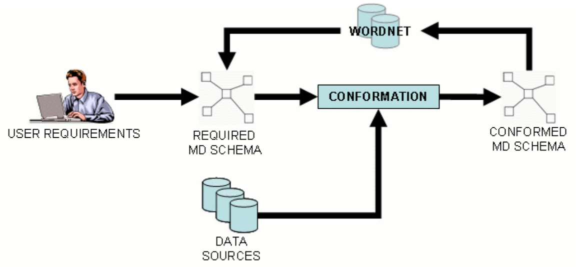


Fig. 1. Using WordNet to enrich the conformed MD schema

that the required MD schema could not be totally specified as many terms and data were missing from the operational sources and only a reduced version of this schema was obtained: the conformed MD schema.

Consequently, data sources may not be enough to obtain required hierarchies and DW users can only analyse data by using conformed hierarchies[1]. The final DW will not therefore completely satisfy final user requirements. This is why this paper presents a preliminary, novel approach to enrich conformed dimension hierarchies by adding new levels of aggregation in order to obtain the required hierarchies, although not enough data is stored in the operational sources. DW users will thus satisfy their analysis needs. To accomplish this, we propose the use of semantic relations among concepts provided by WordNet [7]. The initial hypothesis is that both DWs and WordNet present hierarchical structures: dimension hierarchies in DWs show the relationships between value domains from different dimension attributes (set by levels of aggregation), while WordNet presents hierarchical semantic relations between concepts, such as hypernymy/hyponymy or meronymy/holonymy. Therefore, our approach is based on using these WordNet semantic relations to add new levels to conformed dimension hierarchies in order to obtain the required hierarchies. Figure 1 summarizes this scenario.

2 Using UML for Data Warehouse Modelling

In this paper we use our UML profile for the conceptual design of DWs according to MD modelling [5], which divides data into facts and dimensions. In order to provide data on a suitable level of granularity, hierarchies are defined on the dimensions. The profile is defined by a set of stereotypes and tagged values to represent MD properties elegantly on a conceptual level by using a UML class diagram (see Table 1). We refer reader to [5, 9] for further explanations.

[1] We regard required hierarchies to be those obtained from end user requirements (i.e. they are part of the required MD schema); while conformed hierarchies are those that conform to the data provided by operational sources (i.e. they are part of the conformed MD schema).

Table 1. Class stereotypes of our UML profile related to dimension hierarchies

Stereotype	Description	Icon
Dimension	Represents dimensions consisting of hierarchy levels.	↗
Base	Represents dimension hierarchy levels and attributes.	B

3 Using WordNet to Enrich Dimension Hierarchies

Our approach is based on using the semantic relations provided by Wordnet to enrich conformed hierarchies (i.e. adding the new aggregation levels) in order to obtain the required ones. The main tasks are shown in Fig. 2. A dimension attribute (denoted as D) is chosen from a conformed hierarchy. All instance values of this attribute (denoted as W) are then obtained from operational data sources. These values must be disambiguated by using a word sense disambiguation (WSD) algorithm to obtain their right senses (denoted as S). Afterwards, iterations are used to obtain hypernyms or meronyms of senses (denoted as H). A new hierarchy level is created for each of these iterations with its corresponding instance values. Iterations are repeated until every hierarchy level needed by the required hierarchy is obtained. In this way we take advantage of semantic relations of WordNet (hypernyms or meronyms) to enrich dimension hierarchies.

Figure 3 shows an example of our approach: a user requirement states that a quantity of sold product must be analysed according to several aggregation levels (subtype, type, and class of product). So a hierarchy is constructed according to this requirement (i.e. required hierarchy). However, only the name of the product is available from the data sources. Thus, when the required hierarchy is conformed to these sources, the resulting conformed hierarchy does not have

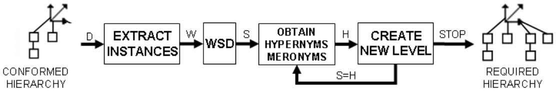
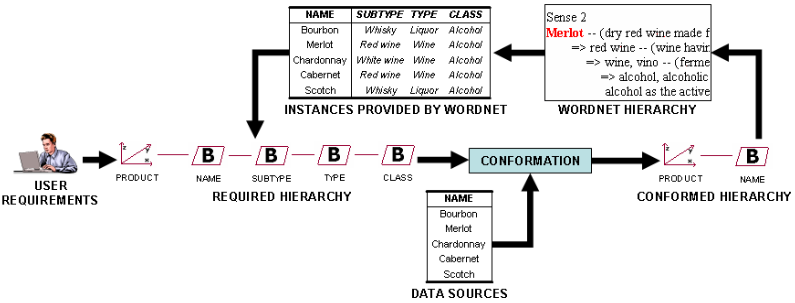


Fig. 2. Main tasks of our approach



NAME	SUBTYPE	TYPE	CLASS
Bourbon	Whisky	Liquor	Alcohol
Merlot	Red wine	Wine	Alcohol
Chardonnay	White wine	Wine	Alcohol
Cabernet	Red wine	Wine	Alcohol
Scotch	Whisky	Liquor	Alcohol

Sense 2
Merlot -- (dry red wine made f
=> red wine -- (wine havir
=> wine, vino -- (ferme
=> alcohol, alcoholic
alcohol as the active

NAME
Bourbon
Merlot
Chardonnay
Cabernet
Scotch

Fig. 3. An example of applying our approach

enough aggregation levels to satisfy user needs. Our approach was then applied to enrich this conformed hierarchy and obtain the required hierarchy. The instances of the new levels (provided by WordNet) can also be seen in Fig. 3.

4 Conclusion and Future Work

Obtaining the required dimension hierarchies captured from DW users in the requirement analysis stage is crucial for improving the decision-making process. However, when required hierarchies are conformed to operational sources, we found that these sources may not provide enough data to construct every level of required hierarchies, meaning that only conformed hierarchies can be obtained. Therefore, user requirements are not satisfied, as conformed hierarchies may not deliver the information expected to support the decision-making process. In this paper, we propose the application of semantic relations from WordNet to obtain the required hierarchies. The advantage of our proposal is clear: the enrichment of conformed hierarchies by adding new aggregation levels in order to satisfy the required hierarchies. These required hierarchies allow DW users to satisfy their information analysis needs, since they better support the decision-making process. Finally, we plan to use WordNet within DWs systems to overcome inaccuracy problems regarding summarizability [2, 3].

References

1. Akoka J., Comyn-Wattiau I., Prat N.: Dimension Hierarchies Design from UML Generalizations and Aggregations. ER 2001. LNCS 2224, pp. 442-455, Springer.
2. Horner J., Song I-Y., Chen P.: An Analysis of Additivity in OLAP Systems. 7th ACM Int. Workshop on Data Warehousing and OLAP (DOLAP), pp. 83-91, 2004.
3. Horner J., Song I-Y.: A Taxonomy of Inaccurate Summaries and Their Management in OLAP Systems. ER 2005. LNCS 3716, pp. 433-448.
4. Inmon W.: Building the Data Warehouse, John Wiley & Sons, 1996.
5. Luján-Mora S., Trujillo J., and Song I-Y.: A UML Profile for Multidimensional Modelling in Data Warehouses, Data & Knowledge Engineering. In Press.
6. Mazón J-N, Trujillo J., Serrano M., Piattini M.: Designing Data Warehouses: from Business Requirement Analysis to Multidimensional Modelling. Int. Workshop on Requirements Engineering for Business Needs and IT Alignment, REBNITA 2005.
7. Miller G.A., Beckwith R., Fellbaum C., Gross D., Miller K.J.: WordNet: An On-Line Lexical Database, International Journal of Lexicography, 3(4), 1990.
8. Object Management Group (OMG). Unified Modelling Language Specification 1.5. http://www.omg.org/cgi-bin/doc?formal/03-03-01. 2004.
9. Trujillo J., Palomar M., Gómez J., Song I.Y.: Designing Data Warehouses with OO Conceptual Models. IEEE Computer, 34(12):66-75, 2001.
10. Yu, E.: Modelling Strategic Relationships for Process Reenginering, Ph.D. thesis. University of Toronto, 1995.

A Composite Approach for Ontology Mapping*

Ying Wang, Jianbin Gong, Zhe Wang, and Chunguang Zhou

College of Computer Science, Key Laboratory of Symbol Computation
and Knowledge Engineering of the Ministry of Education,
Jilin University, Changchun 130012, China
yichunwy@163.com, cgzhou@jlu.edu.cn

Abstract. Ontology mapping is one of the important problems for the development of Semantic Web. Establishing such mappings has been the focus of a variety of research originating from diverse communities. In this paper, we propose a Composite Approach for Ontology Mapping (ACAOM) for semi-automatic ontology mapping based on the combination of the name and instance methods. We conclude that the combination is a promising method.

1 Introduction

Ontologies are the cores of the Semantic Web because they are the carriers of the meaning contained in the Semantic Web. However in many cases, different domains define different ontologies containing the same concepts. A flexible approach is needed to establish semantic correspondences between ontologies. We propose a mapping approach based on the combination of the name and instance methods.

The rest of the paper is organized as follows. Section 2 describes the main ideas in our approach and the mapping strategies used. Section 3 discusses the experimental results and concludes the paper with discussions on future research.

2 A Composite Approach for Ontology Mapping (ACAOM)

2.1 Name-Based Strategy

In this paper, we use a semantic dictionary WordNet [1] and add a path method to it. We use WordNet as auxiliary information to calculate similarity values between concepts in two ontologies and integrate the measure of path length into our mapping approach. We modify Lin's method [2] to obtain the following formula:

$$sim_{new}(s_1, s_2) = \frac{2\square\log\left(p\left(s_1, s_2\right)\right)}{\log\left(p\left(s_1\right)\right) + \log\left(p\left(s_2\right)\right)} \bullet \frac{1}{2}\alpha^l \tag{1}$$

When we search for the common hypernym of word senses (senses for short thereafter) s_1 and s_2, we design a punishment coefficient $\frac{1}{2}\alpha^l$, where α is a constant between

[*] This work was supported by the Natural Science Foundation of China (Grant No. 60175024) and the Key Science-Technology Project of the National Education Ministry of China (Grant No. 02090). To whom correspondences should be addressed, Email: cgzhou@jlu.edu.cn.

D. Bell and J. Hong (Eds.): BNCOD 2006, LNCS 4042, pp. 282 – 285, 2006.

0 and 1 and is used to adjust a decrease on the similarity between two senses when the path length between them is deepened. When we search for the common hypernym of senses s_1 and s_2, l expresses the longest distance either sense s_1 or sense s_2 passes by in a hierarchical hypernym structure. Because senses s_1 and s_2 occupy one of the common branches, this value has to be halved.

Words w_1 and w_2 may contain many senses, we use $s(w_1)$ and $s(w_2)$ to denote the set of senses of word w_1 and word w_2 respectively, that is, $s(w_1)=\{s_{1i} \mid i=1,2,......,m\}$ and $s(w_2)=\{s_{1j} \mid j=1,2,......,n\}$. Assume that the numbers of senses that words w_1 and w_2 contain are m and n, we define the similarity between them as follows:

$$sim(w_1, w_2) = \max\left(sim(s_{1i}, s_{2j})\right) \tag{2}$$

2.2 Instance-Based Strategy

This strategy exploits the vector space model to denote documents and then finds mapping results between entities. In this paper, we assume that documents have been associated with concept nodes in ontologies. We establish feature vectors for each document that belongs to the concept nodes and then compute the feature vectors for each concept node.

In the pre-processing stage, we process documents in order to perform the computation described below. This process includes removing html or other tags, removing stop words according to a stop list, such as, a, the etc, and performing word normalization using the porter stemming algorithm [3]. Then we use vectors to denote the weight values of the documents.

In a vector space model, we attach a weight to each word to measure how important the word is in the document and we deploy the method developed in the Smart system [4]. The formulas used in the method are given below:

$$w_i = new\_tf_i \square idf_i = (0.5 + 0.5\frac{tf_i}{\max\_tf}) \square \lg\frac{N}{n_t} \tag{3}$$

where new_tf_i expresses the computation of word frequency, tf_i (term frequency) is the number of times that word i appears in document d, idf_i expresses inverse document frequency and N is the total number of documents in document set D, n_t is the number of documents containing word i and w_i is the weight of word i.

We differentiate between leaf-nodes and non-leaf nodes in an ontology and process them differently. For each leaf-node, its feature vector is computed as the average number of documents assigned to it. Let C^K be the feature vector of concept node K and D_j is the collection of documents that have been assigned to it. w_{ij} is the weight of word i in document j. We have:

$$C_i^k = \frac{\sum_{D_j \in K} w_{ij}}{\left|D_j\right|} \tag{4}$$

We put an emphasis on all the sub nodes of non-leaf nodes. The vector of feature *i* is thus constructed as follows:

$$C_i^k = \sum C_i^{sub} \tag{5}$$

where C_i^{sub} is the vector of feature *i* for a leaf-node that is under node *K* and the vector of feature *i* of a non-leaf node is defined as the sum of feature vectors associated with its child-nodes.

A new approach [5] is proposed to measure the degree of similarity between two vectors:

$$SIM = \frac{\left|C^a - C^b\right|^2}{\left|C^a\right|^2 + \left|C^b\right|^2} = \frac{\sum_{i=1}^{n}(C_i^a - C_i^b)^2}{\sum_{i=1}^{n}\left(C_i^a\right)^2 + \sum_{i=1}^{n}\left(C_i^b\right)^2} \tag{6}$$

where *SIM* is the degree of similarity between concept nodes *a* and *b*, C^a and C^b are the feature vectors of *a* and *b* respectively and *n* is the given count of feature vectors. The *SIM* approach takes into account both the angle and the length of vectors. When two vectors are equal, the value of *SIM* is 0. If two vectors are orthogonal, the value of *SIM* is 1. However, the results are opposite to the common sense. So we modify the formula as follows:

$$SIM_{new} = 1 - SIM \tag{7}$$

We integrate the results that are computed by the two mapping strategies and use the following common combination method:

$$sim(e_{i_1 j_1}, e_{i_2 j_2}) = \sum\nolimits_{k=1}^{2} w_k sim_k(e_{i_1 j_1}, e_{i_2 j_2}) \tag{8}$$

where w_k is the weight for an individual strategy and assigned by hand. For this method a fixed constant *a* is taken as a threshold value. If $sim(e_{i_1 j_1}, e_{i_2 j_2}) > a$, then the mapping will be correct.

3 Experimental Results and Conclusions

We evaluated ACAOM using two data sets that describe courses at Cornell University and Washington University [6]. We run both our system and the iMapper [7] system and used information retrieval metrics, Precision and Recall, to evaluate our method. The ontologies of Course Catalog I have between 34 to 39 concepts and precisions range from 82.4% to 85.3% and recalls range from 75.7% to 85.3%. The ontologies of Course Catalog II have between 166 to 176 concepts and precisions range from 66.1% to 72.9% and recalls range from 57.4% to 70%. As these results show, ACAOM can achieve better results than the iMapper System.

Although ACAOM produces better results of ontology mapping, there are several reasons that prevent ACAOM from correctly matching the remaining nodes. First, in the name-based strategy, ACAOM does not consider the structures between words and assumes that all the words are equally important. Second, in the instance-based strategy, we use word frequencies only to carry out the computation and do not analyze the importance of words.

In this paper, we proposed an ontology mapping approach which combines two strategies. These two strategies make use of the information about both names and instances assigned to concept nodes respectively to calculate similarities between entities. An integrated approach has been designed to incorporate both strategies. The experimental results show that ACAOM performs better than iMapper and it improves the precision of iMapper from +2.4% to 5.9%.

There are several aspects that can be improved in our proposed system. (1) We could realize ontology merging and integration in the same system. ACAOM can be applied to other aspects of ontology related issues, such as, queries based on distributed ontology. (2) Our method cannot support n:m mappings at present, which are useful in many cases. We will extend our method to address issues in the future.

References

1. Miller,G.A.: WordNet: A Lexical Database for English. Communications of the ACM,Vol.38 (1995) 39–41
2. Lin, Dekang.: An Information-Theoretic Definition of Similarity. In Proceedings of the 15th International Conference on Machine Learning, Madison, WI, (1998) 296-304
3. http://www.tartarus.org/~martin/PorterStemmer/
4. Buckley,C., Lewit, A.F.:Optimization of Inverted Vector Searches. SIGIR (1985) 97-110
5. Wang Jianyong, Xie Zhengmao, Lei Ming, and Li Xiaoming: Research and Evaluation of Near-replicas of Web Pages Detection Algorithms. Chinese Journal of Electronics, Vol s1.(2000)
6. http://anhai.cs.uiuc.edu/archive/summary.type.html
7. Su,Xiaomeng., Gulla, J.A.: Semantic Enrichment for Ontology Mapping. In Proceedings of the 9th International Conference on Natural Language to Information Systems (NLDB04), Salford, UK, June 23-25, (2004) 217-228

Towards the Completion of Expressing and Checking Inheritance Constraints in UML

Djamel Berrabah

CRIP5, Paris 5 University
45 rue des Saints Pères, 75270 Paris cedex 06, France
berrabah@math-info.univ-paris5.fr

Abstract. The automation of a conceptual schema translation to design a database using CASE tools is one of the multiple efforts devoted to face the problems of database modeling. These tools often do not take into account all the information (structures and constraints) of the real word. Our goal is to enrich these tools and to improve them in order to take into account a big number of the defined constraints. We aim to combine the advantages of the object and relational approaches. So, constraints in object databases are expressed in OCL while they are expressed in relational databases by using active mechanisms.

1 Introduction

In database (DB) design methodologies [8, 17], rules are defined to translate a "*conceptual schema*" (CS), such as UML class diagram, into a relational or object "*target schema*" (TS). TS-elements obtained do not coincide completely with CS-elements, thus bringing about some semantic losses [3]. This problem often arises when constraints are not correctly translated. “Participation constraints” (PCs) defined on generalization/specialization relationships may be one of those constraints. These constraints concern the linking rules of general object to special objects. Today’s most current commercial CASE tools like Power AMC [16] and Rational Rose [15] do not take these constraints into account and only generate an incomplete database schema.

This paper represents a part of our research dealing with database modelling process (from conceptual level to database implementation) [2, 3]. The aim is to provide an efficient mechanism which deals automatically with PCs (to express and translate them). These mechanisms consist in creating additional OCL-constraints (Object Constraint Language) [12, 14] or trigger-based SQL constraints. Thus, an automatic module to express PCs defined in a CS has been thought to be a good idea to implement and check them during DB manipulations. Thus, the translation is improved and the semantic loss is reduced.

The structure of this paper is as follows. Section 2 presents basic concept of constraints. Section 3 points out how to deal with PCs that are defined on generalization/specialization relationships using a constraint specification language. Our approach is based on trigger-based SQL scripts and OCL. Finally, the paper ends with a conclusion.

D. Bell and J. Hong (Eds.): BNCOD 2006, LNCS 4042, pp. 286 – 290, 2006.

2 Constraints

A constraint constitutes a semantic restriction linked to one or several elements of the CS (property, class, relationship). It represents semantic information associated with these elements. A CS must include the correct constraint definitions, without conflicts [3], of all suitable constraints. The graphic elements offered by the CASE tool do not allow expressing the totality of the constraints. In addition, no mechanism is generated to ensure the satisfaction of all the expressed constraints which must be translated in the TS. These constraints are known as integrity constraints. They must be satisfied in each DB state. Sometimes, they are checked by declarative constraints which are not always sufficient. Thus other means, such as OCL and triggers, are needed to express and check them. OCL is a formal specification language to express additional constraints in UML. The kind of constraints which can be expressed using OCL includes invariants on the structures of the CS. In SQL 2003 [7], a trigger is expressed by ECA rules [5, 6]. It constitutes a good means to implement referential actions. It is activated during DB transition state.

3 Transformation Rules of a CS

In this section, we study participation constraints (PCs) defined on generalization/ specialization relationships. In other words, we provide rules to translate these constraints using a constraint specification language. PCs refer to the conditions of linking general class objects to two or several special class objects. They can have the same definition as those defined on binary relationships but they have not the same semantic. The aim of this study is to combine the advantages of both OCL and SQL in checking these constraints. All SQL statements are represented in ECA-rules form.

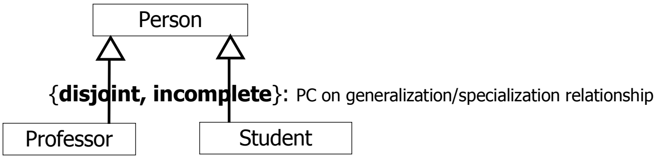

Person
{disjoint, incomplete}: PC on generalization/specialization relationship
Professor
Student

Fig. 1. Human resources management schema

3.1 Translation of Generalization/Specialization Relationships

Generalization/specialization relationships naturally arise from superclass/subclass hierarchies in semantic data modeling [8]. In this kind of relationship, every object can belong to the generalization as well as to its specialization. Each special object has a link with exactly one general object but the reverse does not obligatory hold. This kind of relationship may be represented as a one-to-one relationship. A multiplicity "1" on the side of the generalization which means that a special object is obligatory related to one and only one general object. On the side of the specialization, a multiplicity "0..1" which means that a general object may relates at most one special object.

3.2 Checking PCs on Generalization/Specialization Relationships

Inheritance constraints are divided into disjoint and complete constraint (Fig.2). Disjoint constraint specifies whether two objects of different specializations may be related to the same object of the generalization. Therefore, we say that an inheritance constraint is disjoint if each general object has a link to at most one special object. In the reverse case, constraint is said overlapping. To check disjoint constraint, we must ensure that each general object is member of at most one specialization. Complete constraint specifies whether objects of the specifications are related to all general objects. So, we say that an inheritance constraint is complete if each general object has a link to at least one special object (but not in the same specialization), else it is said incomplete. Inheritance constraint can be disjoint and complete at the same time. In this case each general object must be related to at least one special object but at most one object in both specializations.

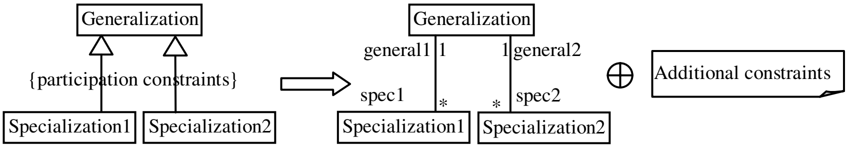


Fig. 2. PCs on generalization/specialization relationship

Example

In this example, we show how to translate disjoint and complete constraints when they are defined together on generalization/specialization relationships. In this case, we can do it in two different ways; using disjoint and complete translation together or the following translation.

OCL

```
Context g: generalization inv:
Self.allInstances→forAll(g| g.spec1→notEmpty xor g.spec2→notEmpty
```

SQL

```
Trigger1
event: insert on Generalization
condition: none
action: insert obligatory object
in one in only Specialization
```

```
Trigger2
event: delete on Specialization1
condition: none
action: delete object from generaliza-
tion or insert it in Specialization2
```

```
Trigger3
event: update on Specialization1
condition: none
action: delete new object from Specialization2 if it exists there or
         reject operation
        delete old object from Generalization or insert it in Spe-
        cialization2
```

4 Conclusion and Perspectives

In the literature, two categories of studies can be found. The first one concerns the formal and/or semi-formal translation. Among these studies, [9] discusses the expression of structure properties in a UML class diagram through UML basic structures and OCL. [10, 13] present how to translate UML diagrams into formal specification Z and B. The second category groups studies on checking the constraints of the target schema. Among these, we can find those which deal with multiplicity constraints using assertions [4] or triggers [1]. We also find [2] which uses OCL to translate participation constraints defined on binary relationships. Our major aim is to study both categories.

In this paper, we reported a systematic study of the use of participation constraints for the specification of assertions defined on the behavior of superclass/subclass object participations. The use of these constraints in a conceptual schema is necessary to satisfy the customer requirement. Our aim is to remove ambiguities from the definition of participation constraints. Though these later have, on binary relationships as well as on generalization, the same definitions, their semantics is not the same. We have translated participation constraints using OCL and riggers-based SQL additional constraints to cover object and relational models.

References

1. Al-Jumaily, H.T., Cuadra, D., Martinez, P. "Plugging Active Mechanisms to Control Dynamic Aspects Derived from the Multiplicity Constraint in UML. The workshop of 7th International Conference on the Unified Modeling Language, Portugal (2004) pages.
2. Berrabah, D., Boufarès, F., Ducateau, C.F.: Analysing UML Graphic Constraint, How to cope with OCL. 3rd International Conference on Computer Science and its Applications, California USA (2005).
3. Berrabah, D., Boufares, F., Ducateau, C. F., Gargouri, F.: Les conflits entre les contraintes dans les schémas conceptuels de Bases de Données: UML – EER. Journal of Information Sciences for Decision Making, Special Issue of the 8th MCSEAI'04, N°19 (2005) Paper number 234.
4. Boufarès, F.: Un outil intelligent pour l'analyse des schémas EA. Interne Report. Informatics Laboratory of Paris Nord, University of Paris 13 France (2001).
5. Ceri, S. and Widom, J.: Deriving production rules for constraint maintenance. In Proc. of the 16th International Conference on Very Large Data Bases, Brisbane Australia (1990) 566-577.
6. Cochrane, R.J., Pirahesh, H. and Mattos, N.M.: Integrating triggers and declarative constraints in SQL database systems. In Proceedings of the 22nd International Conference on Very Large Data Bases, Mumbai India (1996) 567-578.
7. Eisenberg, A., Melton, J., Kulkarni, K., Michels, J., Zemke, F.: SQL: 2003 has been published. ACM SIGMOD Record, Volume 33, Issue 1, March (2004).
8. Elmasri, R., Navathe, S.: Fundamentals of Database Systems. 4th ed., Addison-Wesley (2003).
9. Gogolla, M., Richters, M.: Expressing UML Class Diagrams Properties with OCL. Object Modeling with the OCL. Springer, (2002) 85-114.

10. Laleau, A., Mammar, A.: Overview of method and its support tool for generating B from UML notations. Proceeding of 15th international conference on Automated Software Engineering, Grenoble France (2000).
11. OMG, editor: UML 2.0 Object Constraint Language Specification. OMG (2005). http://omg.org.
12. Shroff, M., France, R. B.: Towards a Formalization of UML Class Structures in Z. 21st IEEE Annual international computer Software and Applications Conference (1997) 646-651.
13. Warmer, J., Kleppe, A. The Object Constraint Language: Getting Your Models Ready for MDA. 2nd Ed. Paperback-Edition (2003).
14. Rational: http://www-306.ibm.com/ software/ rational/ sw-bycategory/ subcategory/ SW710.html
15. Sybase: http://www.sybase.com/products/information management/powerdesigner.
16. Toby, J. T.: Database Modeling & Design. 3rd ed. Morgan, Kaufmann Series in data management systems (1999).

A New Trajectory Indexing Scheme for Moving Objects on Road Networks*

Jae-Woo Chang[1], Jung-Ho Um[1], and Wang-Chien Lee[2]

[1] Dept. of Computer Eng., Chonbuk National Univ., Chonju, Chonbuk 561-756, Korea
jwchang@chonbuk.ac.kr, jhum@dblab.chonbuk.ac.kr
[2] Dept. of CS&E., Pennsylvania State Univ., University Park, PA 16802

Abstract. In this paper, we propose an efficient signature-based indexing scheme for efficiently dealing with the trajectories of current moving objects on road networks. We show that our indexing scheme achieves much better trajectory retrieval performance than the existing trajectory indexing schemes, such as TB-tree, FNR-tree and MON-tree.

1 Introduction

Even though most of the existing work on spatial databases considers Euclidean spaces, objects in practice can usually move on road networks, where the network distance is determined by the length of the real shortest path on the network. For example, a gas station nearest to a given point in Euclidean spaces may be more distant in a road network than another gas station. Therefore, the network distance is an important measure in spatial network databases (SNDB). Meanwhile, there have been a couple of studies on trajectory indexing schemes for both Euclidean spaces and spatial networks (i.e., roads) [PJT00, F03, AG05]. First, Pfoser et al. [PJT00] proposed a hybrid index structure which preserves trajectories as well as allows for R-tree typical range search in Euclidean spaces, called TB-tree (Trajectory-Bundle tree). The TB-tree has fast accesses to the trajectory information of moving objects, but it has a couple of problems in SNDB. First, because moving objects move on a predefined spatial network in SNDB, the paths of moving objects are overlapped due to frequently used segments, like downtown streets. Secondly, because the TB-tree constructs a three-dimensional MBR including time, the dead space for the moving object trajectory can be highly increased. Next, Frentzos [F03] proposed a new indexing technique, called FNR-tree (Fixed Network R-tree), for objects constrained to move on fixed networks in two-dimensional space. Its general idea is to construct a forest of 1-dimensional (1D) R-trees on top of a 2-dimensional (2D) R-tree. The 2D R-tree is used to index the spatial data of the network while the 1D R-trees are used to index the time interval of each object movement inside a given link of the network. The FNR-tree outperforms the R-tree in most cases, but it has a critical drawback that the FNR-tree has to maintain a tremendously large number of R-trees. This is because it constructs as large number of R-trees as the total number of segments in the networks. Finally, Almeida and Guting proposed a new index structure for moving objects on

* This work is financially supported by the Ministry of Education and Human Resources Development(MOE), the Ministry of Commerce, Industry and Energy(MOCIE) and the Ministry of Labor(MOLAB) though the fostering project of the Lab of Excellency.

D. Bell and J. Hong (Eds.): BNCOD 2006, LNCS 4042, pp. 291 – 294, 2006.

network, called MON-tree, for both edge-oriented and route-oriented models. The MON-tree outperforms the FNR-tree in both updating and querying, but it has a drawback that the MON-tree should still maintain a very large number of R-trees, like the FNR-tree.

2 Trajectory Indexing Scheme for Current Moving Objects

To overcome the weaknesses of the existing schemes, we propose a new trajectory indexing scheme for moving objects on road networks, which is based on a signature file technique for efficiently dealing with the trajectories of moving objects. Figure 1 shows the structure of our trajectory indexing scheme. Our main idea is to create a signature of a moving object trajectory and maintain partitions which store the fixed number of moving object trajectories and their signatures together in the order of their start time. The main reason to use partitions is that because a partition is created and maintained depending on its start time, it is possible to efficiently retrieve the trajectories of moving objects on a given time. So, our trajectory indexing scheme has the following advantages. First, our indexing scheme is not affected by the overlap of moving objects' paths and never causes the dead space problem because it is not a tree-based structure like TB-tree. Secondly, our indexing scheme well supports a complex query containing a partial trajectory condition since it generates signatures using a superimposed coding. Finally, our indexing scheme can achieve very good insertion performance because it does not maintain a large number of trees, like FNR-tree and MON-tree. Our trajectory indexing scheme consists of a partition table and a set of partitions. A partition can be divided into three areas; trajectory information, location information, and signature information. A partition table for maintaining a set of partitions to store trajectories can be resided in a main memory due to its small size. To achieve good retrieval performance, we also store both the signature and the

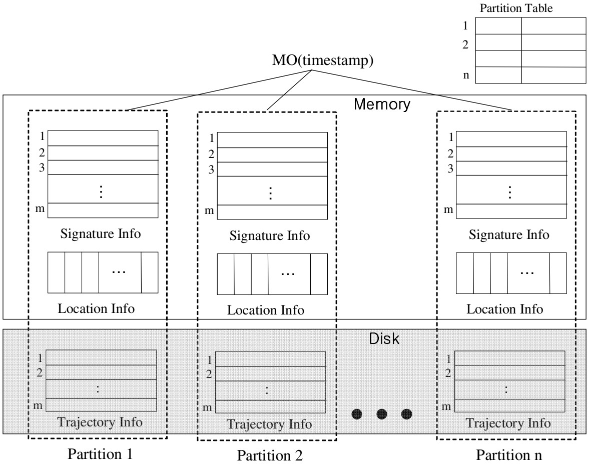


Fig. 1. Structure of our trajectory indexing scheme

location information in a main memory because of their relatively small size. To answer a user query, we find partitions to be accessed by searching the partition table. The trajectory information area maintains moving object trajectories which consist of a set of segments (or edges). The location information area contains the location of an object trajectory stored in the trajectory information area. This allows for accessing the actual object trajectories corresponding to potential matches to satisfy a query trajectory in the signature information area. To construct our trajectory indexing scheme in an efficient manner, we make use of a superimposed coding because it is very suitable to SNDB applications where the number of segments for an object trajectory is variable [ZMR98].

3 Performance Analysis

We implement our trajectory indexing scheme under Pentium-IV 2.0GHz CPU with 1GB main memory. For our experiment, we use a road network consisting of 170,000 nodes and 220,000 edges [WMA]. We also generate 50,000 moving objects randomly on the road network by using Brinkhoff's algorithm [B02]. For performance analysis, we compare our indexing scheme with TB-tree, FNR-tree and MON-tree, in terms of insertion time, storage space, and retrieval time for moving object trajectories. First, Table 1 shows insertion times to store a moving object's trajectory. It is shown that our indexing scheme preserves nearly the same insertion performance as TB-tree, while it achieves about two orders of magnitude better insertion performance than FNR-tree and MON-tree. This is because both FNR-tree and MON-tree construct an extremely great number of R-trees. Secondly, we measure storage space for storing moving object trajectories, as shown in Table 1. It is shown that our indexing scheme requires nearly the same storage space as TB-tree, while it needs about one fifth of the storage space required for FNR-tree and MON-tree.

Table 1. Trajectory insertion time and storage space

	TB-tree	FNR-tree	MON-tree	Our indexing scheme
Trajectory insertion time(sec)	0.7488	344	260	0.6552
Storage space(MB)	4.42	20.2	23.75	4.87

Finally, we measure retrieval time for answering queries whose trajectory contains 2 to 20 segments, as shown in Figure 2. It is shown that our indexing scheme requires about 9 ms while MON-tree, FNR-tee and the TB-tree needs 15ms, 21ms, and 630ms, respectively, when the number of segments in a query is 2. It is shown that our indexing scheme outperforms the existing schemes when the number of segments in a query trajectory is small. The TB-tree achieves the worst retrieval performance due to a large extent of overlap in its internal nodes. As the number of segments in queries increase, the retrieval time is increased in the existing tree-based schemes; however, our indexing scheme requires constant retrieval time. The reason is why our indexing scheme creates a query signature combining all the segments in a query and it searches for potentially relevant trajectories of moving objects once by using the query signature as a filter. When the number of segments in a query is 20, it is shown that our indexing scheme requires about 9 ms while MON-tree, FNR-tree and TB-tree needs 108ms, 157ms and

1.3s, respectively. Thus our indexing scheme achieves at least one order of magnitude better retrieval performance than the existing schemes. This is because our indexing scheme constructs an efficient signature-based indexing structure by using a superimposed coding technique. On the contrary, because the TB-tree builds a MBR for each segment in a query and the number of range searches increases in proportion to the number of segments, the TB-tree dramatically degrades on trajectory retrieval performance when the number of segments is great. Similarly, because both MON-tree and FNR-tree search for an R-tree for each segment in a query, they degrade on retrieval performance as the number of segments in the query is increased.

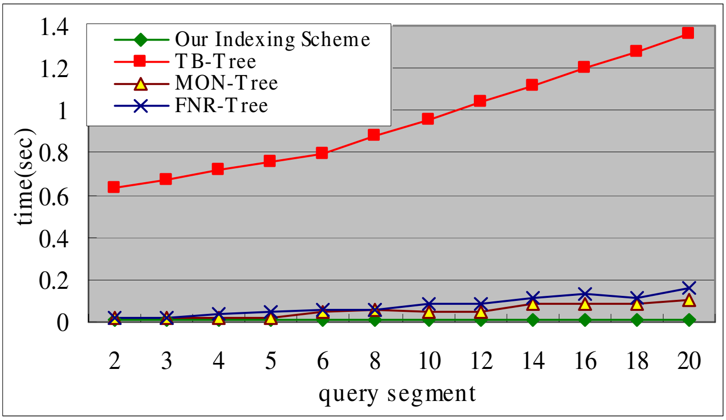


Fig. 2. Trajectory retrieval performance

4 Conclusions

We proposed an efficient signature-based indexing scheme for current moving objects' trajectories on road networks. We showed that our indexing scheme achieved at least one order of magnitude better retrieval performance than the existing trajectory indexing schemes, such as TB-tree, FNR-tree and MON-tree.

References

[AG05] V.T. Almeida and R.H. Guting, "Indexing the Trajectories of Moving Objects in Networks," GeoInformatica, Vol. 9, No. 1, pp 33-60, 2005.

[B02] T. Brinkhoff, "A Framework for Generating Network-Based Moving Objects," GeoInformatica, Vol. 6, No. 2, pp 153-180, 2002.

[F03] R. Frentzos, "Indexing Moving Objects on Fixed Networks," Proc. of Int'l Conf on Spatial and Temporal Databases (SSTD), pp 289-305, 2003.

[PJT00] D. Pfoser, C.S. Jensen, and Y. Theodoridis, "Novel Approach to the Indexing of Moving Object Trajectories," Proc. of VLDB, pp 395-406, 2000.

[WMA] http://www.maproom.psu.edu/dcw/

[ZMR98] J. Zobel, A. Moffat, and K. Ramamohanarao, "Inverted Files Versus Signature Files for Text Indexing," ACM Tran. on Database Systems, Vol. 23, No. 4, pp 453-490, 1998.

Author Index