SIGMOD Record – March 2006 – Volume 35 Issue 1 – www.sigmod.org/record

[**Editor's note**: With the exception of the last pages –which would be the back cover of the printed issue– that are not included in this file, it has the same contents as the printed edition. All the articles are also available individually online and have been put together here for convenience only.]

SIGMOD Record

SIGMOD Record is a quarterly publication of the Special Interest Group on Management of Data (SIGMOD) of the Association for Computing Machinery (ACM). SIGMOD is dedicated to the study, development, and application of database and information technology. *SIGMOD Record Web Edition* is also freely available online at `http://www.sigmod.org/record`.

SIGMOD Record solicits contributions of articles, technical notes, reports, and proposals for special sections. Conference announcements and calls for papers are published if relevant to the interests of the group and, in most cases, are limited to one page. All contributions should be sent to the editor for consideration. Submitted technical papers are reviewed for importance and correctness. Priority is given to papers that deal with current issues of interest to a broad audience. Papers should be submitted electronically in PDF format to `record@sigmod.acm.org`, and they should follow a format similar to that of the SIGMOD conference proceedings (but with a larger font): 10 point font, single-space, 2-column, 8.5" by 11" page size with 1" margins all around and no page numbers. They should also be formatted for letter size pages and must contain all fonts embedded. Submitted articles are limited to 6 pages unless prior agreement of the editor is obtained.

Mario A. Nascimento, *SIGMOD Record* Editor.
`record@sigmod.acm.org`
Dept. of Computing Science, University of Alberta
Edmonton, AB, Canada

Associate Editors:

Ugur Çetintemel (Research Centers), `ugur@cs.brown.edu`
Brian Cooper, Georgia Institute of Technology (Articles, Reports, Notes), `cooperb@cc.gatech.edu`
Andrew Eisenberg, IBM Corporation (Standards), `andrew.eisenberg@us.ibm.com`
Cesar Galindo-Legaria (Research Surveys), `cesarg@microsoft.com`
Alexandros Labrinidis, University of Pittsburgh (Web Edition), `labrinid@cs.pitt.edu`
Leonid Libkin, University of Toronto (Database Principles), `libkin@cs.toronto.edu`
Jim Melton, Oracle Corporation (Standards), `jim.melton@acm.org`
Jignesh Patel, Univ. of Michigan, Ann Arbor (Systems and Prototypes), `jignesh@eecs.umich.edu`
Ken Ross, Columbia University (Influential Papers), `kar@cs.columbia.edu`
Len Seligman, The MITRE Corporation (Industry Perspectives), `seligman@mitre.org`
Marianne Winslett, University of Illinois (Distinguished DB Profiles), `winslett@cs.uiuc.edu`

SIGMOD Record (ISSN 0163-5808) is published quarterly by the Association for Computing Machinery, Inc., 1515 Broadway, New York, NY 10036. Periodicals postage paid at New York, NY 10001, and at additional mailing offices. POSTMASTER: Send address changes to *SIGMOD Record*, ACM, 1515 Broadway, New York, NY 10036.

Visit the SIGMOD Online website at `http://www.acm.org/sigmod`

SIGMOD Officers, Committees, and Awardees

Chair
Raghu Ramakrishnan
Department of Computer Sciences
University of Wisconsin-Madison
1210 West Dayton Street
Madison, WI 53706-1685
USA
raghu@cs.wisc.edu

Vice-Chair
Yannis Ioannidis
University Of Athens
Department of Informatics & Telecom
Panepistimioupolis, Informatics Bldngs
157 84 Ilissia, Athens
HELLAS
yannis@di.uoa.gr

Secretary/Treasurer
Mary Fernández
ATT Labs - Research
180 Park Ave., Bldg 103, E277
Florham Park, NJ 07932-0971
USA
mff@research.att.com

Information Director: Alexandros Labrinidis, University of Pittsburgh, labrinid@cs.pitt.edu.

Associate Information Directors: Manfred Jeusfeld, Dongwon Lee, Michael Ley, Frank Neven, Altigran Soares da Silva, Jun Yang.

Advisory Board: Tamer Ozsu (Chair), University of Waterloo, tozsu@cs.uwaterloo.ca, Rakesh Agrawal, Phil Bernstein, Peter Buneman, David DeWitt, Hector Garcia-Molina, Jim Gray, Masaru Kitsuregawa, Jiawei Han, Alberto Laender, Krithi Ramamritham, Hans Schek, Rick Snodgrass, and Gerhard Weikum.

SIGMOD Conference Coordinator: Jianwen Su, UC Santa Barbara, su@cs.ucsb.edu

SIGMOD Workshops Coordinator: Laurent Amsaleg, IRISA Lab, Laurent.Amsaleg@irisa.fr

Industrial Advisory Board: Daniel Barbará (Chair), George Mason Univ., dbarbara@isse.gmu.edu, José Blakeley, Paul G. Brown, Umeshwar Dayal, Mark Graves, Ashish Gupta, Hank Korth, Nelson M. Mattos, Marie-Anne Neimat, Douglas Voss.

SIGMOD Record Editorial Board: Mario A. Nascimento (Editor), University of Alberta, mn@cs.ualberta.ca, José Blakeley, Ugur Çetintemel, Brian Cooper, Andrew Eisenberg, Leonid Libkin, Alexandros Labrinidis, Jim Melton, Len Seligman, Jignesh Patel, Ken Ross, Marianne Winslett.

SIGMOD Anthology Editorial Board: Curtis Dyreson (Editor), Washington State University, cdyreson@eecs.wsu.edu, Nick Kline, Joseph Albert, Stefano Ceri, David Lomet.

SIGMOD DiSC Editorial Board: Shahram Ghandeharizadeh (Editor), USC, shahram@pollux.usc.edu, A. Ailamaki, W. Aref, V. Atluri, R. Barga, K. Boehm, K.S. Candan, Z. Chen, B. Cooper, J. Eder, V. Ganti, J. Goldstein, G. Golovchinsky, Z. Ives, H-A. Jacobsen, V. Kalogeraki, S.H. Kim, L.V.S. Lakshmanan, D. Lopresti, M. Mattoso, S. Mehrotra, R. Miller, B. Moon, V. Oria, G. Ozsoyoglu, J. Pei, A. Picariello, F. Sadri, J. Shanmugasundaram, J. Srivastava, K. Tanaka, W. Tavanapong, V. Tsotras, M. Zaki, R. Zimmermann.

SIGMOD Digital Review Editorial Board: H. V. Jagadish (Editor), Univ. of Michigan, jag@eecs.umich.edu, Alon Halevy, Michael Ley, Yannis Papakonstantinou, Nandit Soparkar.

Sister Society Liaisons: Stefano Ceri (VLDB Foundation and EDBT Endowment), Hongjun Lu (SIGKDD and CCFDBS), Yannis Ioannidis (IEEE TCDE), Serge Abiteboul (PODS and ICDT Council).

Latin American Liaison Commitee: Claudia M. Bauzer Medeiros (Chair), University of Campinas, cmbm@ic.unicamp.br Alfonso Aguirre, Leopoldo Bertossi, Alberto Laender, Sergio Lifschitz, Marta Mattoso, Gustavo Rossi.

Awards Committee: Moshe Y. Vardi (Chair), Rice University, vardi@cs.rice.edu. Rudolf Bayer, Masaru Kitsuregawa, Z. Meral Ozsoyoglu, Pat Selinger, Michael Stonebraker.

Award Recipients:

Innovation Award: Michael Stonebraker, Jim Gray, Philip Bernstein, David DeWitt, C. Mohan, David Maier, Serge Abiteboul, Hector Garcia-Molina, Rakesh Agrawal, Rudolf Bayer, Patricia Selinger, Don Chamberlin, Ronald Fagin.

Contributions Award: Maria Zemankova, Gio Wiederhold, Yahiko Kambayashi, Jeffrey Ullman, Avi Silberschatz, Won Kim, Raghu Ramakrishnan, Laura Haas, Michael Carey, Daniel Rosenkrantz, Richard Snodgrass, Michael Ley, Surajit Chaudhuri.

Editor's Notes

As I type this text, 2006 is only a few days old, and as such, I wish we all accomplish the successes we aim at this New Year.

The first thing I want to communicate to you is that José Blakeley is retiring from his position as Associate Editor (Research Surveys). I remember the first time I met him in 1995. At that time he was already an Associate Editor; he has been in this role for over 10 years, and even he cannot remember exactly for how long he has been doing it. Besides being, in all likelihood, the longest running Associate Editor ever, as well as doing a fantastic job in such position, José is well known in our communit. I want to thank him very much for all the service he has done for SIGMOD's community and for the *Record*.

To serve as the new Associate Editor (Research Surveys) I have invited, Cesar Galindo-Legaria, and he has kindly accepted the invitation. Cesar is the development manager of the query optimizer in the Microsoft SQL Server product and well published in the database literature. I am confident Cesar will continue the tradition José built, and keep the *Record* a good venue for publishing thorough and informative survey papers, which some have told me to be an important feature in the *Record*.

As you may recall from previous issues, Jignesh M. Patel and Karl Aberer have also retired from their positions as Associate Editors (System/Prototypes and Book Reviews, respectively) and I am still in the process of finding colleagues to take over their positions. It is not easy to do so as many of the good candidates I have talked to are simply too busy to commit to yet something else. But I am sure this is just a matter of time until we have the full board of Associate Editors rebuilt. In fact, by the time you read this I hope this is not longer a concern.

As usual, I am not going to introduce each of the papers in this issue, but I can tell you that it covers a number of different topics. As an editor I am happy to see that happening. On the one hand, it shows the broadness of our domain and, on the other hand, it also allows almost everyone to find at least one article close to his/her research interests.

There is not much more news or updates from my part at this time (which I think is good news). Let me conclude by transmitting message from Raghu (who was unable to write the usual Chair's Message for this issue): "several issues are developing and a full update will be forthcoming in the next issue." Until then I hope you enjoy this issue.

Mario Nascimento, Editor.
October 2005

Exploiting Predicate-window Semantics over Data Streams

Thanaa M. Ghanem Walid G. Aref Ahmed K. Elmagarmid

Department of Computer Sciences, Purdue University, West Lafayette, IN 47907-1398
{ghanemtm,aref,ake}@cs.purdue.edu

ABSTRACT

The *continuous sliding-window* query model is used widely in data stream management systems where the focus of a continuous query is limited to a set of the most *recent* tuples. In this paper, we show that an interesting and important class of queries over data streams cannot be answered using the *sliding-window* query model. Thus, we introduce a new model for continuous window queries, termed the *predicate-window* query model that limits the focus of a continuous query to the stream tuples that qualify a certain predicate. *Predicate-window* queries have some distinguishing characteristics, e.g., (1) The window predicate can be defined over any attribute in the stream tuple (ordered or unordered). (2) Stream tuples qualify and disqualify the window predicate in an out-of-order manner. In this paper, we discuss the applicability of the predicate-window query model. We will show how the existing *sliding-window* query models fail to answer some of the *predicate-window* queries. Finally, we discuss the challenges in supporting the *predicate-window* query model in data stream management systems.

1. INTRODUCTION

The emergence of data streaming applications calls for new query processing techniques to cope with the high rate and unbounded nature of data streams. Queries over data streams are characterized by the following: (1) Most of the queries in the streaming environment are continuous. Continuous queries need continuous reevaluation as new tuples arrive, and (2) Usually, queries are interested only in a specific part (*window-of-interest*) of the received data. The sliding-window query model [1] is introduced to answer continuous queries that are interested only on the most recent stream tuples. There are two common types of sliding-windows: Time-based sliding window (e.g., tuples in the last hour) and tuple-based sliding window (e.g., the last 100 tuples). Window-aware operators (e.g., window-join [3, 5, 6] and window-aggregates [7]) are modifications of their counterpart traditional operators to support sliding-window queries. The main difference in window-aware query operators is the need to process tuples expired from the window as well as new tuples incoming into the window.

1.1 Motivation

Continuous *sliding-window* queries over data streams have been introduced to limit the focus of a continuous query to a specific part (*window-of-interest*) of the incoming stream tuples. The *window-of-interest* in the sliding-window query model includes the most-recent input tuples. In a sliding-window query over n input streams, S_1 to S_n, a window of size w_i is defined over the input stream S_i. The sliding-window w_i can be defined over any ordered attribute *attr* in the stream tuple (e.g., a timestamp or a sequence number). As the window slides, the query answer is updated to reflect both the new tuples entering the sliding-window and the old tuples expiring from the sliding-window. Tuples enter and expire from the sliding-window in a First-In-First-Expire (FIFE) fashion.

An interesting and important class of queries is not supported by the sliding-window query model. Consider a continuous query that is interested only in the input tuples that qualify a certain predicate p, where p is defined over an unordered attribute. For example, consider a temperature monitoring application in which a large number of sensors are spatially distributed, and each sensor sends continuously its current temperature. A common query in this environment is: Q_1 *"Continuously, report the sensor identifiers for sensors that have temperature greater than 90"*. At any time point T', the window-of-interest for Query Q_1 includes only the sensors that qualify the predicate "*temperature greater than 90*". If a sensor S reports a temperature greater than 90, then S should be considered in Q_1's window. Whenever S reports another temperature that disqualifies the predicate "*temperature greater than 90*", S expires from Q_1's window. Notice that sensors enter and expire from Q_1's window in an out-of-order manner. A sensor expires (and hence is deleted) from Q_1's window only when the sensor reports another temperature that disqualifies the window predicate.

To utilize the sliding-window query model, the query semantics reads as follows: Q_2: *"Continuously, report sensor identifers for sensors that have temperature greater than 90 in the last T time units"*, where T is the size of the sliding-window. The Query Q_2 is semantically different from Query Q_1. The main difference between the two queries is that the window-of-interest in Q_1 includes "*sensors having temperature greater than 90*" while the window-of-interest in Q_2 includes "*sensors that have reported temperature greater than 90 in the last T time units*".

Example 1: This example illustrates the difference between Q_1 and Q_2 (with T=5). Consider the temperature monitoring application where the input stream has the schema <SensorID, Temperature, TimeStamp>. Assume that the following input tuples have arrived <2,88,1> <2,92,2> <3,91,3> <1,95,4> <2,89,5> <3,95,6>. Q_1 and Q_2 produce the following output: (1) When tuple <2,92,2> arrives, Sensor 2 is reported in the answer. Similarly, when

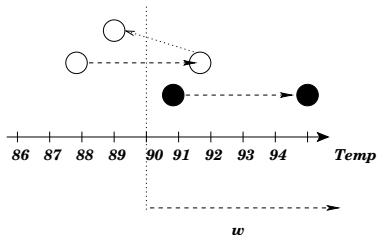


Figure 1: Example 1

tuple <3,91,3> arrives, Sensor 3 is reported in the query answer. Later, when tuple <2,89,5> arrives, Sensor 2 expires (is deleted) from the answer. On the other hand, when tuple <3,95,6> arrives, Sensor 3 is not deleted from the query answer since Sensor 3 still qualifies the window predicate only with a different temperature. Figure 1 gives the behavior of Sensors 2 (white circles) and Sensor 3 (black circles) in query Q_1. (2) In Query Q_2, when tuples <2,92,2> and <3,91,3> arrive, Sensors 2 and 3 are reported in the answer. Later, when tuple <2,89,5> arrives, the answer will not be affected since temperature 89 disqualifies the predicate. When tuple <3,95,6> arrives, Sensor 3 will be reported again in the query answer. To summarize, at time 6, the answer to Q1 is Sensors 3 and 1, because these are the sensors with temperature greater than 90 at time 6. In contrast, the answer to Q2 is Sensors 2, 3 and 1. Sensor 2 appears in the answer of Q2, because Sensor 2 reports a temperature greater than 90 once in its history in the past 5 time units. Notice that Sensor 2 will expire from Q_2 at time 7 (when tuple <2,92,2> is 5 time units old).

1.2 The Negative Tuples Approach

In the rest of this paper we assume that the pipelined query execution model with the negative tuples approach [1, 2] is used to process *window* queries over data streams. The pipelined query execution model for data streams is a modification of the one used in traditional database management systems [2] where all query operators are connected via first-in-first-out queues. In the negative tuples approach, a special operator, termed EXPIRE, is added at the bottom of the query pipeline; one EXPIRE operator per data stream. EXPIRE buffers the input stream tuples, and outputs a negative tuple whenever a tuple expires from the window. The negative tuple is processed by the various operators in the query pipeline to negate the effect (if any) of the corresponding positive tuple. The output of the continuous query is a continuous stream of positive and negative tuples. A negative tuple is interpreted as a deletion of a previously produced positive tuple.

2. THE PREDICATE-WINDOW QUERY MODEL

The predicate-window query model is a generalization over the sliding-window query model where the former supports a larger class of continuous queries over data streams. The window-of-interest for the predicate-window includes the input stream tuples that satisfy a given predicate.

Assumptions: In the predicate window query model, we have the following assumptions:

- Each input stream tuple t has a *correlation* attribute $t.CORAttr$. The input stream tuples with the same value of the correlation attribute are correlated together as follows. If a later tuple t_n carries the same values of the correlation attribute as that of t, then t_n is considered an update over t. In Example 1, the correlation attribute is $SensorID$. Therefore, tuple <2,89,5> is an update over tuple <2,91,2>.
- There is no regular pattern for updates. In Example 1, some sensors may send their readings every fixed time interval and some other sensors send their readings whenever a change in temperature is detected.

2.1 Continuous Predicate-window Query Semantics

A predicate-window query Q is defined over n data streams S_1 to S_n and n window predicates P_1 to P_n where the window predicate P_i is defined over the tuples in stream S_i. At any point in time T, the answer to Q equals the answer to a snap-shot query Q', where Q' is issued at time T and the inputs to Q' are the tuples in stream S_i that qualify the predicate P_i at time T.

Assume that an input tuple t_i from stream S_i has the following schema: $t_i < CORAttr, PAttrs, Attrs >$, where $CORAttr$ is the correlation attribute, $PAttrs$ are the attributes over which the predicate P_i is defined and $Attrs$ are the other attributes. A tuple t_i qualifies the predicate P_i at time T, iff: (1) t_i arrives in the stream at point in time before T, (2) $t_i.PAttrs$ qualifies P_i and (3) There is no stream tuple t_i' that arrives after t_i and $t_i'.CORAttr = t_i.CORAttr$.

2.2 Syntax and Operators

We represent the predicate-window by adding a new construct, termed PWINDOW, to SQL. The syntax for PWINDOW is as follows:

PWINDOW <predicate> ON <CORAttr>

where <*predicate*> is the predicate that qualifies (and disqualifies) tuples into (and out of) the window and <*CORAttr*> is one or more attributes that correlate incoming stream tuples.

Example 1 revisited: The following is the SQL syntax for the query Q_1 in Example 1:

SELECT S.SensorID
FROM Sensors S
[*PWINDOW S.Temperature > 90 ON S.SensorID*]

A new operator PWINDOW needs to be incorporated in the stream query engine. The PWINDOW operator is a generalization of the EXPIRE operator. PWINDOW is placed at the bottom of the query pipeline (Figure 2). PWINDOW encapsulates the window predicate (or multiple predicates) and applies it on every incoming stream tuple. PWINDOW is responsible for notifying the query pipeline by any changes in the window contents. PWINDOW is a statefull operator that needs to keep all tuples currently in the window in its state H. PWINDOW produces three different types of tuples:

1. **Positive Tuple (t^+)**: When a new incoming stream tuple t qualifies the window predicate and $t.CORAttr$ is not in H, PWINDOW inserts t in H and output a positive tuple for t.
2. **Update Tuple (t^u)**: When a new incoming stream tuple t qualifies the window predicate and $t.CORAttr$

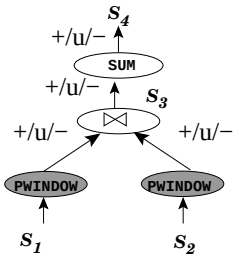


Figure 2: The PWINDOW operator

is already in H, PWINDOW updates the attributes of t in H and produces an update tuple for t as output.

3. **Negative Tuple (t^-):** When a new incoming stream tuple t does not qualify the window predicate and $t.CORAttr$ is in H, PWINDOW deletes t from H and produces a negative tuple for t as output.

Different operators in the query pipeline will be furnished by methods to process the different types of tuples. The output of the query is a stream of positive, update, and negative tuples. An update tuple is interpreted as a replacement for the previously produced positive tuple with the same values of the correlation attributes. The negative tuple is interpreted as a deletion of the previous positive (or update) tuple with the same values of the correlation attributes.

3. A COMPARISON WITH THE EXISTING WINDOW APPROACHES

In this section, we show how the existing sliding-window query approaches fail to answer some of the predicate-window queries. We use query Q_1 (from Example 1) as a running example.

3.1 WHERE Clause

The main difference between a predicate in PWINDOW and a predicate in the *where-clause* is that a disqualified tuple in the PWINDOW predicate may result in a negative tuple as an output while a disqualified tuple in the *where-clause* predicate does not result in any output tuples. We illustrate the difference between the *window* predicate and the *where* predicate by the following example. Consider a SQL query that is similar to Q_1 but the *window* predicate is expressed in the *where* predicate as follows:
SELECT S.SensorID
FROM Sensors S
WHERE S.Temperature > 90
If this query is continuously running as the stream tuples arrive, at time 2, when tuple <2,92,2> arrives, Sensor 2 is reported in the query answer. Later, when tuple <2,89,5> arrives, and since the temperature 89 disqualifies the *where* predicate, tuple <2,89,5> does not affect the query answer. The output from the SQL query with the *where* predicate is different from the expected output of Q_1. In Q_1, although tuple <2,89,5> disqualifies the window predicate, tuple <2,89,5> results in an output negative tuple to expire Sensor 2 from the query answer.

The *where* predicate cannot be used to express predicate-window queries. When a tuple t qualifies the *where* predicate and is reported in the query answer, t will remain in the query answer for ever. In the predicate-window query model, when a tuple t qualifies the window predicate and is reported in the query answer, later, t may be deleted from the query answer if t receives an update so that t does not qualify the window predicate any more.

3.2 Sliding-windows

The sliding-window query model fails to answer some of the predicate-window queries (as shown in Example 1). The sliding-window query model is characterized by the following: (1) A window with size w is defined over an ordered attribute in the stream schema (e.g., a timestamp or a sequence number) and (2) Tuples enter and expire from the sliding-window in a First-In-First-Expire (FIFE) fashion. Some of the predicate-window queries do not follow the characteristics of the sliding-window query model. For example, consider query Q_1. The window predicate is defined over the unordered attribute temperature. There is no window size for the sliding-window that can emulate the behavior of Q_1. Moreover, in Q_1, tuples enter and expire from the predicate-window in an out-of-order manner. A tuple expires from the predicate window whenever the tuple receives an update that disqualifies the window predicate. Due to the different characteristics, some of the predicate-window queries cannot be answered using the sliding-window query model.

3.3 Partitioned Sliding-windows

Partitioned sliding-window queries have been introduced and used by several data stream management systems [1, 7]. A partitioned sliding-window partitions the input stream into sub-streams and the sliding-window is applied on each sub-stream independently. The windows of the various sub-streams are merged to produce the final query answer. The CQL clause for the partitioned-window is as follows:
$PARTITION BY$ <*PARAttr*>
ROWS <w>
$WHERE$ <*predicate*>
where <*PARAttr*> is the partitioning attribute, <w> is the sub-stream sliding-window size, and <*predicate*> is an optional window filter.

The "*partition by* <*PARAttr*>" in the partitioned-window clause is similar to the "*ON* <*CORAttr*>" in the PWINDOW clause. The two clauses aim to group input stream tuples having the same value of some attribute. Although having some similarities, we show that partitioned sliding-windows fail to answer some predicate-window queries.

A partitioned sliding-window query may have two classes of predicates as follows: (1) Partitioned-window predicates (*where* <*predicate*> in the PARTITION BY clause) and (2) Query predicates (the outer *where* clause in the query). The difference between the partitioned-window predicate and the query predicate is as follows. The partitioned-window predicate qualifies (and disqualifies) tuples into (and out-of) the window for each sub-stream. In this case, the window size is calculated over the qualified tuples only. For example, if the window size is 3, then at any time point, the last 3 qualified tuples will be inside the window of the corresponding sub-stream. On the other hand, the query predicate qualifies (and disqualifies) tuples into (and out-of) the query answer. In this case, the window size is calculated over both qualified and disqualified tuples. For example, if the window size is 3, the last 3 tuples will be inside the

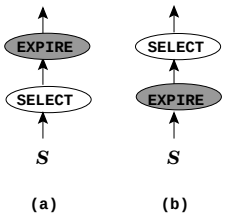


Figure 3: Partitioned sliding-window

window of the corresponding sub-stream. From these last 3 tuples, only the qualified tuples will be used in the query answer.

In the following, we show that both the partitioned-window query with window predicates and the partitioned-window query with query predicates are semantically different from the window predicate in the predicate-window query model.

3.3.1 Partitioned-window predicates

A partitioned-window clause partitions the stream into sub-streams. A partitioned-window predicate qualifies (and disqualifies) tuples into (and out-of) each sub-stream. Assume that a partitioned sliding-window Q_p query that is similar to Q_1 but with the window predicate "*temperature > 90*" is used as the partitioned-window filter. The CQL syntax for Q_p is as follows:
SELECT S.SensorID
FROM Sensors S
[*Partition By S.SensorID*
Rows 1
WHERE S.Temperature > 90]

The semantics of the query Q_p is as follows: *"For each sensor, continuously report the last reading with temperature > 90"*. The query pipeline for Q_p is shown in Figure 3a, where the window filter (the *select* operator) is applied before the window size (the *EXPIRE* operator). Let Q_p be a continuously running while the stream tuples arrive. At time 2, when tuple <2,92,2> arrives, Sensor 2 is reported in the query answer. Later, when tuple <2,89,5> arrives, since 89 disqualifies the selection filter, tuple <2,89,5> is filtered out and does not contribute to the window for Sensor 2 sub-stream. Tuple <2,92,2> expires from the window only when Sensor 2 reports another reading with temperature greater than 90. Notice that the output of Q_p is different from the output of the predicate-window query Q_1. In Q_1, when tuple <2,89,5> arrives, Sensor 2 expires from the query answer.

The window filter in Q_p is different from the window predicate in Q_1 in the following: the window filter in Q_p qualifies (and disqualifies) tuples into (and out of) the sub-streams. On the other hand, the window predicate in Q_1, qualifies (and disqualifies) sub-streams into (and out of) the query answer.

3.3.2 Query predicates

The other type of predicates in the partitioned-window query is the query predicate. The query predicate in a partitioned-window query qualifies tuples into the query answer. The window for each sub-stream may include both qualified and disqualified tuples. Consider a partitioned-window query Q_p' similar to Q_1 but with the window predicate used as a query predicate as follows:
SELECT S.SensorID
FROM Sensors S
[*Partition By S.SensorID*
ROWS 1]
WHERE S.Temperature > 90

The semantics for Q_p' is as follows: "*Continuously report the readings with temperature greater than 90 considering only the last reading for each sensor*". The pipeline for query Q_p' is given in Figure 3b where the window size (the *EXPIRE* operator) is applied before the query filter (the *select* operator). Assume that query Q_p' is continuously running when the stream tuples arrive. At time 3, tuple <3,91,3> arrives to the EXPIRE operator. Since it is the most recent reading for Sensor 3, tuple <3,91,3> will be forwarded to the *select* operator. Since 91 qualifies the selection predicate, Sensor 3 is produced in the query answer. Later, at time 6, tuple <3,95,6> arrives. Upon receiving <3,95,6>, since only the last reading for each sub-stream resides inside the window, the EXPIRE operator will emit two tuples: a negative tuple -<3,91,3> and a positive tuple <3,95,6>. Both tuples will be passed to the *select* operator. Both -<3,91,3> and <3,95,6> appear in the query answer. Notice that the semantics of the reception of -<3,91,3> then <3,95,6> is that Sensor 3 is deleted from the answer then Sensor 3 is reported again in the answer.

Q_p' query answer (including the deletion then insertion of Sensor 3) is semantically different from the predicate-window query Q_1. The semantics of the predicate-window query requires that at any point in time, the query answer includes all sensors satisfying the window predicate. Q_1 semantics does not hold in the time interval between the deletion and insertion of Sensor 3 in Q_p' window. The length of the time period for the semantically wrong answer is non-deterministic since tuples may encounter delays in the query pipeline. The problem can be worse if an aggregate operation (e.g., COUNT) is performed over the output tuples. Assume that another query is interested in the COUNT of sensors having temperature greater than 90. Assume that before tuple -<3,91,3>, the COUNT value was 10. Upon receiving -<3,91,3>, the COUNT operator will update its answer to 9. When tuple <3,95,6> is processed by the COUNT operator, a new count with value 10 will be produced. Notice that the count of value 9 should not appear in the query answer.

The previous examples shows that in the partitioned-sliding-window, the independent application of the partitioned-window clause and the where-clause is semantically different from the predicate-window query. The reason is that the independent evaluation of the window and the query predicates cannot capture the case of a tuple still being inside the window but only with a different value.

3.4 The NOW window

The keyword NOW defines a window over a data stream [1]. The NOW window means that at any point in time, say T, the answer of the query should include only the tuples that have a timestamp T. The NOW window is semantically different from the predicate-window query. Consider a query Q_n that is similar to Q_1 with the NOW window. The CQL syntax for Q_n is as follows:
SELECT S.SensorID

FROM Sensors S [NOW]
WHERE S.Temperature > 90
The semantics for Q_n is as follows: "*Report the sensors that have reported temperature greater than 90 NOW*". Assume that Q_n is continuously running when the input stream tuples arrive. At time 2, the query answer will include only Sensor 2 (because of the arrival of tuple <2,92,2>). Similarly, at time 3, the window will include only Sensor 3.

Query Q_n's answer is different from Q_1's answer. At any time point T, the NOW window includes only tuples that arrive at time T. On the other hand, at any time T, the predicate-window may include tuples that have arrived before T.

3.5 Punctuations

A punctuation is an artificial tuple, carrying a predicate p, that is inserted in the data stream to mark the end of a subsequence [9]. A punctuation tuple with predicate p arriving at time T means that no more tuples qualifying p will arrive later (after time T) in the input stream. The punctuation predicate does not carry any information about the input stream tuples that have arrived before time T and already have been used in generating the query answer. The tuples used in generating the query answer before the arrival of a punctuation p may include both tuples qualifying p and tuples disqualifying p.

The punctuation predicate cannot be used to represent the window predicate in the predicate-window query model. The reason is that before the arrival of a punctuation p, tuples disqualifying p may be included in the window-of-interest of a query. On the other hand, a window predicate, say wp, ensures that, at any time point, only tuples qualifying wp are included in the window-of-interest of the query. Due to the different semantics, punctuations fail to evaluate predicate-window queries.

4. TYPES OF PREDICATE-WINDOW QUERIES

The window predicate can take several other forms other than the selection predicate in Query Q_1. In this section, we discuss the various types of predicate-window queries.

4.1 Select predicate-window

In the *select* predicate-window type, the window predicate is a selection predicate that is defined over one attribute in the input stream. The selection predicate compares the incoming stream tuple against a constant (e.g., Temperature > 90).

4.2 Join predicate-window

The join predicate-window is a generalization of the select predicate-window. The join window predicate is defined over an attribute in the input stream tuple and compares the incoming stream tuple against a set of constants stored in a relational table.

Example: Consider the following scenario: Persons wearing RFID's are moving inside a building. Each RFID continuously reports the RoomID of the current location. Consider the following query: *"Continuously report the identifiers of persons located in one of the AlertRooms"*. The pre-defined set of alert rooms is stored in a relational table $AlertValues$. The window predicate in this query is a join predicate between the incoming stream and the $AlertValues$ table.

4.3 Dynamic predicate-window

In the *select* and the *join* predicate-windows, the *window predicate* is fixed and the updates cause tuples to qualify into (or disqualify from) the window. The *dynamic* predicate-window is another type of predicate-windows in which tuples expire from the window because the window predicate is continuously changing (e.g., current time).

Example: A sliding-window query is a dynamic predicate-window in which the window predicate is defined over the timestamp attribute. Consider the following sliding-window query: "*Continuously report the sensor identifiers for sensors that have reported a reading in the last T time units*". The same query can be rephrased as "*Continuously report the sensor identifiers for tuples that have a timestamp greater than $NOW - T$*". The SQL representation for this query is:
SELECT S.SensorID
FROM Sensors S
PWINDOW S.TimeStamp > NOW − T ON S.TimeStamp

4.4 IN/OUT predicate-window

In the previous sections, we introduced the predicate-window query model with one predicate defined in the PWINDOW clause. In this section, we introduce an extended predicate-window query model, namely the IN/OUT predicate-window model. The main idea in the IN/OUT predicate-window model is to distinguish between two predicates: (1) **IN window-predicate**: tuples that qualify the IN window-predicate will be considered by the query. (2) **OUT window-predicate**: when a tuple currently in the predicate-window qualifies the OUT window-predicate, then that tuple will expire from the window. The IN and OUT window predicates are different and are independent. The two predicates should not have any overlap (no stream tuple can satisfy both the IN and OUT predicates at the same time). In the predicate-window query model, the OUT window-predicate (implicitly) is the *complement* of the IN window predicate.

5. CHALLENGES IN REALIZING PREDICATE-WINDOWS IN DATA STREAM MANAGEMENT SYSTEMS

In this section, we discuss the challenges in realizing the predicate-window query model in data stream management systems.

5.1 Incremental Maintenance of the Query Answer

As discussed in Section 2.2, the PWINDOW operator is responsible for tracking changes in the window and emitting tuples accordingly (positive, update, or negative tuples). The output tuples from PWINDOW flow in the query pipeline and are processed by the various operators. The results of processing these tuples by the various operators are used to update the query answer incrementally. For each relational operator and for each tuple type, the following should be specified: (1) The actions to be taken by the operator to process the input tuple, (2) the changes in the operator's state (if any) due to the processed tuple, and (3) the output of the operator.

The incremental maintenance of continuous predicate-window queries is different from the traditional incremental

query maintenance. The incremental evaluation of continuous queries in traditional databases has been addressed in Tapestry [8] and the maintenance of materialized views [4]. Tapestry addresses *append-only* queries in which an output tuple will remain in the query answer forever. Unlike Tapestry, the output tuple of a predicate-window query may be updated or deleted. On the other hand, materialized views deal with data resident on disk and the query answer is materialized. In materialized views, changes to the base tables are reflected into the materialized view via incremental maintenance algorithms [4]. Unlike materialized views, both the input to and output of the predicate-window query is a stream of tuples.

Long-living tuples: Unlike sliding-windows, a tuple entering the predicate-window may remain in the window for long periods of time. We call the tuples that do not expire from the predicate-window as "*long-living-tuples*". The number of tuples inside a predicate-window can grow unboundedly due to long-living tuples. Limiting the number of tuples inside a predicate-window is an interesting area of research.

5.2 Predicate Selectivity

For the window predicate, two different selectivities can be distinguished: positive selectivity and negative selectivity. The positive selectivity is defined the same as the traditional selectivity definition. Positive and update tuples will contribute to the positive selectivity of the window predicate. Negative tuples emitted from the window predicate will contribute to the negative selectivity. The negative selectivity can be defined as the selectivity of the OUT predicate in the predicate-window query. The OUT predicate can be implicit as the complement of the window predicate or explicit as in the IN/OUT predicate-window query model.

Positive and negative selectivities are illustrated by Figure 4. Given query Q_1 as in Example 1, the OUT predicate in this query is (implicitly) the complement of the window predicate "temperature greater than 90". Figure 4 gives the input of two different sensors to the PWINDOW operator. The circles in the figure represent the input to the PWINDOW operator. The *white* circles represent positive or update output tuples, the *black* circles represent negative output tuples, and the *gray* circles represent filtered out inputs. The two PWINDOW operators have the same number of input tuples (11 tuples) and the same number of positive/update tuples (5 tuples) but a different number of negative tuples (black circles). The negative selectivity of the query depends on the update pattern of the input tuples. Estimating the selectivity of the window predicate is critical for query optimization. Estimating the positive and negative selectivities of the window predicate is another interesting area for future research.

5.3 Shared Execution of Predicate-window Queries

Applications over data streams always involve a large number of concurrent continuous queries over the same data. Queries must be handled collectively by exploiting similarities and sharing resources such as computation, memory, and disk bandwidth among the queries. The PWINDOW operator is a new operator introduced by the predicate-window query model. Sharing the PWINDOW operator among several predicate-window queries can greatly improve the performance of the query processing engine. Efficient techniques for sharing the PWINDOW is an interesting area for future research.

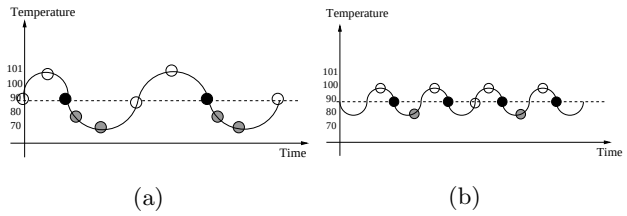


Figure 4: Object Update Pattern

6. CONCLUSION

In this paper, we proposed the predicate-window query model as a general model for window queries over data streams. Examples are discussed to illustrate how the existing sliding-window query approaches fail to answer some of the predicate-window queries. Moreover, the predicate-window query model can emulate the behavior of the sliding-window query model. We discussed several challenges and open research issues that need to be thought of for efficient realization of the predicate-window query model inside a data stream management system.

7. ACKNOWLEDGMENT

This work was supported in part by the National Science Foundation under Grants IIS-0093116, IIS-0209120, and 0010044-CCR.

8. REFERENCES

[1] A. Arasu, S. Babu, and J. Widom. The CQL Continuous Query Language: Semantic Foundations and Query Execution. Technical report, Stanford University, October 2003.

[2] T. M. Ghanem, M. A. Hammad, M. F. Mokbel, W. G. Aref, and A. K. Elmagarmid. Query Processing using Negative Tuples in Stream Query Engines. Technical Report 04-040, Purdue University, April 2005.

[3] L. Golab and M. T. Ozsu. Processing Sliding Window multi-joins in Continuous queries over Data Streams. In *VLDB*, 2003.

[4] A. Gupta and I. S. Mumick. Maintenance of Materialized Views: Problems, Techniques, and Applications. *IEEE Data Eng. Bull.*, 18(2):3–18, 1995.

[5] M. A. Hammad, W. G. Aref, and A. K. Elmagarmid. Stream Window Join: Tracking Moving Objects in Sensor-Network Databases. In *SSDBM*, 2003.

[6] J. Kang, J. F. Naughton, and S. D. Viglas. Evaluating Window Joins over Unbounded Streams. In *ICDE*, 2003.

[7] J. Li, D. Maier, K. Tufte, V. Papadimos, and P. A. Tucker. Semantics and Evaluation Techniques for Window Aggregates in Data Streams. In *SIGMOD*, 2005.

[8] D. B. Terry, D. Goldberg, D. Nichols, and B. M. Oki. Continuous Queries over Append-Only Databases. In *SIGMOD*, pages 321–330, 1992.

[9] P. A. Tucker, D. Maier, T. Sheard, and L. Fegaras. Exploiting Punctuation Semantics in Continuous Data Streams. *TKDE*, 15(3):555–568, 2003.

Micro-views, or on How to Protect Privacy while Enhancing Data Usability - Concepts and Challenges

Ji-Won Byun
byunj@cs.purdue.edu

Elisa Bertino
bertino@cerias.purdue.edu

CERIAS and Department of Computer Science
Purdue University
656 Oval Drive, West Lafayette, IN 47907

ABSTRACT

The large availability of repositories storing various types of information about individuals has raised serious privacy concerns over the past decade. Nonetheless, database technology is far from providing adequate solutions to this problem that requires a delicate balance between an individual's privacy and convenience and data usability by enterprises and organizations - a database which is rigid and over-protective may render data of little value. Though these goals may seem odd, we argue that the development of solutions able to reconcile them will be an important challenge to be addressed in the next few years. We believe that the next wave of database technology will be represented by a DBMS that provides high-assurance privacy and security. In this paper, we elaborate on such challenges. In particular, we argue that we need to provide different views of data at a very fine level of granularity; conventional view technology is able to select only up to a single attribute value for a single tuple. We need to go even beyond this level. That is, we need a mechanism by which even a single value inside a tuple's attribute may have different views; we refer them as micro-views. We believe that such a mechanism can be an important building block, together with other mechanisms and tools, of the next wave of database technology.

1. INTRODUCTION

Current information technology enables many organizations to collect, store and use a vast amount of personal information in their databases. The use of innovative knowledge extraction techniques combined with advanced data integration and correlation techniques [7, 8, 16] makes it possible to automatically extract a large body of information from the available databases and from a large variety of information repositories available on the web. Such a wealth of information and extracted knowledge raises, however, serious concerns about the privacy of individuals. As privacy awareness increases, individuals are becoming more reluctant to carry out their businesses and transactions online, and many enterprises are losing a considerable amount of potential profits [12]. Also, enterprises that collect information about individuals are in effect obligated to keep the collected information private and must strictly control the use of such information. Thus, information stored in the databases of an enterprise is not only a valuable property of the enterprise, but also a costly responsibility. Consequently, data management techniques providing high-assurance privacy and at the same time avoiding unnecessary restrictions to data access are in great demand. Equipped with such techniques, organizations shall be able to utilize information analysis and knowledge extraction to provide better and tailored services to individuals without violating individual privacy.

To date, issues related to privacy have been widely investigated and several privacy protecting techniques have been developed. To our best knowledge, the most well known effort is the W3C's Platform for Privacy Preference (P3P) [20]. P3P allows websites to express their privacy policy in a machine readable format so that using a software agent, consumers can easily compare the published privacy policies against their privacy preferences. P3P, however, does not provide any functionality to keep these promises in the internal privacy practice of enterprises. To complement P3P's lack of enforcement mechanisms, many privacy-aware access control models have also been investigated [3, 4, 9, 10]. Although all these models do protect privacy of data providers[1], they are very rigid and do not provide ways to maximize the utilization of private information. Specifically, in those models access decision is always binary; i.e., a data access is either allowed or denied as in most conventional access control models.

We believe that a new generation of privacy-aware access control models should maximize information usability by exploiting the nature of information privacy. First, information privacy is context-specific. For instance, consider address data of consumers. The tolerance level of individuals for their address being used for direct marketing could be significantly different from the address being used for consumer analysis. Furthermore, the tolerance level varies from individual to individual. Some consumers may feel that it is acceptable to disclose their purchase history or browsing habits in return for better services; others may feel that revealing such information violates their privacy. These differences in individuals suggest that access control models should be able to maximize the utilization of private information by taking such large variations into account.

Second, the use of data generalization[2] can significantly increase the comfort level of data providers. For example, suppose that an enterprise has collected the birth dates of its consumers. Such information is very personal, and many individuals may not be comfortable with their information being used. Suppose now that the enterprise promises its consumers that this information will be used only in a generalized form; e.g., $\langle 07/23/1970 \rangle$ will be generalized to a less specific value $\langle 07/1970 \rangle$ or a categorical value $\langle 1965 - 1975 \rangle$. This assurance will surely comfort many consumers. Clearly,

[1]By data providers, we refer to the subjects to whom the stored data is related.

[2]Data generalization refers to techniques that "replace a value with a less specific but semantically consistent value" [17].

Term	Description	Example
Privacy level	Level of privacy required by data provider	Low, Medium, High
Data type	Types of data being collected	Name, Address, Income, Age
Data usage type	Types of potential data usage (i.e. purpose)	Marketing, Admin, Shipping

Table 1: Privacy level, data type and data usage type

privacy enhancing access control models should be able to utilize more information by employing data generalization techniques.

The development of a DBMS that addresses the above requirements is a challenging task which requires revisiting theoretical foundations of data models as well as languages and architectures. A core DBMS component which is crucial in such a context is the access control system. Current access control systems are fundamentally inadequate with respect to the above goals. For example, fine-grained access control of data, an important requirement for privacy, poses several difficult problems and to date no satisfactory solution exists. We have yet to understand the relevant technical requirements.

In this paper, we pose as a new challenge the development of a new generation of access control systems. As an example, we propose a radically new access control model that is able to exploit the subtle nature of information privacy to maximize the usability of private information for enterprises with privacy guarantees. Our model is not to be considered a complete solution; rather it is meant to show some of capabilities that, in our opinion, a suitable model should provide. In particular, our model is based on the notion of a *micro-view*. A micro-view applies the well known idea of views at the level of the atomic components of tuples, that is, to an attribute value. By using the different precisions, one is able to finely calibrate the amount of information released by queries.

The remainder of this paper is organized as follows. In Section 2, we present a high-level description of our access control model. We discuss the technical challenges raised by our model in Section 3 and provide a brief survey of related work in Section 4. We then conclude our discussion in Section 5.

2. A SKETCH OF OUR "NAIVE" MODEL

Our model is based on a typical life-cycle of data concerning individuals. During the data collection phase, a data provider submits her privacy requirement, which specifies permissible usages of each data item and a level of privacy for each usage. This requirement is then stored in the database along with the collected data, and access to the data is strictly governed by the data provider's requirement. In this section, we first illustrate how data collection is carried out in our model and discuss the access control model in detail.

2.1 Data collection and preprocess

As a prerequisite to our approach, supported privacy levels, types of data and possible data usages (i.e., purposes) have to be clearly defined and clarified through the published privacy policy. Table 1 describes these concepts with some practical examples.

For simplicity of discussion, we consider the following privacy levels only: *Low*, *Medium* and *High*. We also consider only three data types, *name*, *address* and *income*, and two usage types, *admin* and *marketing*.

As previously mentioned, when individuals release their personal information, they specify permissible usages of each of their data items and a level of privacy for each usage. For instance, a data provider may select *Low* on *Address* for *Admin*; that is, she does not have any privacy concern over the address information when it is used for the purpose of administration. Thus, the address information can be used for the administrative purpose without any modification. However, the data provider may select *High* on *Address* for *marketing*. This indicates that she has great concerns about privacy of the address information when it is used for the purpose of marketing; thus, the address information should be used only in a sufficiently generalized form for the marketing purpose. Note that if one wishes to allow data providers to completely opt out from any use of data, another privacy level (e.g., Opt-out) can be added to indicate that the particular data should not be used at all.

Additional to storing the specified privacy requirements, the actual data items are preprocessed before being stored in the following way. Each data item is generalized and stored according to a multilevel organization, where each level corresponds to a specific privacy level. Intuitively, data for a higher privacy level requires a higher degree of generalization. For instance, the address data is stored into three levels: entire address for *Low*, city and state for *Medium* and state for *High*.

Table 2 illustrates some fictional records and privacy requirements stored in a conceptual database relation. Notice that every data item is stored in three different generalization levels, each of which corresponds to a particular privacy level. *PL_Admin* and *PL_Marketing* are metadata columns[3] storing the set of privacy levels of data for *Admin* and *Marketing*, respectively. For instance, {L, L, M} in *PL_marketing* indicates that for the marketing purpose the privacy levels of *Name* and *Address* are both *Low* while the privacy level of *Income* is *Medium*.

Note that exactly how such data is organized and stored in the database is a crucial issue as it determines the performance and storage efficiency. However, this issue is beyond the scope of this paper and shall be discussed in our future work.

2.2 Access Control

In our model, data users query the database using standard SQL statements. However, the data accessible to each query varies depending on the privacy levels of the data and the purpose of the query[4]. That is, each query runs as if it is running on a view that is defined by the purpose of the query and the privacy levels of data. We call such views *micro-views*. Tables 3 and 4 illustrate this effect. For in-

[3]The metadata columns may be viewable to any user, but they can be modified only by authorized users.

[4]In this paper, we assume that each query is associated with a specific purpose.

CustID	Name		Address		Income		PL_Admin	PL_Marketing
1001	*L*	Alice Park	*L*	123 First St., Seattle, WA	*L*	45,000	{L, M, H}	{H, H, H}
	M	Alice P.	*M*	Seattle, WA	*M*	40K-60K		
	H	A.P.	*H*	WA	*H*	Under 100K		
1002	*L*	Aaron Parker	*L*	491 3rd St, Lafayette, IN	*L*	121,000	{L, L, M}	{H, M, H}
	M	Aaron P.	*M*	Lafayette, IN	*M*	120K-140K		
	H	A.P.	*H*	IN	*H*	Over 100K		
1003	*L*	Carol Jones	*L*	35 Oval Dr, Chicago, IL	*L*	64,000	{L, L, L}	{L, M, H}
	M	Carol J.	*M*	Chicago, IL	*M*	60K-80K		
	H	C.J.	*H*	IL	*H*	Under 100K		

Table 2: Private information and metadata

CustID	Name	Address	Income
1001	Alice Park	Seattle	Under 100K
1002	Aaron Parker	491 3rd St, Lafayette, IN	120K-140K
1003	Carol Jones	35 Oval Dr, Chicago, IL	64,000

Table 3: Micro-view for *Admin* purpose

CustID	Name	Address	Income
1001	A. P.	WA	Under 100K
1002	A. P.	Lafayette, IN	Over 100K
1003	Carol Jones	Chicago, IL	Under 100K

Table 4: Micro-view for *Marketing* purpose

stance, any query against the base table in Table 2 with *Admin* purpose returns a result that is equivalent to the result of the query run on the micro-view in Table 3. As the micro-views directly reflect the information that is allowed by each data provider, querying against these views does not violate privacy.

Note that the major difference of our model from conventional database models is that in our model, different sets of data may be returned for the same query, depending on the privacy levels of data and the purpose of the query. For instance, suppose that the following query is written against the base table in Table 2: "**SELECT** * **FROM** Customer **WHERE** CustID = 1002". If the purpose of this query is *Admin*, then the system will return a tuple ⟨'Aaron Parker', '491 3rd St, Lafayette, IN', '120K-140K'⟩ as Aaron's privacy levels for *Admin* are specified as {L, L, M}. On the other hand, if the purpose of the query is *Marketing*, then a tuple ⟨'A. P.', 'Lafayette, IN', 'Over 100K'⟩ will be retrieved as his privacy levels for *Marketing* is {H, M, H}.

An important issue to be addressed is how we associate a purpose with each query. Note that it is not trivial for a system to correctly infer the purpose of a query as it means that the system must correctly deduce the actual intention of database users. However, if we assume that users are honest, then the problem of associating a purpose with each query becomes relatively easy; i.e., users themselves can specify the purpose of their queries with an additional clause. For instance, a simple select statement "**SELECT** name **FROM** customer" can be extended to a form of "**SELECT** name **FROM** customer **FOR** marketing". We believe that this is a reasonable approach. Many privacy violations occur from accidentally accessing unauthorized information, and thus it is important to develop a mechanism that database users can use to protect themselves from committing such accidental violations. A more sophisticated approach which validates whether users are indeed permitted to use their claimed purposes is thoroughly investigated in [5].

3. CHALLENGES

The comprehensive development of the approach we have sketched in the previous section and its integration in a DBMS architecture requires addressing several interesting challenges.

Policy specification language. The core of our model is that data providers can specify their privacy requirements using a privacy level for each data category. There is thus a strong need for a language in which privacy specifications can be expressed precisely. A challenge is that the language must be powerful enough to express every possible requirement, yet simple enough to avoid any ambiguity or conflict. Thus usability is a crucial issue. Especially as we cannot assume that every data provider would be an expert in privacy or any type of technology, GUI tools that are intuitive and instructive must be provided for them. We believe that many valuable lessons can be learned from existing technology related to P3P [20] and APPEL [19] and work on user interaction design (See [21] for example). It is important that data providers have a clear understanding of the guarantees provided by each privacy level.

Data generalization. Needless to say, devising a quality data generalization technique is one of the key challenges. There are two important issues to be addressed here. The first issue is that the generalization process must preserve meaningful information from actual data as inadequate information would not be of any use. For example, although numeric or structured data may be easy to generalize into multiple levels that are meaningful, it is unclear how unstructured data (e.g., textual data) should be generalized into multiple levels. We need also to devise generalization policies and ontologies supporting systematic and consistent data generalization across the database. The other important issue is that the generalization process must produce a sufficient level of data privacy by effectively suppressing distinctive information in individual data. For instance, consider the names of individuals. There are certain names that are less frequent than the others, and inadequate generalization techniques would not hide the uniqueness of such names. Moreover, if the content of database may changes dynamically, hiding such uniqueness becomes much more challenging. Clearly, a key challenge in data generalization is to balance the trade-off between information preservation and information loss. Also, generalization must be efficient. In many cases, the system will have to perform data generalization "on the fly" while processing a query. In other cases, post-processing of queries is required because what has to be returned may depend on various factors such as the cardinality and statistics of the results or even the past accesses. Many valuable lessons can be learned from var-

ious generalization techniques that are available in statistical databases [1]. The main challenge here is that these techniques may have to be used dynamically in a variety of settings, ranging from data storage to query processing.

Metrics for data privacy and data usability. So far, we have claimed that both privacy and usability of data can be achieved when data is sufficiently generalized. However, a key question is: how can we determine whether or not a certain generalization strategy provides a sufficient level of privacy and usability? As one can generalize data in various ways and degrees, we need metrics that methodologically measure the privacy and usability of generalized data. Such metrics are necessary to devise generalization techniques that satisfy the requirements of both data providers and data users.

Metadata storage. In our "naive" model, we assumed that the collected data is generalized and stored into multiple privacy levels at the preprocessing stage. This approach is simple and effective, yet may require large storage space. For instance, suppose there are n privacy levels in a system. This means that the required storage space would be n times the size of the collected data. Another approach is to postpone the generalization process to the time of data access. This method does not require any additional storage and may help avoid unnecessary data generalization. As some data items are accessed much less frequently than the others, those rarely accessed data items do not have to be generalized unless they are accessed. However, the overall performance may significantly suffer. Another possible solution is to use both pre-generalization and post-generalization selectively. For example, only data items that are expected to be accessed frequently are pre-generalized and stored. Other data items that are not pre-processed should be generalized when they are accessed. Also, for better performance the post-generalized data may be cached in a special space. Using this approach, one can try to reduce the overall cost of generalization process. However, a challenge here is to balance the trade-off between storage and performance. Yet another approach could be based on the use of views, which would have to be extended with innovative capabilities for value generation and transformation.

Complex query processing. In this paper we have considered only simple queries; i.e., queries without join, subqueries or aggregations. A key question here is whether complex queries can be introduced in our model. Even though it seems that they can be correctly processed in the model, it is not clear whether the results of such queries would be still meaningful. For instance, how do we calculate the sum of several generalized values, and how do we interpret such results?

Applicability to general-purpose access control. Although we have limited our discussion to access control for privacy protection, we believe it is possible to extend our model to a general-purpose access control model. For instance, each user can be assigned a trust level[5], and the access control system can control, based on the user's trust level, degrees of precision on accessible information. This approach is very similar to multilevel secure database systems [6, 15, 13], where every piece of information is classified into a security level and every user is assigned a security clearance. However, the main difference is that our approach can provide a much finer level of control as the access control decision is based on the question of "how much information can be allowed for a certain user", rather than "is information allowed for a certain user or not". In other words, our model utilizes the elaborated version of cover story in multilevel secure databases. This type of finer grained access control can be extremely useful for internal access control within an organization as well as information sharing between organizations. Even though such an extension seems very promising at this point, further investigation is required to confirm this hypothesis.

Other issues. There are many other issues that require careful investigation, such as problems of polyinstantiation [11, 14], inference and integrity. Addressing such issues is also crucial for the development of comprehensive access control models for high-assurance privacy.

[5]The trust level is not chosen by users, but assigned to users by security officers.

4. RELATED WORK

To date, several approaches have been reported that deal with various aspects of the problem of high-assurance privacy systems. Here we briefly discuss the approaches that have provided some initial solutions that can certainly be generalized and integrated into comprehensive solutions to such problem.

The W3C's Platform for Privacy Preference (P3P) [20] allows web sites to encode their privacy practice, such as what information is collected, who can access the data for what purposes, and how long the data will be stored by the sites, in a machine-readable format. P3P enabled browsers can read this privacy policy automatically and compare it to the consumer's set of privacy preferences which are specified in a privacy preference language such as A P3P Preference Exchange Language (APPEL) [19], also designed by the W3C.

The concept of Hippocratic databases that incorporates privacy protection within relational database systems was introduced by Agrawal et al. [2]. The proposed architecture uses privacy metadata, which consist of privacy policies and privacy authorizations stored in two tables. A privacy policy defines for each attribute of a table the usage purpose(s), the external-recipients and a retention period, while a privacy authorization defines which purposes each user is authorized to use.

Byun et al. presented a comprehensive approach for privacy preserving access control based on the notion of purpose [4, 5]. In the model, purpose information associated with a given data element specifies the intended use of the data element, and the model allows multiple purposes to be associated with each data element. The granularity of data labeling is discussed in detail in [4], and a systematic approach to implement the notion of access purposes, using roles and role-attributes is presented in [5].

Previous work on multilevel secure relational databases [6, 13, 15] also provides many valuable insights for designing a fine-grained secure data model. In a multilevel relational database system, every piece of information is classified into a security level, and every user is assigned a security clear-

ance. Based on this access class, the system ensures that each user gains access to only the data for which he has proper clearance, according to the basic restrictions. These constraints ensure that there is no information flow from a lower security level to a higher security level and that subjects with different clearances see different versions of multilevel relations.

In order to prevent re-identification of anonymized data, Sweeney introduced the notion of k-anonymity [18]. K-anonymity requires that information about each individual in a data release be indistinguishable from at least k-1 other individuals with respect to a particular set of attributes. Sweeney also proposed a technique using generalization and suppression of data to achieve k-anonymity with minimal distortion [17].

5. CONCLUSIONS

In this paper, we discussed a new approach for access control that maximizes the usability of private information for enterprises while, at the same time, assuring privacy. We believe that one direction for next-generation DBMS technology is represented by DBMS with high-assurance security and privacy. The "naive" model we presented in this paper provides an example of access control for such a new DBMS. Based on this model, we discussed many challenges that need to be addressed. We conclude this paper by saying that ultimately suitable access control systems with high-privacy assurance will be built by integrating techniques such as view mechanisms, statistical databases, anonymization, privacy-preserving computation and data mining. The main challenge is how to integrate such techniques in a full-fledged DBMS ensuring good performance.

6. REFERENCES

[1] Nabil Adam and John Wortmann. Security-control methods for statistical databases: A comparative study. *ACM Computing Surveys (CSUR)*, 21, 1989.

[2] Rakesh Agrawal, Jerry Kiernan, Ramakrishman Srikant, and Yirong Xu. Hippocratic databases. In *The 28th International Conference on Very Large Databases (VLDB)*, 2002.

[3] Paul Ashley, Calvin S. Powers, and Matthias Schunter. Privacy promises, access control, and privacy management. In *Third International Symposium on Electronic Commerce*, 2002.

[4] Jiwon Byun, Elisa Bertino, and Ninghui Li. Purpose based access control for privacy protection in relational database systems. Technical Report 2004-52, Purdue University, 2004.

[5] Jiwon Byun, Elisa Bertino, and Ninghui Li. Purpose based access control of complex data for privacy protection. In *Symposium on Access Control Model And Technologies (SACMAT)*, 2005.

[6] Dorothy Denning, Teresa Lunt, Roger Schell, William Shockley, and Mark Heckman. The seaview security model. In *The IEEE Symposium on Research in Security and Privacy*, 1998.

[7] Xin Dong, Alon Halevy, Jayant Madhavan, and Ema Nemes. Reference reconciliation in complex information spaces. In *ACM International Conference on Management of Data (SIGMOD)*, 2005.

[8] I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 1969.

[9] IBM. *The Enterprise Privacy Authorization Language (EPAL)*. Available at www.zurich.ibm.com/security/enterprise-privacy/epal.

[10] Kristen LeFevre, Rakesh Agrawal, Vuk Ercegovac, Raghu Ramakrishnan, Yirong Xu, and David DeWitt. Disclosure in hippocratic databases. In *The 30th International Conference on Very Large Databases (VLDB)*, August 2004.

[11] Fausto Rabitti, Elisa Bertino, Won Kim, and Darrell Woelk. A model of authorization for next-generation database systems. In *ACM Transactions on Database Systems (TODS)*, March 1991.

[12] Forrester Research. Privacy concerns cost e-commerce $15 billion. Technical report, September 2001. Available at www.forrester.com.

[13] Ravi Sandhu and Fang Chen. The multilevel relational data model. In *ACM Transactions on Information and System Security*, 1998.

[14] Ravi Sandhu and Sushil Jajodia. Polyinstantiation integrity in multilevel relations. In *IEEE Symposium on Security and Privacy*, 1990.

[15] Ravi Sandhu and Sushil Jajodia. Toward a multilevel secure relational data model. In *ACM International Conference on Management of Data (SIGMOD)*, 1991.

[16] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *ACM International conference on Knowledge discovery and data mining (SIGKDD)*, 2002.

[17] Latanya Sweeney. Achieving k-anonymity privacy protection using generalization and suppression. In *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 2002.

[18] Latanya Sweeney. K-anonymity: A model for protecting privacy. In *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 2002.

[19] World Wide Web Consortium (W3C). *A P3P Preference Exchange Language 1.0 (APPEL 1.0)*. Available at www.w3.org/TR/P3P-preferences.

[20] World Wide Web Consortium (W3C). *Platform for Privacy Preferences (P3P)*. Available at www.w3.org/P3P.

[21] Kaping Yee. User interaction design for secure systems. In *The 4th International Conference on Information and Communications Security*, 2002.

Research Issues in Data Stream Association Rule Mining

Nan Jiang and Le Gruenwald
The University of Oklahoma, School of Computer Science, Norman, OK 73019, USA
Email: {nan_jiang, ggruenwald} @ou.edu

ABSTRACT

There exist emerging applications of data streams that require association rule mining, such as network traffic monitoring and web click streams analysis. Different from data in traditional static databases, data streams typically arrive continuously in high speed with huge amount and changing data distribution. This raises new issues that need to be considered when developing association rule mining techniques for stream data. This paper discusses those issues and how they are addressed in the existing literature.

1. INTRODUCTION

A data stream is an ordered sequence of items that arrives in timely order. Different from data in traditional static databases, data streams are continuous, unbounded, usually come with high speed and have a data distribution that often changes with time [Guha, 2001]. As the number of applications on mining data streams grows rapidly, there is an increasing need to perform association rule mining on stream data. An association rule is an implication of the form $X \Rightarrow Y$ (s, c), where X and Y are frequent itemsets in a transactional database and $X \cap Y = \varnothing$, s is the percentage of records that contain both X and Y in the database, called support of the rule, and c is the percentage of records containing X that also contain Y, called the confidence of the rule. Association rule mining is to find all association rules the support and confidence of which are above or equal to a user-specified minimum support and confidence, respectively.

One example application of data stream association rule mining is to estimate missing data in sensor networks [Halatchev, 2005]. Another example is to predict frequency estimation of Internet packet streams [Demaine, 2002]. In the MAIDS project [Cai, 2004], this technique is used to find alarming incidents from data streams. Association rule mining can also be applied to monitor manufacturing flows [Kargupta, 2004] to predict failure or generate reports based on web log streams, and so on.

Data streams can be further classified into offline streams and online streams. Offline streams are characterized by regular bulk arrivals [Manku, 2002]. Among the above examples, generating reports based on web log streams can be treated as mining offline data streams because most of reports are made based on log data in a certain period of time. Other offline stream examples include queries on updates to warehouses or backup devices. Queries on these streams are allowed to be processed offline.

Online streams are characterized by real-time updated data that come one by one in time. From the above examples, predicting frequency estimation of Internet packet streams is an application of mining online data streams because Internet packet streams is a real-time one packet by one packet process. Other online data streams are stock tickers, network measurements and sensor data. They have to be processed online and must keep up with the rapid speed of online queries. They have to be discarded right after arrived and being processed. In addition, unlike with offline data streams, bulk data processing is not possible for online stream data.

Due to the characteristics of stream data, there are some inherent challenges for stream data association rule mining. First, due to the continuous, unbounded, and high speed characteristics of data streams, there is a huge amount of data in both offline and online data streams, and thus, there is not enough time to rescan the whole database or perform a multi-scan as in traditional data mining algorithms whenever an update occurs. Furthermore, there is not enough space to store all the stream data for online processing. Therefore, a one scan of data and compact memory usage of the association rule mining technique are necessary. Second, the mining method of data streams needs to adapt to their changing data distribution; otherwise, it will cause the concept drifting problem [Wang, 2003], which we will discuss in Section 2.3.1. Third, due to the high speed characteristics of online data streams, they need to be processed as fast as possible; the speed of the mining algorithm should be faster than the data coming rate, otherwise data approximation techniques, such as sampling and load shedding, need to be applied which will decrease the accuracy of the mining results. Fourth, due to the continuous, high speed, and changing data distribution characteristics, the analysis results of data streams often keep changing as well. Therefore, mining of data streams should be an incremental process to keep up with the highly update rate, i.e. new iterations of mining results are built based on old mining results so that the results will not have to be recalculated each time a user's request is received. Fifth, owing to the unlimited amount of stream data and limited system resources, such as memory space and CPU power, a mining mechanism that adapts itself to available resources is needed; otherwise, the accuracy of the mining results will be decreased.

Traditional association rule mining algorithms are developed to work on static data and, thus, can not be applied directly to mine association rules in stream data. The first recognized frequent itemsets mining algorithm

for traditional databases is Apriori [Agrawal, 1993]. After that, many other algorithms based on the ideas of Apriori were developed for performance improvement [Agrawal, 1994, Han, 1999]. Apriori-based algorithms require multiple scans of the original database, which leads to high CPU and I/O costs. Therefore, they are not suitable for a data stream environment, in which data can be scanned only once. Another category of association rule mining algorithms for traditional databases proposed by Han and Pei [Han, 2000] are those using a frequent pattern tree (FP-tree) data structure and an FP-growth algorithm which allows mining of frequent itemsets without generating candidate itemsets. Compared with Apriori-based algorithms, it achieves higher performance by avoiding iterative candidate generations. However, it still can not be used to mine association rules in data streams since the construction of FP-tree requires two scans of data.

As more and more applications generate a large amount of data streams every day, such as web transactions, telephone records, and network flows, much research on how to get frequent items, patterns and association rules in a data stream environment has been conducted [Chang, 2003, Chang, 2004, Charikar, 2004, Chi, 2004, Cormode, 2003, Demaine, 2002, Giannella, 2003, Huang, 2002, Jin, 2003, Karp, 2003, Li, 2004, Lin, 2005, Manku, 2002, Relue, 2001, Yang, 2004, Yu, 2004]. However, these algorithms are focused on one or more application areas, and none of them fully addresses the issues that need to be solved in order to mine association rules in data streams.

In [Gaber, 2005], Gaber et al briefly discussed some general issues concerning stream data mining. They did not provide a thorough discussion for issues that need to be considered in the specific area of data stream association rule mining; they merely addressed the state of the art solutions. In this paper, we focus on research issues concerning association rule mining in data streams and, whenever possible, review how they are handled in the existing literature.

The rest of this paper is organized as follows. Section 2 discusses general issues that need to be considered for all data association rule mining algorithms for data streams. Section 3 describes application dependent issues. Section 4 summarizes the merits and lessons learned from the existing studies and concludes the paper.

2. GENERAL ISSUES IN DATA STREAM ASSOCIATION RULE MINING

The characteristics of data streams as pointed out in Section 1 indicate that when developing association rule mining techniques, there are more issues that need to be considered in data streams than in traditional databases. In this section, general issues are discussed. These issues are crucial and need to be taken into account in all applications when developing an association rule mining technique for stream data.

2.1. Data Processing Model

The first issue addresses which parts of data streams are selected to apply association rule mining. From the definition given in Section 1, data streams consist of an ordered sequence of items. Each set of items is usually called “transaction”. The issue of data processing model here is to find a way to extract transactions for association rule mining from the overall data streams. Because data streams come continuously and unboundedly, the extracted transactions are changing from time to time.

According to the research of Zhu and Shasha [Zhu, 2002], there are three stream data processing models, Landmark, Damped and Sliding Windows. The Landmark model mines all frequent itemsets over the entire history of stream data from a specific time point called landmark to the present. A lot of research has been done based on this model [Charikar, 2004, Cormode, 2003, Jin, 2003, Karp, 2003, Li, 2004, Manku, 2002, Yang, 2004, Yu, 2004]. However, this model is not suitable for applications where people are interested only in the most recent information of the data streams, such as in the stock monitoring systems, where current and real time information and results will be more meaningful to the end users.

The Damped model, also called the Time-Fading model, mines frequent itemsets in stream data in which each transaction has a weight and this weight decreases with age. Older transactions contribute less weight toward itemset frequencies. In [Chang, 2003] and [Giannella, 2003], they use exactly this model. This model considers different weights for new and old transactions. This is suitable for applications in which old data has an effect on the mining results, but the effect decreases as time goes on.

The Sliding Windows model finds and maintains frequent itemsets in sliding windows. Only part of the data streams within the sliding window are stored and processed at the time when the data flows in. In [Chang, 2004, Chi, 2004, Lin, 2005], the authors use this concept in their algorithms to get the frequent itemsets of data streams within the current sliding window. The size of the sliding window may be decided according to applications and system resources. The mining result of the sliding window method totally depends on recently generated transactions in the range of the window; all the transactions in the window need to be maintained in order to remove their effects on the current mining results when they are out of range of the sliding window.

All these three models have been used in current research on data streams mining. Choosing which kind of data process models to use largely depends on application needs. An algorithm based on the Landmark model can be converted to that using the Damped model by adding a decay function on the upcoming data streams. It can also be converted to that using Sliding Windows by keeping track of and processing data within a specified sliding window.

2.2. Memory Management

The next fundamental issue we need to consider is how to optimize the memory space consumed when running the mining algorithm. This includes how to decide the information we must collect from data streams and how to choose a compact in-memory data structure that allows the information to be stored, updated and retrieved efficiently. Fully addressing these issues in the mining algorithm can greatly improve its performance.

2.2.1. Information to Be Collected and Stored in Memory

Classical association rule mining algorithms on static data collect the count information for all itemsets and discard the non-frequent itemsets and their count information after multiple scans of the database. This would not be feasible when we mine association rules in stream data due to the two following reasons. First, there is not enough memory space to store all the itemsets and their counts when a huge amount of data comes continuously. Second, the counts of the itemsets are changing with time when new stream data arrives. Therefore, we need to collect and store the least information possible, but enough to generate association rules.

In [Karp, 2003], the most frequent items and their counts are stored in the main memory. This technique stores the most important information. However, because it discards infrequent items and their counts and discarded items may become frequent in the future, it cannot get the information associated with non-frequent items when later they become frequent. In [Yang, 2004], the available computer memory is used to keep frequency counts of all short itemsets (itemsets with $k \leq 3$, where k is the maximum size of frequent itemsets), thus the association rule mining for short itemsets in data streams becomes trivial. But as pointed out by the authors, this technique only suits limited applications where $k \leq 3$ and $n \leq 1800$ (n is the total number of data items). We can see that there is a trade off between the information we collect and the usage of system resources. The more information we collect to get more accurate results, the more memory space we use and the more processing time is needed.

2.2.2. Compact Data Structure

An efficient and compact data structure is needed to store, update and retrieve the collected information. This is due to bounded memory size and huge amounts of data streams coming continuously. Failure in developing such a data structure will largely decrease the efficiency of the mining algorithm because, even if we store the information in disks, the additional I/O operations will increase the processing time. The data structure needs to be incrementally maintained since it is not possible to rescan the entire input due to the huge amount of data and requirement of rapid online querying speed.

In [Manku, 2002], a lattice data structure is used to store itemsets, approximate frequencies of itemsets, and maximum possible errors in the approximate frequencies. In [Li, 2004], the authors employ a prefix tree data structure to store item ids and their support values, block ids, head and node links pointing to the root or a certain node. In [Giannella, 2003], a FP-tree is constructed to store items, support information and node links. A proper data structure is a crucial part of an efficient algorithm since it is directly associated with the way we handle newly arrived information and update old stored information. A small and compact data structure which is efficient in inserting, retrieving and updating information is most favorable when developing an algorithm to mine association rules for stream data.

2.3. One Pass Algorithm to Generate Association Rules

Another fundamental issue is to choose the right type of mining algorithms. Association rules can be found in two steps: 1) finding large itemsets (support is ≥ user specified support) for a given threshold support and 2) generate desired association rules for a given confidence. In the following subsections, we discuss the issues that need to be considered to generate and maintain frequent itemsets and association rules in data streams.

2.3.1. Frequent Itemsets

There exist a number of techniques for finding frequent itemsets in data streams. Based on the result sets produced, stream data mining algorithms can be categorized as exact algorithms or approximate algorithms.

In exact algorithms, the result sets consist of all of the itemsets the support values of which are greater than or equal to the threshold support. In [Karp, 2003] and [Yang, 2004], the authors use the exact algorithms to generate the result frequent itemsets. It is important for many applications to know the exact answers of the mining results; however, additional cost is needed to generate the accurate result set when the processing data is huge and continuous. The technique proposed in [Karp, 2003] takes two scans to generate the exact result set, and in [Yang, 2004], the algorithm generated can only mine short itemsets, which cannot be applied to large itemsets. Another option to get the exact mining results with relatively small memory usage is to store and maintain only special frequent itemsets, such as closed or maximal frequent itemsets, in memory. In [Chi, 2004] and [Mao, 2005], the authors proposed algorithms to maintain only closed frequent itemsets and maximal frequent itemsets over a sliding window and landmark processing model, respectively. In both of these cases, how we can get all the information to further generate association rules based on these special itemsets is an additional issue that needs to be considered.

Approximate algorithms generate approximate result sets with or without an error guarantee. Approximate mining frequent patterns with a probabilistic guarantee can take two possible approaches: false positive oriented and false negative oriented. The former includes some infrequent

patterns in the result sets, whereas the latter misses some frequent patterns [Yu, 2004].

Since data streams are rapid, time-varying streams of data elements, itemsets which are frequent are changing as well. Often these changes make the model built on old data inconsistent with the new data, and frequent updating of the model is necessary. This problem is known as concept drifting [Wang, 2003]. From the aspect of association rule mining, when data is changing over time, some frequent itemsets may become non-frequent and some non-frequent itemsets may become frequent. If we store only the counts of frequent itemsets in the data structure, when we need the counts for potential non-frequent itemsets which would become frequent itemsets later, we cannot get this information. Therefore, the technique to handle concept drifting needs to be considered. In [Chi, 2004], Chi et al proposed a method to reflect the concept drifts by boundary movements in the closed enumeration tree (CET).

From the above discussions, we can see that when designing a stream data association rule mining algorithm, we need to answer a number of questions: should we use an exact or approximate algorithm to perform association rule mining in data streams? Can its error be guaranteed if it is an approximate algorithm? How to reduce and guarantee the error? What is the tradeoff between accuracy and processing speed? Is data processed within one pass? Can this algorithm handle a large amount of data? Up to how many frequent itemsets can this algorithm mine? Can this algorithm handle concept drifting and how?

In the current works published in this area, [Karp, 2003] [Yang, 2004] [Chi, 2004] and [Mao, 2005] proposed exact algorithms, while [Li, 2004], [Yu, 2004], [Chang, 2004], [Manku, 2002], [Charikar, 2004] and [Giannella, 2003] proposed approximate algorithms. Among them [Yu, 2004] uses the false negative method to mine association rules, while the other approximate algorithms use the false positive method. [Chi, 2004] considered the concept drifting problem in its proposed algorithm.

2.3.2. Mechanism to Maintain and Update Association Rules

The next step after we get frequent itemsets is to generate and maintain desired association rules for a given confidence. As we can see from the previous discussions, mining association rules involves a lot of memory and CPU costs. This is especially a problem in data streams since the processing time is limited to one online scan. Therefore, when to update association rules, in real time or only at needs, is another fundamental issue.

The problem of maintaining discovered association rules was first addressed in [Cheung, 1996]. The authors proposed an incremental updating technique called FUP to update discovered association rules in a database when new transactions are added to the database. A more general algorithm, called FUP2, was proposed later in [Cheung, 1997] which can update the discovered association rules when new transactions are added to, delete from, or modified in the database. However in a data stream environment, stream data are added continuously, and therefore, if we update association rules too frequently, the cost of computation will increase drastically.

In [Lee, 1997], the authors proposed an algorithm, called DELI, which uses a sampling technique to estimate the difference between the old and new association rules. If the estimated difference is large enough, the algorithm signals the need of an update operation; otherwise, it takes the old rules as an approximation of the new rules. It considers the difference in association rules, but does not consider the performance of incremental data mining algorithms for evolving data, which is especially the situation in data stream mining. [Zheng, 2003] proposed a metric distance as a difference measure between sequential patterns and used a method, called TPD, to decide when to update the sequential patterns of stream data. The authors suggested that some initial experiments be done to discover a suitable incremental ratio and then this ratio be used to decide when would be better to update sequential patterns. The TPD method is only suitable for streams with little concept drifting, that is to say the change of data distribution is relatively small.

2.4. Resource Aware

Resources such as memory space, CPU, and sometimes energy, are very precious in a stream mining environment. They are very likely to be used up when processing data streams which arrive with rapid speed and a huge amount. What should we do when the resources are nearly consumed? If we totally ignore the resources available, for example the main memory, when processing the mining algorithm, data will be lost when the memory is used up. This would lead to the inaccuracy of the mining results, thus degrade the performance of the mining algorithm. Shall we just shed the incoming data or adjust our technique to handle this problem?

In [Gaber, 2003, Gaber, 2004, Teng, 2004], the authors discussed this issue and proposed their solutions for resource-aware mining. Gaber et al. proposed an approach, called AOG, which uses a control parameter to control its output rate according to memory, time constrains and data stream rate [Gaber, 2003, Gaber, 2004]. Teng et al. proposed an algorithm, called RAM-DS, to not only reduce the memory required for data storage but also retain good approximation of temporal patterns given limited resources like memory space and computation power [Teng, 2004].

3. APPLICATION DEPENDENT ISSUES

Different data stream application environments may have different needs for an association rule mining algorithm. In this section, we discuss issues that are application dependent.

3.1. Timeline Query

Stream data come continuously over time. In some applications, user may be interested in getting association rules based on the data available during a certain period of time. Then the storage structure needs to be dynamically adjusted to reflect the evolution of itemset frequencies over time. How to efficiently store the stream data with timeline and how to efficiently retrieve them during a certain time interval in response to user queries is another important issue.

In [Giannella, 2003], the authors proposed a method to incrementally maintain tilted-time windows for each pattern at multiple time granularities, which is convenient for applications where users are more interested in getting detailed information from the recent time period. In [Lin, 2005] a time-sensitive sliding window model is created to mine and maintain the frequent itemsets during a user defined time interval.

3.2. Multidimensional Stream Data

In applications where stream data are multi-dimensional in nature, multi-dimensional processing techniques for association rule mining need to be considered. Take a sensor data network as an example and assume that it gets and distributes the weather information. It is possible that when the temperature for one sensor S goes up, its humidity will decrease and the temperature from the sensors in close vicinity and toward the same wind direction of the sensor S will also increase. Here, temperature and humidity are the multidimensional information of the sensor. How to efficiently store, update and retrieve the multidimensional information to mine association rules in multidimensional data streams is an issue we need to consider in this situation.

[Pinto, 2001] proposed a method to integrate multidimensional analysis and sequential data mining, and [Yu, 2005] proposed an algorithm to find sequential patterns from d-dimensional sequence data, where $d > 2$.

3.3. Online Interactive Processing

In some applications, users may need to modify the mining parameters during the processing period, especially when processing data streams because there is not a specific stop point during the mining process. Therefore, how to make the online processing interactive according to user inputs before and during the processing period is another important issue.

In [Parthasarathy, 1999], the authors presented techniques for maintaining frequent sequences upon database updates and user interaction and without re-executing the algorithm on the entire dataset. In [Veloso, 2003], the interactive approach makes use of selective updates to avoid updating the entire model of frequent itemsets. Ghoting and Parthasarathy proposed a scheme in [Ghoting, 2004] which gives controlled interactive response times when processing distributed data streams.

3.4. Distributed Environment

In a distributed environment, stream data comes from multiple remote sources. Such an environment imposes excessive communication overhead and wastes computational resources when data is dynamic. In this situation, how to minimize the communication cost, how to combine frequency counts from multiple nodes, and how to mine data streams in parallel and update the associated information incrementally are additional issues we need to consider.

Otey discussed this problem and presented an approach making use of parallel and incremental techniques to generate frequent itemsets of both local and global sites in [Otey, 2003] and [Otey, 2004]. In [Veloso, 2003b], the authors proposed a distributed algorithm which imposes low communization overhead for mining distributed datasets. Schuster et al presented a distributed association rule mining algorithm called D-ARM to perform a single scan over the database [Schuster, 2003]. The scheme proposed in [Ghoting, 2004] gives controlled interactive response times when processing distributed data streams. Wolff and Schuster proposed an algorithm to mine association rules in large-scale distributed peer-to-peer systems [Wolff, 2004], by which every node in the system can reach the exact solution.

3.5. Visualization

In some data stream applications, especially monitoring applications, there is a demand for visualization of association rules to facilitate the analysis process. An interactive use of visualized graphs can help the users understand the relationship between related association rules better so that they can further select and explore a specific set of rules from the visualization.

In [Hofmann, 2000], the authors showed how Mosaic plots can be used to visualize association rules. In [Bruzzese, 2004], Bruzzese and Buono proposed a visual strategy to both overview the association rule structure and further investigate inside a specific set of rules selected by the user. In [Cai, 2004], the authors developed a set of visualization tools which can be served for continuous queries and mining displays; they trigger alarms and give messages when some alarming incidents are being detected based on the ongoing stream data.

4. CONCLUSIONS

In this paper we discussed the issues that need to be considered when designing a stream data association rule mining technique. We reviewed how these issues are handled in the existing literature. We also discussed issues that are application-dependent.

From the above discussions, we can see that most of the current mining approaches adopt an incremental and one pass mining algorithm which is suitable to mine data streams, but few of them address the concept drifting problem. Most of these algorithms produce approximate results [Li, 2004, Yu, 2004, Chang, 2004, Manku, 2002,

Charikar, 2004, Giannella, 2003, Lin, 2005]. This is because due to the huge amount of data streams and limited memory, there is not enough space to keep frequency counts of all itemsets in the whole data streams as we do in traditional databases. A few of the proposed algorithms generate exact mining results by maintaining a small subset of frequent itemsets from data streams and keeping their exact frequency counts [Yang, 2004, Chi, 2004, Mao, 2005]. To keep track of the exact frequency counts of target itemsets with limited memory space, one way is to adopt the sliding window data processing model, which maintains only part of the frequent itemsets in sliding window(s) as in [Chi, 2004]. Another way is to maintain only special itemsets such as short frequent itemsets, closed frequent itemsets or maximal frequent itemsets as in [Yang, 2004, Mao, 2005].

The current stream data mining methods require users to define one or more parameters before their execution; however, most of them do not mention how users can adjust these parameters online while they are running. It is not desirable/feasible for users to wait until a mining algorithm to stop before they can reset the parameters. This is because it may take a long time for the algorithm to finish due to the continuous arrival and huge amount of data streams. Some proposed methods let users adjust only certain parameters online, but these parameters may not be the key ones to the mining algorithms, and thus are not enough for a user friendly mining environment. For example, in [Ghoting, 2004], the authors proposed a method to mine distributed data streams which allows the users, to modify online only one of the mining parameters, the response time, to trade off between the query response time and accuracy of the mining results. For further improvement, we may consider to either let users adjust online or let the mining algorithm auto-adjust most of the key parameters in association rule mining, such as support, confidence and error rate.

Research in data stream association rule mining is still in its early stage. To fully address the issues discussed in this paper would accelerate the process of developing association rule mining applications in data stream systems. As more of these problems are solved and more efficient and user-friendly mining techniques are developed for the end users, it is quite likely that in the near future data stream association rule mining will play a key role in the business world.

Acknowledgement

This material is based upon work supported by (while serving at) the National Science Foundation (NSF), the NASA Grant No. NNG05GA30G issued through the Office of Space Science and the OSU Grant. Any opinion, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

5. REFERENCES

[Agrawal, 1993] Rakesh Agrawal, Tomasz Imielinski, Arun Swami; Mining Association Rules between Sets of Items in Massive Databases; Int'l Conf. on Management of Data; May 1993.

[Agrawal, 1994] Rakesh Agrawal, Ramakrishnan Srikant; Fast Algorithms for Mining Association Rules; Int'l Conf. on Very Large Databases; September 1994.

[Bruzzese, 2004] Dario Bruzzese, Paolo Buono; Combining Visual Techniques for Association Rules Exploration; The Working Conf. on Advanced Visual Interfaces; May 2004.

[Cai, 2004] Y. Dora Cai, Greg Pape, Jiawei Han, Michael Welge, Loretta Auvil; MAIDS: Mining Alarming Incidents from Data Streams; Int'l Conf. on Management of Data; June 2004.

[Chang, 2003] Joong Hyuk Chang, Won Suk Lee, Aoying Zhou; Finding Recent Frequent Itemsets Adaptively over Online Data Streams; ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining; August 2003.

[Chang, 2004] Joong Hyuk Chang, Won Suk Lee; A Sliding Window Method for Finding Recently Frequent Itemsets over Online Data Streams; Journal of Information Science and Engineering; July 2004.

[Charikar, 2004] Moses Charikar, Kevin Chen, Martin Farach-Colton; Finding Frequent Items in Data Streams; Theoretical Computer Science; January 2004.

[Cheung, 1996] David W. Cheung, Jiawei Han, Vincent T. Ng, C.Y. Wong; Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique; IEEE Int'l Conf. on Data Mining; November 1996.

[Cheung, 1997] David W. Cheung, S.D. Lee, Benjamin Kao; A General Incremental Technique for Maintaining Discovered Association Rules; Int'l Conf. on Database Systems for Advanced Applications; 1997.

[Chi, 2004] Yun Chi, Haixun Wang , Philip S. Yu , Richard R.; Moment: Maintaining Closed Frequent Itemsets over a Stream Sliding Window; IEEE Int'l Conf. on Data Mining; November 2004.

[Cormode, 2003] Graham Cormode, S.Muthukrishnan; What's Hot and What's Not: Tracking Most Frequent Items Dynamically; ACM Transactions on Database Systems; March 2005.

[Demaine, 2002] Erik D. Demaine, Alejandro Lpez-Ortiz, J. Ian Munro; Frequency Estimation of Internet Packet Streams with Limited Space; European Symposium on Algorithms; September 2002.

[Gaber, 2003] Mohamed Medhat Gaber, Shonali Krishnaswamy, Arkady Zaslavsky; Adaptive Mining Techniques for Data Streams Using Algorithm Output Granularity; The Australasian Data Mining Workshop; December 2003.

[Gaber, 2004] Mohamed Medhat Gaber, Arkady Zaslavsky and Shonali Krishnaswamy; Resource-Aware Knowledge Discovery in Data Streams; Int'l Workshop on Knowledge Discovery in Data Streams; September 2004.

[Gaber, 2005] Mohamed Medhat Gaber, Arkady Zaslavsky, Shonali Krishnaswamy; Mining Data Streams: A Review; ACM SIGMOD Record Vol. 34, No. 2; June 2005.

[Ghoting, 2004] Amol Ghoting, Srinivasan Parthasarathy; Facilitating Interactive Distributed Data Stream Processing and Mining; IEEE Int'l Symposium on Parallel and Distributed Processing Systems; April 2004.

[Giannella, 2003] Chris Giannella, Jiawei Han, Jian Pei, Xifeng Yan, Philip S. Yu; Mining Frequent Patterns in Data Streams at Multiple Time Granularities; Data Mining: Next Generation Challenges and Future Directions, AAAI/MIT; 2003.

[Guha, 2001] Sudipto Guha, Nick Koudas, Kyuseok Shim; Data Streams and Histograms; ACM Symposium on Theory of Computing; 2001.

[Han, 1999] Jiawei Han, Guozhu Dong, Yiwen Yin; Efficient mining of partial periodic patterns in time series database; IEEE Int'l Conf. on Data Mining; March 1999.

[Han, 2000] Jiawei Han, Jian Pei, Yiwen Yin; Mining Frequent Patterns without Candidate Generation; Int'l Conf. on Management of Data; May 2000.

[Halatchev, 2005] Mihail Halatchev and Le Gruenwald; Estimating Missing Values in Related Sensor Data Streams; Int'l Conf. on Management of Data; January 2005.

[Hofmann, 2000] Heike Hofmann, Arno P. J. M. Siebes, Adalbert F. X. Wilhelm; Visualizing Association Rules with Interactive Mosaic Plots; ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining; August 2000.

[Huang, 2002] Hao Huang, Xindong Wu, Richard Relue; Association Analysis with One Scan of Databases; IEEE Int'l Conf. on Data Mining; December 2002.

[Jin, 2003] Cheqing Jin, Weining Qian, Chaofeng Sha, Jeffrey X. Yu, Aoying Zhou; Dynamically Maintaining Frequent Items over a Data Stream; Int'l Conf. on Information and Knowledge Management; 2003.

[Kargupta, 2004] Hillol Kargupta, Ruchita Bhargava, Kun Liu, Michael Powers, Patrick Blair, Samuel Bushra, James Dull, Kakali Sarkar, Martin Klein, Mitesh Vasa, David Handy; VEDAS: A Mobile and Distributed Data Stream Mining System for Real-Time Vehicle Monitoring; SIAM Int'l Conf. on Data Mining; 2004.

[Karp, 2003] Richard M. Karp, Scott Shenker; A Simple Algorithm for Finding Frequent Elements in Streams and Bags; ACM Transactions on Database Systems; March 2003.

[Lee, 1997] S.D. Lee, David W. Cheung; Maintenance of Discovered Association Rules: When to update?; Research Issues on Data Mining and Knowledge Discovery; 1997.

[Li, 2004] Hua-Fu Li, Suh-Yin Lee, and Man-Kwan Shan; An Efficient Algorithm for Mining Frequent Itemsets over the Entire History of Data Streams; Int'l Workshop on Knowledge Discovery in Data Streams; Sept. 2004.

[Lin, 2005] Chih-Hsiang Lin, Ding-Ying Chiu, Yi-Hung Wu, Arbee L. P. Chen; Mining Frequent Itemsets from Data Streams with a Time-Sensitive Sliding Window; SIAM Int'l Conf. on Data Mining; April 2005.

[Manku, 2002] Gurmeet Singh Manku, Rajeev Motwani; Approximate Frequency Counts over Data Streams; Int'l Conf. on Very Large Databases; 2002.

[Mao, 2005] Guojun Mao, Xindong Wu, Chunnian Liu, Xingquan Zhu, Gong Chen, Yue Sun, Xu Liu; Online Mining of Maximal Frequent Itemsequences from Data Streams; University of Vermont, Computer Science Technical Report CS-05-07; June 2005.

[Otey, 2003] Matthew Eric Otey, Chao Wang, Srinivasan Parthasarathy, Adriano Veloso, Wagner Meira Jr.; Mining Frequent Itemsets in Distributed and Dynamic Databases; IEEE Int'l Conf. on Data Mining; 2003.

[Otey, 2004] Matthew Eric Otey, Srinivasan Parthasarathy, Chao Wang, Adriano Veloso, Wagner Meira Jr.; Parallel and Distributed Methods for Incremental Frequent Itemset Mining; IEEE Transactions on Systems, Man and Cybernetics; December 2004.

[Parthasarathy, 1999] S. Parthasarathy, M. J. Zaki, M. Ogihara, S. Dwarkadas; Incremental and interactive sequence mining; Int'l Conf. on Information and Knowledge Management; 1999.

[Pinto, 2001] Helen Pinto, Jiawei Han, Jian Pei, Ke Wang, Qiming Chen, Umeshwar Dayal; Multi-Dimensional Sequential Pattern Mining; Int'l Conf. on Information and Knowledge Management; 2001.

[Relue, 2001] Richard Relue, Xindong Wu, Hao Huang; Efficient runtime generation of association rules; Int'l Conf. on Information and Knowledge Management; October 2001.

[Schuster, 2003] Assaf Schuster, Ran Wolff, and Dan Trock; Distributed Algorithm for Mining Association Rules; IEEE Int'l Conf. on Data Mining; November 2003.

[Teng, 2004] Wei-Guang Teng, Ming-Syan Chen, and Philip S. Yu; Resource-Aware Mining with Variable Granularities in Data Streams; SIAM Int'l Conf. on Data Mining; 2004.

[Veloso, 2003] Adriano Veloso, Wagner Meira Jr., Marcio Carvalho, Srini Parthasarathy, Mohammed J. Zaki; Parallel, Incremental and Interactive Mining for Frequent Itemsets in Evolving Databases; Int'l Workshop on High Performance Data Mining: Pervasive and Data Stream Mining; May 2003.

[Veloso, 2003b] Adriano Veloso, Matthew Eric Otey, Srinivasan Parthasarathy, Wagner Meira Jr.; Parallel and Distributed Frequent Itemset Mining on Dynamic Datasets; Int'l Conf. on High Performance Computing; 2003.

[Wang, 2003] Haixun Wang, Wei Fan, Philip S. Yu, Jiawei Han; Mining Concept-Drifting Data Streams using Ensemble Classifiers; ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining; August 2003.

[Wolff, 2004] Ran Wolff, Assaf Schuster; Association Rule Mining in Peer-to-Peer Systems; IEEE Transactions on Systems, Man and Cybernetics, Part B, Vol. 34, Issue 6; December 2004.

[Yang, 2004] Li Yang, Mustafa Sanver; Mining Short Association Rules with One Database Scan; Int'l Conf. on Information and Knowledge Engineering; June 2004.

[Yu, 2004] Jeffrey Xu Yu, Zhihong Chong, Hongjun Lu, Aoying Zhou; False Positive or False Negative: Mining Frequent Itemsets from High Speed Transactional Data Streams; Int'l Conf. on Very Large Databases; 2004.

[Yu, 2005] Chung-Ching Yu, Yen-Liang Chen; Mining Sequential Patterns from Multidimensional Sequence Data; IEEE Transactions on Knowledge and Data Engineering; January 2005.

[Zheng, 2003] Qingguo Zheng, Ke Xu, Shilong Ma; When to Update the Sequential Patterns of Stream Data; Pacific-Asia Conf. on Knowledge Discovery and Data Mining; 2003.

[Zhu, 2002] Yunyue Zhu, Dennis Shasha; StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time; Int'l Conf. on Very Large Data Bases; 2002.

Join Minimization in XML-to-SQL Translation: An Algebraic Approach

Murali Mani Song Wang Dan Dougherty Elke A. Rundensteiner
Computer Science Dept, WPI
{mmani,songwang,dd,rundenst}@cs.wpi.edu

Abstract

Consider an XML view defined over a relational database, and a user query specified over this view. This user XML query is typically processed using the following steps: (a) our translator maps the XML query to one or more SQL queries, (b) the relational engine translates an SQL query to a relational algebra plan, (c) the relational engine executes the algebra plan and returns SQL results, and (d) our translator translates the SQL results back to XML. However, a straightforward approach produces a relational algebra plan after step (b) that is *inefficient* and has redundant joins. In this paper, we report on our preliminary observations with respect to how joins in such a relational algebra plan can be minimized. Our approach works on the relational algebra plan and optimizes it using novel rewrite rules that consider pairs of joins in the plan and determine whether one of them is redundant and hence can be removed. Our study shows that algebraic techniques achieve effective join minimization, and such techniques are useful and can be integrated into mainstream SQL engines.

1 Introduction

Queries, and their corresponding algebra plans, generated automatically by translating queries specified over virtual views tend to have unnecessary joins [16]. Such algebra plans take much longer time to execute when compared to an equivalent algebra plan without the unnecessary joins. In this paper, we study the problem of how to remove unnecessary joins from a relational algebra plan.

As it sounds, this problem has been extensively studied in the more than thirty years of SQL and relational history [2, 1, 8, 15, 4]. In spite of the large amount of research, current SQL engines do very minimal join-minimization; the only kind of join minimization done is that of removing a join such as $A \overset{\bowtie}{c} B$, where c is a condition of the form $A.key = B.fk$, and $B.fk$ is foreign key attribute(s) of B that reference A. The reason for this minimal adoption is because existing solutions in research assume a set semantics, which give incorrect results when we assume bag semantics required by SQL. As a simple example, consider the algebra plan $\pi_{att_A}(A \times B)$, where A, B are relations, and att_A is the set of attributes of A. This plan returns the attributes of A after doing a cartesian product of A and B. The above plan is equivalent to the plan A, under set semantics. However, under bag semantics the above two plans give different results.

Motivating Example: Let us consider an example application scenario from the medical domain to illustrate the practicality of this problem. Consider two relations in the database of a primary clinic: one that describes doctors, and their speciality, and another that describes patients, who their primary doctor is, and what their primary health issue is. The two relations and their sample data are shown in Table 1.

docID	name	speciality
ID1	Mike	ENT
ID2	Mary	General
ID3	Cathy	General

(a) Doctor Relation with Sample Data

patID	name	primaryHealthIssue	doctor
SSN1	Matt	Arthritis	ID1
SSN2	Joe	Polio	ID1
SSN3	Mark	Cancer	ID2
SSN4	Emily	Arthritis	ID2
SSN5	Greg	Cancer	ID2
SSN6	Terry	Cancer	ID3
SSN7	Tina	Cancer	ID3

(b) Patient Relation with Sample Data

Table 1: Example Relational Database

Now consider that the primary clinic needs to export an XML view of its data to a certain class

of users. The view must specify the patients who have been diagnosed with cancer, and their primary health care physicians, grouped by the physicians. Such view definitions have been studied in several systems such as SilkRoute [5], XPERANTO [14], and CoT [10]. Fig 1 shows this view defined using XQuery [17] (this query is slightly modified from the one in SilkRoute [5] for ease of explanation).

```
<root> {
for $d in //Doctor
where exists (//Patient[@doctor=$d/@docID
          and @primaryHealthIssue='Cancer'])
return <doctor DoctorID={$d/@docID}>
  for $p in //Patient[@doctor=$d/@docID
          and @primaryHealthIssue='Cancer'])
  return <patient PatientID={$p/@patID}/>
</doctor> }
</root>
```

Figure 1: An XML view of the relational database from Table 1 defined using an XQuery

```
<root>
  <doctor DoctorID='ID2'>
    <patient PatientID='SSN3'/>
    <patient PatientID='SSN5'/>
  </doctor>
  <doctor DoctorID='ID3'>
    <patient PatientID='SSN6'/>
    <patient PatientID='SSN7'/>
  </doctor>
</root>
```

Figure 2: The result from a user query `/root` against the view defined in Figure 1

Such a view is typically virtual, and not materialized. Once such a view is defined, one needs to support arbitrary queries to be specified over this view. For instance, the result of the query `/root` is shown in Figure 2. Consider a user query U_1 that retrieves all the patient IDs in the view, which could be specified as `//patient/@PatientID`. Such a query could be answered using the following steps: (a) our translator translates the above XML query into SQL queries, (b) the relational engine translates an SQL query into a relational algebra plan, (c) the relational engine executes the algebra plan to get SQL results, and (d) our translator translates the SQL results back to XML to conform to the view. After these steps, the user will get the answer {SSN3, SSN5, SSN6, SSN7}[1].

[1]Note that we are assuming an unordered semantics. Considering order constraints such as SSN3 and SSN5 must appear next to each other are outside the scope of this work. Such unordered semantics as we assume might be appropriate, if the user knows that the underlying data source is relational.

For step (a), our translator uses a mapping as shown in Figure 3. This mapping says that one root node always exists in the view; the set of doctor children of this root node is the doctors that have a patient with cancer; given a doctor, her patients are those who have cancer. Such mappings are derived from the view query definition [5, 14].

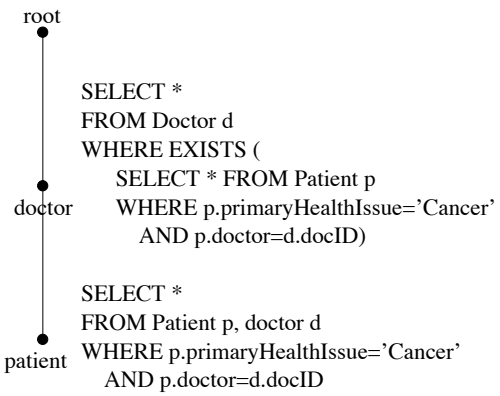


Figure 3: Mapping obtained from the view query (Figure 1) used to answer queries.

Let us see how the translator translates the user XML query U_1 into SQL using the above mapping. The translator can determine that the set of patients can be obtained from the SQL query corresponding to the patient node in the mapping. This query in turn uses the doctor node in the mapping, which in turn can be substituted by the SQL query corresponding to the doctor node in the mapping. After such substitutions, and some minor syntactic rewriting, we get the SQL query Q_1 that answers the user query as:

```
SELECT p.patientID
FROM Patient p,
     (SELECT * FROM Doctor d1
      WHERE EXISTS (
        SELECT * FROM Patient p1
        WHERE p1.primaryHealthIssue='Cancer'
              AND p1.doctor=d1.docID)) d
WHERE p.primaryHealthIssue='Cancer'
      AND p.doctor=d.docID
```

The above query specifies two joins: first there is a join between Doctor $d1$ and Patient $p1$ to produce d, that is the set of doctors who have cancer patients. This d is then joined with Patient p to get the final result. However, from the application semantics, we know that every patient who has cancer will appear in the view. Therefore a simpler SQL query Q_2 for answering U_1 would be:[2]

[2]Q_2 answers U_1 if we assume that every patient has one doctor. However even without this assumption, Q_1 can be optimized to a query which has only one join, as we will see later. In other words, Q_1 always has redundant joins.

```
SELECT p.patientID
FROM Patient p
WHERE p.primaryHealthIssue='Cancer'
```

Even if the query such as Q_1 specifies multiple joins, it might not be inefficient, if the relational engine can optimize the query. A relational engine first translates an SQL query into a relational algebra plan and tries to optimize this plan. This optimized plan is what is executed. However, when we feed Q_1 into a relational engine (we use IBM DB2 V8), we get a final plan that looks like the one shown in Figure 4. Observe that the plan still has the two joins.

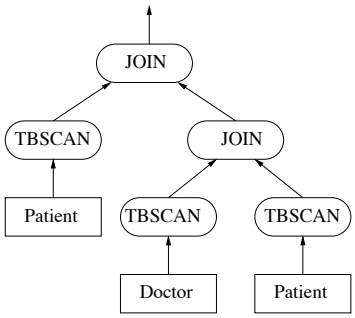


Figure 4: Algebra Plan corresponding to Q_1 generated by an SQL engine.

In this paper, we come up with a novel set of rules for minimizing joins in a relational algebra plan. Our rules determine whether a join in a algebra plan can be removed by examining other joins in the plan. Using our rules, as well as previously studied rules that minimize joins by examining semantic constraints in the schema, we are able to minimize the query plan in Figure 4 to an equivalent query plan without any joins.

Outline of the paper

The rest of the paper is organized as follows. Section 2 describes some of the related work in minimizing joins. Our rules for minimizing joins, along with an example illustration, are described in Section 3. We report on our preliminary experimental studies that show the performance gain possible by such join minimization in Section 4. Our conclusions and future directions are given in Section 5.

2 Related Work

Dan Suciu reported in [16] that the translator (step (a) in our process) in SilkRoute can produce SQL queries with unnecessary joins, and gave some insights as to why this problem might be more critical in the world of XML views, as opposed to plain SQL views. In XML views, there is a query associated with each "type" whereas in SQL views, there is only one "type" and a query associated with that type. Hence in XML views, queries that join multiple view queries are very frequent.

In [9], the authors study the problem of join minimization for XML views. Here the authors try to optimize step (a) (as opposed to step (b) in our approach). They do this by identifying which nodes in the view mapping such as Figure 3 form "bijective" mappings. A node in the view mapping is said to be a bijective mapping with respect to a relation in the SQL database, if there is an element of this node type in the view instance corresponding to every row in the relation. In our example view mapping shown in Figure 3, every row in the Doctor relation does not appear in the view; every row in the Patient relation also does not appear in the view. Therefore both the doctor node and the patient node in Figure 3 do not form bijective mappings. This means that the techniques studied in [9] will end up with an inefficient query plan such as the one in Figure 4.

In [10], the authors study a class of views where every node in the mapping is necessarily bijective. In other words, they disallow a view definition such as the one in Figure 1. By making this assumption, the authors are able to optimize step (a), and come up with minimal SQL queries easily: every XPath expression (or subexpression) that selects every element in the instance corresponding to a node can be obtained by a select query from the corresponding relation (and no joins are needed).

In the previous section, we mentioned the rich body of work that study join minimization assuming set semantics. In [2], Chandra and Merlin showed that there is a unique minimal query for any given conjunctive query, and that such minimization is NP-hard. In [1], the authors considered additional constraints such as functional dependencies specified on the relations, and came up with a *tableau* (matrix) based approach for decreasing joins. Minimization of joins in the presence of functional dependencies was also shown to be NP-complete in the size of the query. In [8], the authors considered functional and inclusion dependencies and showed that minimization of joins is still NP-complete. Here the authors came up with a *chase* technique that, given a query, expands the query by adding conjuncts based on the functional and inclusion dependencies. This expanded query can then be minimized. A graph based approach, consisting of *expansion* and *reduction* steps, for join minimization is studied in [15]. Recently, in [4], the authors consider physical structures such as primary and secondary indexes, extent-based representation of OO classes, join indexes, path indexes, access support relations, gmaps etc. The authors study how to

translate a logical query into a minimal query against the physical schema, using a chase step that expands the logical query to a universal query, and then a backchase step that minimizes the universal query.

The above approaches [2, 1, 8, 15, 4] do provide a good understanding of the problem; however, these techniques cannot be used in SQL engines, because SQL is based on bag semantics. The complexity of join minimization of conjunctive queries under bag semantics as in SQL is studied in [7, 3], and they report that query containment is Π_2^p-hard. Further, in [3], the authors consider `select-from-where` queries with bag semantics, and remark that such queries cannot be minimized without additional semantic constraints. In our work, we consider queries that produce semi-joins in the plans (such as queries with `exists`), and show that these joins can infact be reduced without any additional semantic constraints.

The approach that we propose for join minimization is an algebraic rewriting technique. Algebraic rewriting rules for SQL queries have been studied extensively, for example in [11, 12, 6, 13]. Some of the rules include removal of DISTINCT if one of the returned columns is known to be unique, techniques for decorrelation etc. However, none of the techniques study join minimization that can optimize the query plan shown in Figure 4. We expect that our techniques described in this paper will complement existing algebraic optimization techniques.

3 Rules for Minimizing Joins

In this section, we will describe our rules for minimizing joins in an algebra plan. We will state each rule informally, rather than using a formal notation, for ease of explanation. Further, we assume that some preliminary analysis of the algebra plan has already been done to identify characteristics such as for every operator, *what* columns are needed in the rest of the algebra plan (refer to any commercial optimizer like IBM DB2). We will use the following common notations for our relational algebra operators: select is denoted by σ; project is denoted by π; $\bowtie$ denotes join; $\ltimes$ denotes semi-join; $\overset{o}{\bowtie}_L$ denotes left-outer join; δ removes duplicates; γ denotes grouping.

Before we define the rules, we would like to introduce the notion of *logical entailment*. For instance, we say that the condition $(a = b) \wedge (c = d)$ logically entails the condition $(a = b)$. Given two conditions (boolean expressions) c_1 and c_2, c_2 logically entails c_1 if $c_2 \rightarrow c_1$ is always true. In other words, whenever c_2 evaluates to true c_1 will necessarily be true. A naive method for checking logical entailment is: identify common "terms" in c_1 and c_2 using syntactic analysis, and then check for all combinations of truth values of every term, whether $c_2 \rightarrow c_1$ is true.

Our first two rules are already studied and implemented in most commercial systems. They utilize semantic constraints (key-foreign key constraints) in the schema to remove joins.

Rule 1 $A \overset{\bowtie}{c} B$ can be reduced to $\sigma_{c'}(B)$ if c is a condition that logically entails the condition $A.key = B.fk$, where $B.fk$ is foreign key referencing A, no column in $B.fk$ can be NULL, and no column of A is needed in the rest of the algebra plan. c' is obtained from c by removing the condition $A.key = B.fk$. □

Rule 2 $A \overset{\bowtie}{c} B$ can be reduced to $\sigma_{c'}(B)$ if c is a condition that logically entails the condition $A.key = B.fk$, where $B.fk$ is foreign key of B that references A, and no column of A is needed in the rest of the algebra plan. c' is obtained from c by removing the condition $A.key = B.fk$, and by adding condition of the form $B.fk$ IS NOT NULL. □

Our third and fourth rules are more complex, and form the crux of our approach. They try to remove unnecessary semi-joins that may appear in the algebra plan. Semi-joins may appear in an algebra plan when we decorrelate a correlated SQL query. For example, consider the SQL query corresponding to the doctor node in Figure 3. It specifies a correlated query, which is translated into an algebra plan such as: $Doctor \overset{\ltimes}{c} Patient$, where c = (doctor = docID AND primaryHealthIssue = 'Cancer') is the join condition. The result of this semi-join is the set of rows in the *Doctor* relation, that satisfy the condition.

Now in Q_1, the above result is then joined with the *Patient* relation. The algebra plan corresponding to this is $(Doctor \overset{\ltimes}{c_1} Patient) \overset{\bowtie}{c_2} Patient$. Further, in this query the two conditions c_1 and c_2 are identical. In other words, the doctors who have patients are then joined with patients. We see that the first semi-join can be removed. We now get the query plan $Doctor \overset{\bowtie}{c_2} Patient$.

Rule 3 $(A \overset{\ltimes}{c_1} B) \overset{\bowtie}{c_2} B$ can be reduced to $A \overset{\bowtie}{c_2} B$ if the condition c_2 logically entails the condition c_1. □

The above rule can be implemented by doing a bottom-up traversal of the algebra plan. For any semi-join such as $(A \overset{\ltimes}{c_1} B)$, check if this operator has an "ancestor" operator in the plan that is a join with B, and has a join condition c_2 where c_2 logically entails c_1. This rule can be extended to an ancestor semi-join also, and the correctness holds.

Rule 4 $(A \overset{\ltimes}{c_1} B) \overset{\ltimes}{c_2} B$ can be reduced to $A \overset{\ltimes}{c_2} B$ if the conditions c_2 logically entails the condition c_1. □

Using the above rules, we can come up with an efficient relational algebra plan for Q_1, as shown in Figure 5. First we start with a plan that includes a semi-join and a join. Using Rule 3, we first remove the semi-join. We then use Rule 1 to remove the remaining join. The result is an efficient algebra plan with no unnecessary joins. In our experimental section, we show this efficient plan executes orders of magnitude faster; we achieved improvement of a factor of about 26 for simple queries[3].

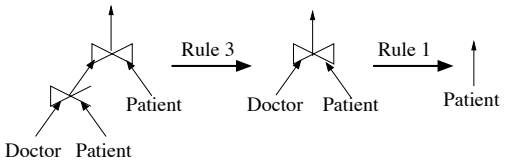


Figure 5: Using our rules to minimize relational algebra plan for query Q_1.

4 Experimental Evaluation

We performed some preliminary experiments to illustrate the effectiveness of our proposed approach. Our experiments were done on IBM DB2 V8 Database Server, which is installed on an 1.4 GHz Pentium machine with 512 MB RAM, running Windows XP. We used the TPC-H[4] benchmark data, loading data of different amounts from 500 MB to 4 GB.

We performed three sets of experiments. The first set of experiments illustrate that joins can be expensive. For this, we executed the following two queries:

Q_4:
```
SELECT COUNT (*) FROM LINEITEM l, PART p
  WHERE l.L_PARTKEY=p.P_PARTKEY
```

Q_5:
```
SELECT COUNT (*) FROM LINEITEM l
```

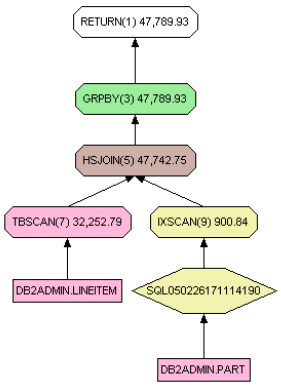
The plans for these two queries are shown in Figure 6. The execution times for these two queries against TPC-H data are shown in Figure 8. Note that this join can actually be very expensive, as it is not a key-foreign key join.

The second set of experiments is similar to our motivating example, and show the effectiveness of Rule 3. For this we executed the queries Q_6, and the equivalent query Q_5. Our rules are able to reduce Q_6 to Q_5. The plan for Q_6 is shown in Figure 7. The execution times for these two queries against TPC-H data are shown in Figure 8. Note that we get considerable performance gain using our rules.

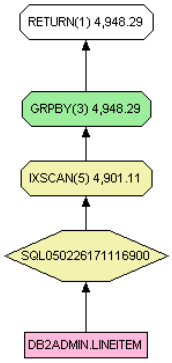
Q_6:
```
SELECT COUNT (*) FROM LINEITEM l,
    (SELECT * FROM ORDERS o1
```

[3]To clarify, for more complex queries, where the percentage of unnecessary joins is smaller, we expect to get lower factors of improvement, but larger absolute values of improvement.

[4]http://www.tpc.org



(a) Query Plan from Q_4



(b) Query Plan from Q_5

Figure 6: Illustrating that joins can be expensive. The execution times are shown in Figure 8.

```
WHERE EXISTS (
(SELECT * FROM LINEITEM l1
WHERE l1.L_ORDERKEY=o1.O_ORDERKEY)) o
WHERE l.L_ORDERKEY=o.O_ORDERKEY
```

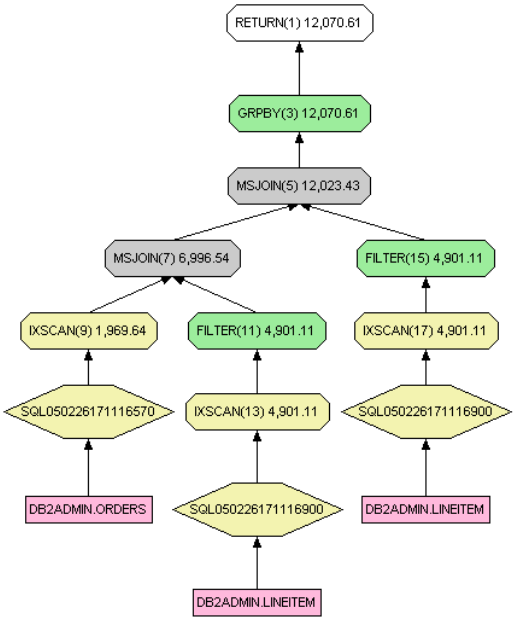


Figure 7: Query plan corresponding to Q_6. Our rules reduce this plan to the plan in Figure 6(b).

The third set of experiments illustrate the effectiveness of Rule 4. For this consider query Q_7 below:

```
SELECT COUNT (*) FROM LINEITEM l
WHERE EXISTS (SELECT * FROM ORDERS o1
        WHERE o1.O_ORDERKEY=l.L_ORDERKEY)
  AND EXISTS (SELECT * FROM ORDERS o1
        WHERE o1.O_ORDERKEY=l.L_ORDERKEY)
```

Using our Rule 4, we can remove one of the joins. We then get an algebra plan that is equivalent to the query Q_8 given below: (Execution times of Q_7 and Q_8 are shown in Figure 8.)

```
SELECT COUNT (*) FROM LINEITEM l
WHERE EXISTS (SELECT * FROM ORDERS o1
        WHERE o1.O_ORDERKEY=l.L_ORDERKEY)
```

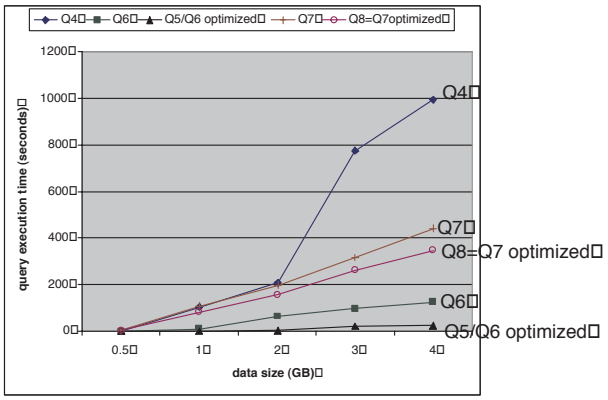


Figure 8: Execution times for the different queries

5 Conclusions and Future Work

In this paper, we have shown that significant performance gain can be achieved by performing join minimization, and that research so far has not solved the join minimization in a satisfactory manner. We have come up with a solution for join minimization that is based on the commercially used algebraic rewriting techniques and preserves SQL bag semantics. We expect that our work will open up renewed interest in this problem, and that the solutions will get adopted into commercial SQL engines. As part of future work, we need to integrate our solutions into commercial optimizers in order to study the query compilation time, as well as demonstrate the feasibility of our techniques.

References

[1] A. V. Aho, Y. Sagiv, and J. D. Ullman. "Efficient Optimization of a Class of Relational Expressions". *ACM Trans. on Database Systems (TODS)*, 4(4):435–454, 1979.

[2] A. K. Chandra and P. M. Merlin. "Optimal Implementation of Conjunctive Queries in Relational Data Bases". *ACM Symposium on Theory of Computing (STOC)*, pages 77–90, 1977.

[3] S. Chaudhuri and M. Y. Vardi. "Optimization of *Real* Conjunctive Queries". In *ACM PODS*, Washington, DC, May 1993.

[4] A. Deutsch, L. Popa, and V. Tannen. "Physical Data Independence, Constraints and Optimization with Universal Plans". In *VLDB*, Edinburgh, Scotland, Sep. 1999.

[5] M. F. Fernandez, Y. Kadiyska, D. Suciu, A. Morishima, and W. C. Tan. "SilkRoute: A Framework for Publishing Relational Data in XML". *ACM Trans. on Database Systems (TODS)*, 27(4):438–493, Dec. 2002.

[6] P. Gassner, G. M. Lohman, K. B. Schiefer, and Y. Wang. "Query Optimization in the IBM DB2 Family". *IEEE Data Eng. Bulletin*, 16(4):4–18, 1993.

[7] Y. E. Ioannidis and R. Ramakrishnan. "Containment of Conjunctive Queries: Beyond Relations as Sets". *ACM Trans. on Database Systems (TODS)*, 20(3):288–324, Sep. 1995.

[8] D. Johnson and A. Klug. "Testing Containment of Conjunctive Queries under Functional and Inclusion Dependencies". In *ACM PODS*, Los Angeles, CA, Mar. 1982.

[9] R. Krishnamurthy, R. Kaushik, and J. F. Naughton. "Efficient XML-to-SQL Query Translation: Where to Add the Intelligence". In *VLDB*, Toronto, Canada, Sep. 2004.

[10] D. Lee, M. Mani, F. Chiu, and W. W. Chu. "NeT & CoT: Translating Relational Schemas to XML Schemas". In *ACM CIKM*, McLean, Virginia, Nov. 2002.

[11] H. Pirahesh, J. M. Hellerstein, and W. Hasan. "Extensible/Rule Based Query Rewrite Optimization in Starburst". In *ACM SIGMOD*, San Diego, CA, June. 1992.

[12] H. Pirahesh, T. Y. C. Leung, and W. Hasan. "A Rule Engine for Query Transformation in Starburst and IBM DB2 C/S DBMS". In *IEEE ICDE*, Birmingham, UK, Apr. 1997.

[13] P. Seshadri, H. Pirahesh, and T. Y. C. Leung. "Complex Query Decorrelation". In *IEEE ICDE*, New Orleans, LA, Feb. 1996.

[14] J. Shanmughasundaram, J. Kiernan, E. Shekita, C. Fan, and J. Funderburk. "Querying XML Views of Relational Data". In *VLDB*, Roma, Italy, Sep. 2001.

[15] S. T. Shenoy and Z. M. Ozsoyoglu. "A System for Semantic Query Optimization". In *ACM SIGMOD*, San Francisco, CA, May. 1987.

[16] D. Suciu. "On Database Theory and XML". *ACM SIGMOD Record*, 30(3):39–45, Sep. 2001.

[17] W3C. XQuery Working Group. http://www.w3c.org/XML/Query.html.

Dynamic Count Filters

J. Aguilar-Saborit* P. Trancoso+ V. Muntes-Mulero*
J.L. Larriba-Pey*
*DAMA-UPC, Computer Architecture Department
Universitat Politecnica de Catalunya
+ Department of Computer Science
University of Cyprus
e-mail: {jaguilar,vmuntes,larri}@ac.upc.edu, pedro@cs.ucy.ac.cy

ABSTRACT

Bloom filters are not able to handle deletes and inserts on multisets over time. This is important in many situations when streamed data evolve rapidly and change patterns frequently. Counting Bloom Filters (CBF) have been proposed to overcome this limitation and allow for the dynamic evolution of Bloom filters. The only dynamic approach to a compact and efficient representation of CBF are the Spectral Bloom Filters (SBF).

In this paper we propose the Dynamic Count Filters (DCF) as a new dynamic and space-time efficient representation of CBF. Although DCF does not make a compact use of memory, it shows to be faster and more space efficient than any previous proposal. Results show that the proposed data structure is more efficient independently of the incoming data characteristics.

1. INTRODUCTION

Streamed data processing is the source for many interesting problems in areas ranging from financial analysis to telecom processing. In such cases, one of the basic problems is to recognize whether a new data item belongs to a set or to know its number of occurrences. The problem becomes challenging when there are large quantities of data to be processed per unit of time that evolve and change rapidly. In those situations, the problem calls for methods that are fast and adaptive.

Counting Bloom filters (CBF) [6] have been designed with some of the previous objectives in mind. CBF are similar to Bloom filters [1] but substitute every presence bit by a fixed size counter. This is a data structure with a fast access time but it suffers from an important problem which is related to the fact that its counters are not flexible. Consequently, the counters may become saturated resulting in inaccuracy in the stored information. As an alternative, *Spectral Bloom Filters* (SBF) [4] have been designed to overcome the lack of adaptiveness. SBF are composed of variable sized counters that adapt to rapidly changing data sets. However, SBF require the use of indexing structures to support this degree of adaptiveness, which make the access to each counter more complex and costly compared to CBF.

Counting Bloom Filters have been investigated for their use in network environments to summarize the content of peer-to-peer systems [3], to reduce file name lookups in large scale distributed systems [8], and in Internet Protocol routing lookups [5]. In the database environment CBF may be used to answer queries regarding the multiplicities of individual items, for example, in aggregate queries or ad-hoc iceberg queries [7, 9].

Besides providing a new data structure to represent counting filters, the Spectral Bloom Filters also propose new methods for reducing the probability and magnitude of lookup errors [4]. Bloom histograms, a further compressed view of SBF, are used to keep counting statistics for paths in XML Data [12].

In this paper we propose a new data structure to represent CBF, that we name *Dynamic Count Filters* (DCF). The DCF structure is designed for speed and adaptiveness in a very simple way. It captures the best of SBF and CBF. As a by-product of its simplicity, DCF does not require the use of indices. This fact reduces the amount of memory requirements in most of the cases.

In Table 1 we present a brief qualitative comparison between the three approaches: CBF, SBF, and DCF.

	Counters size	Access Time	#Rebuilds	Saturated counters
CBF	Static	fast	n/a	Yes
SBF	Dynamic	slow	high	Eventually
DCF	Dynamic	fast	low	No

Table 1: Qualitative Comparison of CBF, SBF, and DCF.

It is relevant to notice that DCF borrows the qualities of the two other techniques, the dynamic counters from SBF and the fast access from CBF. Also, DCF's dynamic counters avoid saturation. Finally, although the cost of a single rebuild of our data structure is high and only slightly better than that to rebuild SBF's, the number of rebuilds for DCF is orders of magnitude smaller, leading to a much smaller overall execution time.

The results from the execution of different real-life data operation scenarios, using both DCF and SBF to represent the data, show that DCF's overall memory size less than the half compared to SBF's, and its overall execution time is less than half the execution time of SBF. In addition, the flexibility of the DCF structure sustains its accuracy even in the presence of unpredicted peaks in the data set size.

The contributions of this paper are as follows:

- The proposal of *Dynamic Count Filters* (DCF) with a detailed description of the basic data structure and operations.
- An efficient mechanism to dynamically resize the DCF structure and avoid useless rebuilds.

- A quantitative evaluation of DCF as well as its comparison with SBF.

This paper is organized as follows: In Section 2 we explain the related work. Section 3 describes the Dynamic Count Filters. In Section 4 we present the setup and discuss the experimental results for the different scenarios. Finally, in Section 5 we present the conclusions.

2. RELATED WORK

A Bloom Filter, proposed by Burton Bloom in 1970 [1], is basically a bit-vector of m bits that represents a set of n data elements, $S = s_1, ...s_n$. The Bloom Filter uses k hash functions [10, 11], $h_1, h_2, ..., h_k$, that map each data element into the Bloom Filter. Each hash function returns a value ranging from 1 to m, thus for each data element, $s \in S$, positions $h_1(s), h_2(s), ..., h_k(s)$ are set to 1. Different data elements from S may map to the same position in the filter, hence, a given data element is in S, $s \in S$ with a given probability of error [1], only if $h_i(s) = 1$ for $1 \leq i \leq k$.

Bloom filters do not address the issue of deletions over multisets. In order to overcome this limitation, Fan *et al.* [6] proposed the *Counting Bloom Filters* (CBF), where a Bloom filter is extended to have the capability of counting the number of different data elements that map to the same location.

2.1 Counting Bloom Filter (CBF)

A CBF represents a total of M data elements, including repeated values. This is done by replacing bit entries of a Bloom Filter by counters ($C_1, C_2, ..., C_m$ counters). Similarly to the original Bloom Filter, each time a new data element s is to be inserted into the set, k hash functions are used to update k entries of the filter. While in the Bloom Filter the entries would be simply set to one, in the CBF the counter in each of the k entries is incremented by one. In an analogous way, whenever a data element s is to be removed from the set, the counters in the same k filter entries are decremented by one. At any point in time the sum of the contents of all counters is equivalent to $\sum_{j=1..m} C_j = k \times M$.

The usual data structure representing the CBF consists of a static data set representation where counters have a fixed size over time. Such a representation has two major drawbacks: (1) whenever an insertion of a new element results in a counter overflow, delete operations to data elements that map to that same filter entry can no longer be reflected; (2) CBF's representation is not optimal as all counters have the same bit length, thus resulting in memory waste.

Dharmapurikar *et al.* [5] addresses the problem of overflowed counters. In their proposed approach, the CBF structure is rebuilt with a larger size once the number of overflowed counters passes a certain threshold. As such, the refresh of the structure is very costly, as far as all messages must be re-inserted again. Moreover, overflowed counters may be useless during a large period of time.

2.2 Spectral Bloom Filter (SBF)

Cohen and Matias [4] proposed the Spectral Bloom Filter (SBF), which is a compact representation of the CBF. The main goal of SBF is to achieve an optimal counter space allocation. It consists of a compact base array of a sequence of $C_1, C_2, ..., C_m$ counters, which represents a set of M data elements using k hash functions as with CBF. At any point in time, the goal of SBF is to keep the size of the base array as close to N bits as possible, where $N = \sum_{j=1..m} \lceil logC_j \rceil$. Note that throughout this paper we assume log to be $\log_2$.

To achieve its goal, each counter C_j in the SBF structure, dynamically varies its size such that it has the minimum necessary bits needed to count the number of items hashed into position j. To allow for this flexibility, the counter space in SBF includes $\varepsilon \times m$ *slack bits* that are placed among the counters. A slack bit is added between every $\lfloor \frac{1}{\varepsilon} \rfloor$ counters, where $0 < \varepsilon \leq 1$.

While the counter space in SBF is kept close to the optimal value, in order to support the flexibility of having counters with different sizes, SBF requires complex index structures. In the context of our paper, we identify the index structures described in [4] as:

- *Coarse Vector* (CV): a bit-vector index that provides offset information for the beginning of a subgroup of counters. Offsets are provided using counters of a fixed-size length in bits.
- *Offset Vector* (OV): a bit-vector which provides straightforward representation of the offsets provided by the CV.

Figure 1 shows the data structures used by SBF. The first-level Coarse Vector (CV1) contains $\frac{m}{\log N}$ offsets of $\log N$ bits each, thus, each offset represents a subgroup of counters (SC). As explained in detail in [4], for the subgroup of counters that fulfills $\sum_{C_j \in SC} \log\lceil C_j \rceil < \log^3 N$, a second-level coarse vector (CV2) is required, providing a more detailed information about the offsets for each counter $C_j \in$ SC. In the case that a subgroup of counters $\sum_{i \in SC} \log\lceil C_j \rceil \geq \log^3 N$, then, as indicated in [4], an Offset Vector (OV) is used, that contains the exact offset for each counter. For simplicity, without loss of generality, the analytical models presented in this paper assume the former case, where the CV2 is required. CV2 divides SC in chunks of $\log \log N$ counters (SC$'$), and holds a total of $\frac{\log N}{\log \log N}$ offsets. Since offsets in CV2 are at most $\log^3 N$, each offset can be represented with $3 \log \log N$ bits, totaling $3 \log N$ per each SC$'$. Finally, the information needed to locate the exact position of the j^{th} counter is given by one offset vector (OV), one per each subgroup SC$'$. The OV consists of $\log \log N$ offsets of $3 \log \log N$ bits each offset, totaling $3(\log \log N)^2$ bits per subgroup SC$'$.

The offset vector OV, can also be substituted by a lookup table depending on a threshold based on the length of SC$'$. More details about this approach can be found in [4].

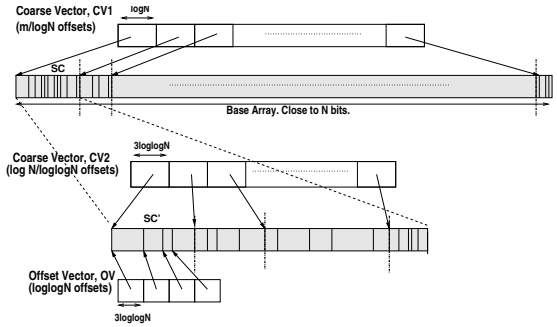


Figure 1: SBF data structures.

3. DYNAMIC COUNT FILTERS (DCF)

Dynamic Count Filters (DCF) are composed of two different vectors. The first vector is a basic CBF with each entry being a counter of fixed size $x = log\frac{M}{n}$, where M is the total number of data elements in the set and n is the number of distinct values in the set. If we consider that the filter has m counters (C_j for $j = 1..m$), the CBF vector, hereby named CBFV, accounts for a total of $m \times x$ bits. The second vector is the *Overflow Counter Vector* (OFV), which also has the same number of entries, each one including a counter (OF_j for $j = 1..m$) that keeps track of the number of times that the corresponding entry in the CBFV suffered an overflow. The size of each counter in the OFV changes dynamically depending on the distribution of the data elements in the data set. At a certain point in time, the size of each counter is equal to the number of bits required to represent the largest value stored in OFV ($y = \lfloor \log(\max(OF_j)) \rfloor + 1$). As such, the size of the OFV accounts for a total of $m \times y$ bits.

Figure 2 represents the data structure for the DCF approach. From this Figure it is possible to observe that the DCF data structure is composed of m entries with m counters split in pairs of counters, $\langle C_1, OF_1 \rangle$,...,$\langle C_m, OF_m \rangle$. All counters in the DCF have equal size of $x + y$ bits, where y varies dynamically its bit length.

The decision of having a fixed size for each counter implies that, on the one hand many bits in the DCF structure will not be used and, on the other hand, the access to both vectors is direct, hence, fast. Therefore, DCF trades counter memory space for a fast access. Overall, DCF's fixed-sized counters result in time and space benefits as it allows for a fast read/write mechanism that has an asymptotic cost of $O(1)$, avoiding the use of complex indexing structures, and saving memory space in most of the cases.

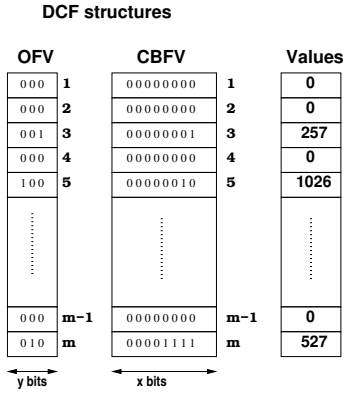


Figure 2: DCF data structure.

3.1 Querying a Data Element

Querying the filter for a certain data element s results, as with the rest of Bloom-like filters, in checking k filter entries, using k different hash functions, $h_1(s), h_2(s), ..., h_k(s)$. Checking a filter entry is performed by accessing the corresponding entries of the CBFV and OFV vectors. As counters in both vectors have the same number of bits, locating an entry is immediate and performed with simple shift and modulo operations. The counter bits are then extracted from the vectors using fast bitwise masking operations (AND and OR). Both accesses are fast with an asymptotic cost of $O(1)$. Hence, for a certain entry j, once we get C_j and OF_j, the compound value V_j for the counter stored in position j of the DCF is calculated as $V_j = (2^x \times OF_j + C_j)$.

This way, when querying a data element s, we end up having k values $V_{h_i(s):i=1..k}$ with a cost of $k \times O(1) \simeq O(1)$. The values associated to s are used depending on the application. For instance if we want to determine the presence of the data element, it is necessary to perform the simple operation of checking if one of the $V_{h_i(s):i=1..k}$ values is zero. If so, then s is not in the data set, otherwise s is in the data set with a probability of false positive as explained in [1].

3.2 Updating the Data Set

Each time we insert or delete a data element s, we must update k filter entries ($h_1(s)$, $h_2(s)$, ..., $h_k(s)$) in the CBFV and OFV. Updates in the CBFV are fast and performed on each counter $C_{h_i(s):i=1..k}$ using simple increment and decrement operations, depending on the operation, insert or delete, respectively. Updates in the OFV are more infrequent but may be more expensive. In addition to updating the $OF_{h_i(s):i=1..k}$ counters when an overflow or underflow occurs in the corresponding $C_{h_i(s)}$ counter, the OFV may need to be resized. In the next sections we explain in more detail the insert and delete operations for a certain data element s.

Inserting a Data Element.

As mentioned before, when a data element s has to be inserted, k counters $C_{h_i(s):i=1..k}$ in the CBFV are incremented by one. In case a counter C_j, for any of those incremented entries $j = h_i(s) : i = 1..k$, overflows, *i.e.* its value increases from $2^x - 1$ to 2^x, the value of C_j is set to zero and the corresponding counter in OFV, OF_j has to be incremented by one. Thus, the data insertion requires at most two read and write operations, which have an asymptotic cost of $O(1)$.

In the case that the overflow counter OF_j is to be incremented from $2^y - 1$ to 2^y, then, before the operation can be performed, one bit must be added to all counters in the OFV in order to avoid counter saturation. We name the action of changing the size of the OFV *Rebuild*. Rebuild operations are expensive as they require the allocation of a new vector, the copy of the contents from the old OFV to the new extended OFV vector, and finally the deallocation of the old OFV vector. Therefore, as the vectors have m entries, the rebuild operation has an asymptotic cost of $O(m)$. Notice that although the rebuild operation is costly, the motivation for having a OFV structure separately from the CBFV is that rebuilding the OFV is cheaper than rebuilding the CBFV. This is because the OFV is smaller than the CBFV and also because it is probable that many more entries contain a zero in the OFV, which will not be the case for CBFV, thus. Note that containing a zero implies no need to copy the counter. Overall, although the asymptotic cost would be the same, memory allocations and data copying for the new OFV vector are cheaper than if we re-create the whole DCF structure.

Delete a Data Element.

Upon deletion of a data element s from the data set, k counters $C_{h_i(s):i=1..k}$ in the CBFV are decremented by one. Whenever one of the counters suffers an underflow, *i.e.*, C_j, for any of those decremented entries $j = h_i(s) : i = 1..k$, is to be decremented but its original value is zero, then its value is set to $2^x - 1$ and its corresponding counter in the OFV, OF_j is decremented by one. Therefore, as in the insert case, at most only two read and write operations are needed, resulting in an asymptotic cost of $O(1)$. Notice that when we decrement a counter, either C_j or OF_j must be

greater than 0, as we do not delete data elements not belonging to the data set.

Similarly to the insert-triggered OFV rebuild, delete operations may also result in OFV rebuilds, in this case in order to save counter memory space that is not needed any longer. Whenever a counter OF_j is decremented from 2^{y-1} to $2^{y-1}-1$ we may check all the other OFV counters and if their values are all smaller than 2^{y-1} then we can reduce the OFV size by one bit per counter. While in theory this operation requires to check all the counter values, in practice this operation is simpler if a simple counter structure keeps track of the bit-usage for the m counters. We show this optimization in the next Section. Shrinking and enlarging the OFV result in the same DCF rebuild operation with an asymptotic cost of $O(m)$.

3.3 Delayed OFV Shrinking

The main difference between the rebuild due to insertion and the rebuild due to deletion is that while the former is required in order to avoid counter saturation, the latter is optional and may be delayed in order to avoid unstable situations of consecutive delete/insert operations that could result in excessive OFV rebuilds.

Therefore, we introduce a threshold between values 2^{x+y-2} and $2^{x+y-1}-1$. We define such threshold as $T = 2^{x+y-2}+(2^{x+y-1}-2^{x+y-2})\times\lambda$, where λ ranges from 0 to 1. Hence, when decreasing entry j in the DCF by one, and being V_j the associated value to the counter $\langle OF_j, C_j\rangle$, then, if $V_j < T$, we rebuild the OFV when all the counters in the OFV $\forall j : j = 1..m : V_j < T$.

Threshold Maintenance

As mentioned in Section 3.2, in order to avoid checking all the counter values for underflow, we perform an optimization and keep an overflow counter structure. We name l, the *overflow level* of any counter in the DCF structure. The overflow level represents the number of bits used in the overflow counter and therefore, it may have a value between 0 and y. Consequently, an entry j in the DCF has overflow level $l > 0$ if its overflow counter OF_j has a value between $2^{l-1} < OF_j \leq 2^l - 1$. It has an overflow level $l = 0$ if $OF_j = 0$. We arrange the different *overflow level* counters into a structure called the *Counter Level* (CL).

In order to use the threshold T as described before, we keep two counters per level in the CL: (1) a counter for the number of OF_j counters with value less or equal than the threshold for the level they belong, *i.e.* counters below $2^{x+y-2}+(2^{x+y-1}-2^{x+y-2})\times\lambda$ for values with a level greater than 0, and below $(2^x-1)\times\lambda$ for counters in level 0; and (2) a counter for the number of OF_j counters with a value equal or larger than the threshold. The level l of a counter stored in position j in the DCF structure, is calculated as:

$$\begin{cases} 0 & \text{if } OF_j = 0; \\ \lfloor log(OF_j)\rfloor + 1 & \text{otherwise.} \end{cases}$$

Then, whenever a data element s is either inserted or deleted and values $V_{h_i(s):i=1..k}$ either incremented or decremented, we must update the corresponding level counters.

Note that the storage needed by the CL structure is negligible: we only need $2\times(y+1)$ counters. Also, the updating process is of asymptotic cost $O(1)$, we only perform simple additions subtractions and comparisons.

Figure 3 shows an example of how the CL structure is used for delayed OFV rebuilds resulting from data deletion operations. In this example we show a DCF structure with eight counters, $m = 8$, and $\lambda = 0.5$. CBFV has $x = 4$ bits per counter, and after several inserts the OFV has $y = 2$ bits. The decimal values of each counter are shown only for clarity of the example, as they have the same information as their corresponding pair of counters. Initially, on the left side, the values for each counter are $\{0, 2, 7, 31, 9, 28, 17, 60\}$, thus, the counters in position 0, 1, 2, and 4 are of level $l = 0$ (with three counters below the threshold and one above), those in positions 3, 5, and 6 are of level $l = 1$ (with one counter below the threshold and two above), and the last counter is of level $l = 2$ (with the counter above the threshold). In the central part of Figure 3, we can see that, after several deletes, level 2 counters and the level 1 counter above the threshold reach zero. Therefore, as all counters in the DCF are below the threshold, we can rebuild the OFV by deleting one bit per position and consequently reducing the size of the whole DCF structure.

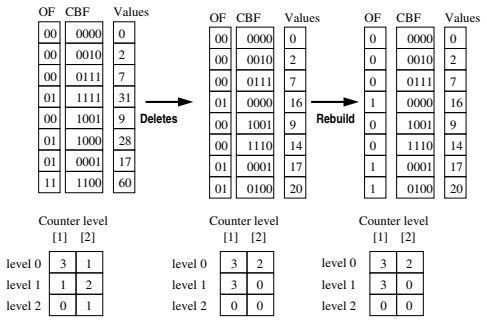


Figure 3: Counter Level and OFV rebuild in DCF.

Choosing the Optimal Threshold λ.

Rebuilds are the most costly operations performed in the DCF structure. Thus, we define as the optimal threshold for DCF, the value of λ which minimizes the number of rebuilds over time.

First, we must determine which are the situations worth to rebuild the structure in case of deletions. For this, we define the ratio $R = n_{inserts}/n_{deletes}$, which is a metric that gives an indication of how the number of insert operations evolve comparing to the number of delete operations over time.

Figure 4 shows the evolution of DCF in terms of the number of rebuilds over time. The chart includes three lines, representing three different scenarios, *T1*, *T2*, and *T3*, each one representing a different ratio of R: T1 for $R > 1$, T2 for $R \simeq 1$, and T3 for $R < 1$.

The results in this chart represent the average number of rebuilds for ten executions using different λ values ranging from 0.1 to 1.0. The first time steps for all scenarios consist of insert operations for the M data elements. After having populated the data set, each time step is composed of both insert and delete operations. The data inserted follows a Zipfian distribution, where the skew defined by θ is randomly selected from a range between 0 and 2, *i.e.* $0 \leq \theta \leq 2$ [2]. Items inserted are also randomly selected for deletion during the delete operations.

From Figure 4 we can identify the following behavior:

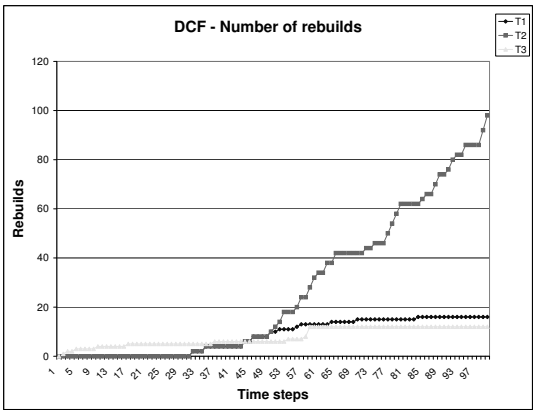


Figure 4: Number of rebuilds in DCF.

- T1 shows, in average for any λ, the number of rebuilds occurred when R grows as time passes. In this case, we are always increasing the number of useless rebuilds caused by deletions, *i.e.* in a short time, we may have a rebuild caused by an insertion, because insertions are more frequent.
- T2 and T3 show that, in average for any λ, the number of rebuilds does not increase when $R \simeq 1$ and $R < 1$ respectively. Hence for T2 and T3 scenarios there may exist a value of λ which minimizes the number of rebuilds, and improves the memory usage of the DCF.

Figure 5 shows the results for different values of $R < 1$, that are the values of λ that minimize the number of rebuilds. The plot shows that for ratios $R \leq 0.6$, where $n_{deletes} \gg n_{inserts}$, independently of the λ we choose, we always have a constant number of rebuilds. However, for $R > 0.6$, the value set for λ becomes important, and values of $\lambda \simeq (1 - R)$ are those that minimize the number of rebuilds. When the incoming data is highly skewed, *i.e.* inserts and deletes often affect the same counters, the number of rebuilds is more sensitive to the selected threshold value. We can see that for $\lambda = (1 - R)$ we have the least number of rebuilds.

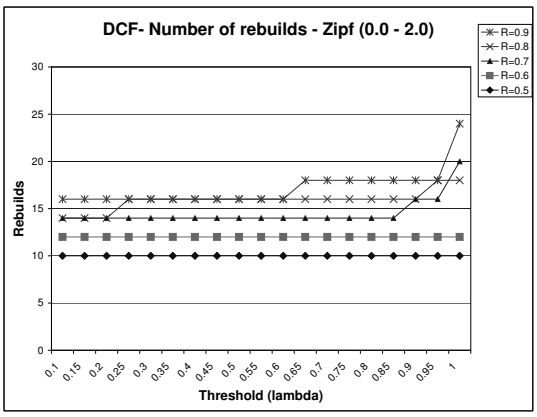


Figure 5: Number of rebuilds in DCF for different λ threshold values and insert-to-delete ratio for Zipfian distributions with θ between 0-2.

4. EXPERIMENTAL RESULTS

For the evaluation of our proposed DCF structures, we have implemented the DCF and SBF representations of the Counting Bloom filters in C language and compiled the programs using full optimization (-O3). The implementation of the SBF has followed the exact specifications given in [4] and detailed in Section 2. We run our tests on a 750MHz IBM PowerPC_RS64-IV processor with 16GB of main memory. The Operating System is AIX version 5.1.

The data we use is based on a total of n distinct values with multiplicities that are inserted and deleted from a data structure. The total number of data elements is $M = av * n$, where av is the average number of operation multiplicities per distinct value. Our experiments consist of performing M consecutive data insert operations, followed by M consecutive data delete operations, picked from the n distinct values in the data set following a Zipfian distribution [2], where the skew is defined by the parameter θ that ranges from 0 to 2. Values for $\theta \simeq 0$ represent uniformly distributed data, while values $\theta \simeq 2$ represent highly skewed data. We use integers as data values.

In order to analyze the behavior of DCF in detail, we focused on a set of experiments aimed at evaluating the performance in terms of access time, memory usage and impact of rebuild operations for the DCF structure, in comparison to the SBF approach.

The number of counters m both for DCF and SBF depends on the fraction of false positives F_p, the number of distinct values n, and the number of hash functions used k. We set by default $F_p = 0.05$ and $k = 3$. The number of distinct values n may change, and is specified for each experiment we describe. Also, the total amount of values M may vary for the different tests.

One metric used to measure the accuracy of the filter is the *accurate representation* of a data element. This is defined as follows: a data element s has an *accurate representation* if the minimum value stored in the k counters matches the real number of times s has been inserted in the data set.

Read, Write, and Rebuild Time

Figure 6 shows the average time to perform a read, a write, and a rebuild operation for both SBF and DCF structures for different data set sizes. The results are shown for four different setups where the number of distinct values n is 1000, 10000, 100000, and 1000000, respectively. For each test, we consider uniform data and a total amount of $M = 100 \times n$ values were first randomly inserted, and then, randomly deleted. The results are expressed in μseconds and the y-axis is represented using a logarithmic scale. Each value in the chart represents the average, for each different operation, over all the instances of each operation during the complete execution of the test.

From the results in Figure 6 it is possible to conclude that DCF is more efficient in terms of the time to access the structure, either read or write operations, compared to the SBF structure. The cost to locate a counter in the DCF is very efficient compared to the SBF lookup that requires traversing the index structures.

Note that as the number of distinct values increases, both techniques keep the read and write times constant. Another important fact is that the rebuild operations, as expected, prove to be the most costly operations. DCF is, in average, slightly faster than SBF and its execution time shows to grow with a similar rate as SBF. How-

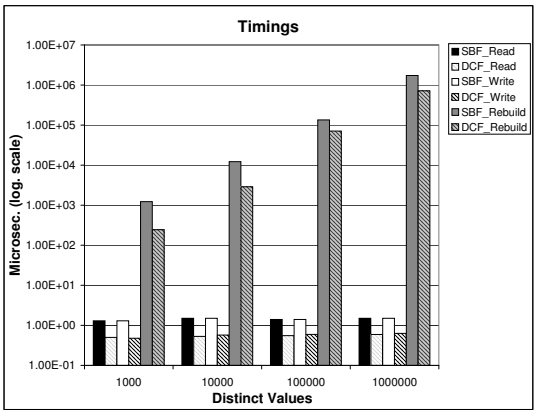


Figure 6: Average time for read, write and rebuild operations for SBF and DCF.

ever, note that DCF requires a total number of rebuilds which is orders of magnitude smaller than SBF. For instance, for $n = 10$K the number of rebuilds for DCF is 8 while that for SBF is more than 45000, and for $n = 1$M the number of rebuilds for DCF just grows to 9 while that for SBF grows to more than 4.5 million rebuilds. This comes from the fact that a DCF rebuild for one counter, automatically rebuilds the rest of the counters, growing the OFV data structure by one bit vector of size m. On the other hand, each SBF rebuild involves only one counter at a time.

Memory Usage

Figure 7 shows results obtained through several static executions varying the skew (Zipfian with $0 \leq \theta \leq 2$) and the total amount of values M. Values are first randomly inserted, and then, randomly deleted. In that chart we show the ratio $R = \frac{Mem_{SBF}}{Mem_{DCF}}$. Notice that while Mem_{DCF} varies dynamically, during the execution of the test, for this static analysis we show only the maximum memory size that DCF requested for the complete execution.

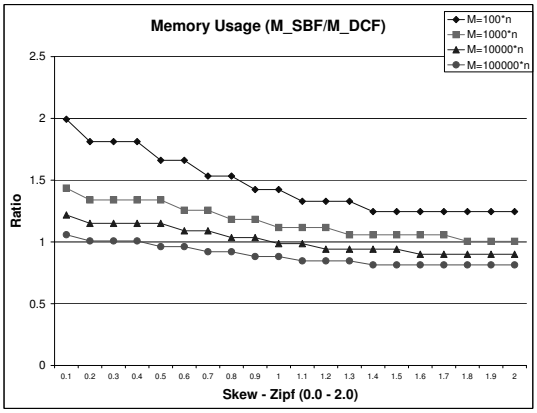


Figure 7: SBF-to-DCF memory ratio for different data distributions and data set sizes.

The chart in Figure 7 shows that, as expected, in the most usual situations DCF needs less memory than SBF. There are two extreme cases: one for non-skewed, almost uniform, data distributions (Zipfian with $\theta = 0.1$), where DCF uses less than the half of the space used by SBF; and another one for highly skewed data or when there is an extremely large multiplicity of elements for a single value, where SBF consumes less memory than DCF.

	Memory (bytes)	Accuracy (%)	Access Time (μsec)
SBF	17280	90.1	1.3
DCF	20125	90.1	0.5
DCF bounded	16950	87.6	0.5

Table 2: SBF and DCF for large number of repeated values.

For those cases where DCF does not behave better in terms of memory usage, we want to understand how the DCF approach would behave if we limited its memory to the maximum memory used by SBF.

Large number of repeated values with limited memory

Table 2 shows the results of the execution for $n = 1000$ distinct values and $M = 10^9 = 10^6 \times n$ uniformly distributed data elements. On the one hand, we observe that without memory limitations, DCF would use 14% more memory than SBF, achieving the same accuracy and being more than two times faster than SBF in access time. On the other hand, if we have a limited amount of memory, forcing DCF to use the same memory as SBF only decreases its accuracy in 2.5%. However, the access time in this case is still more than two times faster than that of SBF.

Execution Time

Figure 8 shows execution time and number of rebuilds (logarithmic scale) for executions where we vary the degree of skewed data from $0 \leq \theta \leq 2$. Each execution consists of the insertion and later deletion of $M = 100 \times n$ items for $n = 1000$ distinct values. All values are inserted first, filling the structure, and then, randomly deleted until the structure becomes empty.

As it is possible to observe from the results in Figure 8, DCF clearly outperforms SBF independently of the incoming data distribution. The gap between both approaches is smaller as the data are more skewed. This fact is expected as for skewed data, the number of rebuilds performed by the SBF decreases, while at the same time the opposite effect happens for CBF. In Figure 9 we show the percentage of the total execution time spent in rebuilding the structure. It is possible to observe for SBF most of the time is spent rebuilding the structure, due to the high number of rebuilds. In contrast, the time that DCF spends in rebuilds is minimum. Although a rebuild for DCF is costly, this approach performs fewer rebuilds during the complete execution. Consequently, even for highly skewed data, DCF spends almost all of its execution time in the read and write operations.

Overall, DCF is faster for the read and write operations. Moreover, although the rebuild operation is costly, it is only performed a few times during the complete execution which results in a clear benefit compared with SBF.

5. CONCLUSIONS

In this paper we propose a new representation of the Counting Bloom Filters to cope with inserts and deletes in multisets over

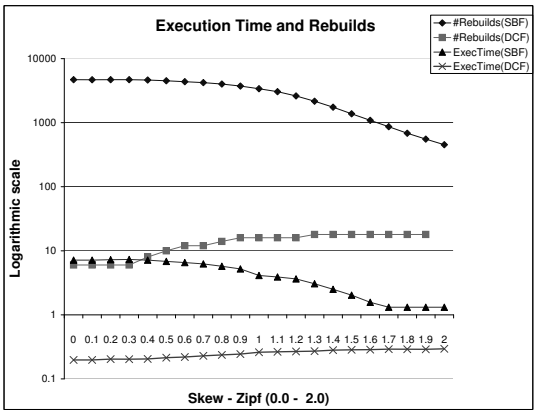


Figure 8: Total execution time.

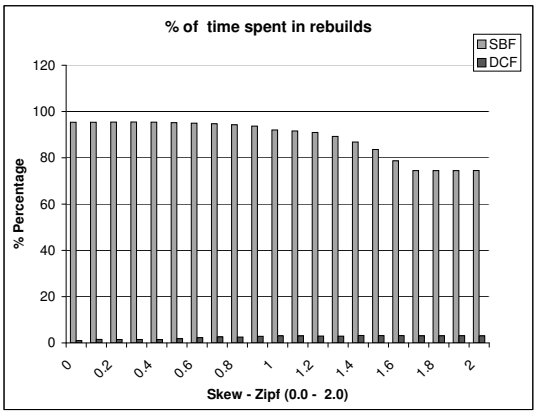


Figure 9: Percentage of time spent in rebuilding the structure.

time: the Dynamic Count Filters (DCF). Our data structure borrows the qualities of previous proposals. On the one hand, it has the fast access times of the Counting Bloom Filters (CBF). On the other hand, it has the adaptivity to changing data patterns of the Spectral Bloom Filters (SBF).

Dynamic Count Filters also show other interesting properties. First, in general, DCF uses a smaller amount of memory than SBF for a fixed amount of counters. However, although in extreme cases (very large number of replicated values) DCF uses a larger amount of memory than SBF, the gains obtained in execution time (about 2 times faster than SBF), make Dynamic Count Filters worth the small additional cost. Second, for a fixed amount of memory, DCF may include more counters and is faster. Third, the total cost for rebuilding DCF is significantly lower than that for SBF.

Overall, we can claim that Dynamic Count Filters are a data structure to be taken into account in many practical situations for their fast access times and for their ability to adapt to the dynamic evolution of data and to situations with small amounts of memory available.

6. ACKNOWLEDGEMENTS

UPC authors thank the Computer Architecture Department, and IBM for the continuous support to our research through CAS grants and fellowships. All the authors thank the HPC-Europa programme, funded by the European Commission's Research Infrastructures activity under contract RII3-CT-2003-506079.

7. REFERENCES

[1] Burton H. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Communications of the ACM*, 13(7):422–426, 1970.

[2] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and zipf-like distributions: Evidence and implications. *In Proceedings of the IEEE Infocom Conference*, 1999.

[3] Andrei Broder and Michael Mitzenmacher. Network Applications of Bloom Filters: A survey. *A survey. In Proc. of Allerton Conference*, 2002.

[4] Saar Cohen and Yossi Matias. Spectral bloom filters. *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 241–252, 2003.

[5] Sarang Dharmapurikar, Praveen Krishnamurthy, and David E. Taylor. Longest prefix matching using bloom filters. *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 201–212, 2003.

[6] L. Fan, , P. Cao, J. Almeida, and A. Z. Broder. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE Trans on Networking*, 8(3):281–293, 2000.

[7] Min Fang, Narayanan Shivakumar, Hector Garcia-Molina, Rajeev Motwani, and Jeffrey D. Ullman. Computing Iceberg Queries Efficiently. *VLDB '98: Proceedings of the 24rd International Conference on Very Large Data Bases*, pages 299–310, 1998.

[8] Jonathan Ledlie, Laura Serban, and Dafina Toncheva. Scaling filename queries in a large-scale distributed file systems. Research Report TR-03-02, Harvard University, January 2002.

[9] G. Manku and R. Motwani. Approximate frequency counts over data streams. *In Proceedings of the 28th International Conference on Very Large Data Bases*, 2002.

[10] James K. Mullin. A second look at bloom filters. *Commun. ACM*, 26(8):570–571, 1983.

[11] M. V. Ramakrishna. Practical performance of Bloom filters and parallel free-text searching. *Commun. ACM*, 32(10):1237–1239, 1989.

[12] Wei Wang, Haifeng Jiang, Hongjun Lu, and Jeffrey Xu Yu. Bloom histogram: Path selectivity estimation for xml data with updates. *VLDB'04: Proceedings of the Thirtieth International Conference on Very Large Data Bases*, pages 240–251, 2004.

Towards a Dynamic Multi-Policy Dissemination Control Model (DMDCON)

Zude Li, Xiaojun Ye
Institute of Information System and Engineering,
School of Software, Tsinghua University, Beijing, China 100084
li-zd04@mails.tsinghua.edu.cn, yexj@tsinghua.edu.cn

Abstract

Dissemination control (*DCON*) is a security policy of controlling digital resource access before and after distribution. It is an extension of traditional access control within client-side domain, digital rights management by payment-free applications, and originator control on recipients' re-dissemination rights allowance. Different application domains may adopt dynamically different resource dissemination policies, but current *DCON* models cannot solve the multi-policy coexistence and compatibility problems. A dynamic multi-policy dissemination control model (*DMDCON*) is proposed to express the dynamic and multi-policy nature existing in reality, which are indispensable for well formed resource dissemination control application. The goal of this paper is to de ne and extend formally some basic concepts related with resource dissemination (such as dissemination policy, chain, tree, etc.) and further, propose a comprehensive *DMDCON* model to describe universal resource dissemination applications through specifying temporal dissemination features, restrictions, and policy revocation (cascade or non-cascade). Finally, we brie y discuss the importance of *DCON* within the usage control domain.

Keywords

dissemination control, dissemination tree, active time range

1 Introduction

Dissemination control (*DCON*) is one of the most important and challenging goals for information security, which is concerned with controlling digital resource even after it has been delivered to a legitimate recipient [5].

DCON is formed beyond some well-known resource access, dissemination, and usage protection policies.

- *DCON* is an extension of traditional access control from the single server-side resource control to continuous resource access authentication, authorization and propagation along the dissemination path scatted over the decentralized and heterogeneous Internet environment [4, 15, 1, 9];
- *DCON* is an expansion of commercial digital rights management (*DRM*) that focuses on commercial copyrighted digital resource distribution by charging payment from recipients based on contracts subscribed in advance [6, 13]. Commercial *DRM* applications mostly concern the payment-based type (*PBT*) of resource dissemination but ignore the payment-free type (*PFT*). Our generic *DCON* model integrates the *PBT* and *PFT* type, and the new zero-payment type (*ZPT*) for universal resource dissemination applications.
- *DCON* is an enlargement of originator control (*ORCON*). *ORCON* is an access control policy that requires recipients to gain originators approval for resource re-dissemination [1, 9], but *DCON* breaks out this constraint and further, enriches the re-dissemination policies in dissemination chain context without security losing.

Based on the above discussion, *DCON* can be formally de ned as a security policy of controlling both digital resource access before distribution and resource usage even after distribution. The control scope of *DCON* described in the de nition indicates the *physical* resource distribution, and the *continuous* resource control along the dissemination chain.

Our contributions of this paper mainly focus on a formal analysis of *DCON* within the dynamic dissemination tree context, including (1) specifying rule-based automatic re-dissemination rights assignment and revocation (cascade or non-cascade); (2) supporting dynamic dissemination modelling with temporal activation and inactivation of dissemination policy; (3) purchasing a dynamic multi-policy *UCON* model based on the policy-compatible analysis and policy-con ict solutions.

Basic concepts related to resource dissemination are presented in session *2*. Two taxonomies of dissemination policy are described in session *3*. Multi-policy *DCON* and dynamic multi-policy *DCON* are proposed in session *4* and session *5* respectively. Finally, the importance of *DMDCON* within the usage control domain is discussed as a conclusion of the whole paper.

2 Basic Concepts

In most literature on resource dissemination (access, distribution, propagation, etc) [12, 13, 6, 5, 9, 3], some basic concepts such as dissemination certi cate, policy, chain, and resource dissemination decision, are introduced. For example, *DCON* means that the distributor or rights holder can control recipients' access to the digital information [12]. But there are no formal de nitions of these concepts to express their control domain boundaries and dynamical features. For the convenience of the latter extensive discussion, we need to formally de ne and extend those elements.

Firstly, the basic concept, *dissemination chain* can be identi ed as a dissemination path consisting of a sequential list of resource dissemination relations from an originator (or agent) to recipients, denoted as a sequence N_1-N_2-$\cdots$-N_k, where any pair, i.e. N_i-N_{i+1} ($i<k$) is a dissemination relation indicating recipient N_i dissemi-

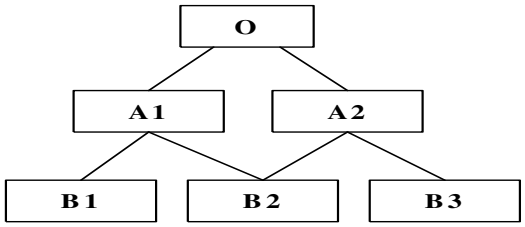


Figure 1. A dissemination tree T on resource $o1$

nates a resource to N_{i+1}. Mostly, a dissemination chain can form a linear-order lattice. Since the dissemination relation can be predened by host organizations, or existed after the resource has been disseminated, we can identify that there are two types of dissemination chain: *predened* chain and *existed* chain. The former indicates a chain is based on predened dissemination relations, which may not exist currently; The latter indicates all dissemination relations along a dissemination chain have been existed. Here we just concern the predened chain type from the view of modelling *DCON*.

Another, we introduce some new concepts elaborately for describing the detail dissemination control process. We define *re-dissemination certificate* as a label indicating whether a subject [1] has capability of re-disseminating resource to others. The *resource re-dissemination allowance* (*SRA*) function is a mapping that indicates whether a subject can re-disseminate specific resource to another one. This function can be formalized as, SRA: $C \times S \times S \times O \rightarrow \{true, false\}$, where C is a boolean parameter denotes whether a re-dissemination certificate is contained in the resource owner node or not by $true$ or $false$ value, S denotes subject set, O denotes object set (resource) and the return of the *SRA* function is a boolean value.

For example, $SRA(true, A1, B1, o1) = true$ denotes $A1$ has a re-dissemination certificate and disseminates $o1$ to $B1$. If the parameter C in the *SRA* function is *false*, then *SRA* always returns *false* no matter of other independent variables. Another, the *resource re-dissemination rights allowance*(*SRRA*) function is a mapping that returns a boolean value indicating whether a subject can grant the re-dissemination right on a specific resource to another subject. It can be formalized as, $SRRA$: $SRA \rightarrow true, false\}$, where *SRA* denotes the return value of a *SRA* function. In the above example, $SRRA(SRA(true, A1, B1, o1)) = true$ denotes $A1$ can grant $B1$ the re-dissemination right on $o1$. It should be noted that $SRRA(false)$ is always *false*.

Dissemination relation in the above dissemination chain definition is from a single subject to another one (one-to-one for short). For describing the one-to-many dissemination relation, we introduce the notion of dissemination tree, an extended version of dissemination chain. Informally, we can see a dissemination tree as the integration of several relative dissemination chains.

DEFINITION 1. Dissemination tree is a tree-shape resource dissemination structure integrated by several relative dissemination chains, where a node represents a subject and an edge between two related nodes represents a dissemination relation.

The root node of a dissemination tree always represents a resource originator (or agent), a dissemination path means a list of partial-order dissemination relations (represented as node pairs, said above) from the root to a specific node. A node is called the *parent* node if it disseminates a resource(or its re-dissemination right) to other nodes (called *son* nodes). The function $Parent$: $S \rightarrow \{S\}$, returns all parents of a node. In addition, we define a node's ancestors as all senior nodes from its parents up to the root node.

In *Fig*.1, root O is the ancestor of all nodes, $A1$ is the parent of both $B1$ and $B2$, $A2$ is the parent of both $B2$ and $B3$. In general, a dissemination tree expresses several dissemination chains or paths with different resource and different dissemination policies. Even, a dissemination tree can have more than one root, which can be called *dissemination network*(direct acyclic graph) or *multi-root* tree. Above all, the essence of a dissemination tree is to integrate various dissemination chains with some shared nodes (as node O and $B2$ in *Fig*.1), and to offer a larger environment for extended dissemination analysis than a single dissemination chain. Within dissemination tree environment, we can expose and solve the policy-conflict problems such as the *multi-parent* one, which is not discovered within single dissemination path but is real existed in many applications.

Similar to the dissemination chain category as described above, there are also two types of dissemination tree: *predened* and *existed* (dissemination) tree. (1) *Predened tree* is built on several related predened dissemination chains; (2) *existed tree* is built on related existed dissemination chains.

On scope, *predened* tree contains *existed* tree, because existing dissemination should have satisfied predened dissemination relations. A *predened* tree frames and restricts the resource (with re-dissemination rights) dissemination flow, but a *existed* tree only describes the current status. In this paper, all dissemination trees proposed are *predened* ones, since they help to analyze possible resource dissemination flows in an integrated way, and policy-conflict problems much fully than any other dissemination tree.

Given a dissemination tree T (*Fig*.1), node $A1$ requires node O for disseminating resource $o1$ and its re-dissemination right, the policy of dealing such a request is as follows:

- for the resource re-dissemination: if O has the re-dissemination certificate and $SRA(true, O, A1, o1) = true$, then $A1$ can acquire $o1$;
- for the resource re-dissemination right: if $A1$ can acquire $o1$ and $SRRA(SRA(true, O, A1, o1)) = true$, then $A1$ can acquire the re-dissemination right on $o1$.

In the above, resource (with its re-dissemination rights) dissemination decision on a node can be performed only by its single parent node [2]. This policy is called *parent-priority*, which indicates that the parent node decides whether its son node can obtain a resource(with its re-dissemination rights). As the decision made above, the single node, O can decide whether $A1$ acquire the re-dissemination right on resource $o1$. More generally, if there is a node $B1$, the son of $A1$, which wants to require the re-dissemination right on $o1$ from $A1$, the formal decision expression is like:
$SRRA(SRA(true, A1, B1, o1)) = true \rightarrow SRRA(SRA(true, O, A1, o1)) = true \cap SRA(true, A1, B1, o1) = true$.
If $B1$ obtains re-dissemination right on $o1$, $A1$ must own $o1$ and its corresponding re-dissemination right firstly.

[1] In this paper, subjects include originators and recipients.

[2] There is no consideration of payment or identification requirement for the resource dissemination charged by the senior nodes.

3 Policy

Different applications may adopt dynamically different resource dissemination policies. There may be many parent nodes of $A1$, or $B1$ in the above, which means that there may be some con icts existing in the dissemination tree. For example, supposing $A1$, $A2$ are two parent nodes of $B2$ (*Fig*.1), $SRRA(SRA(true, A1, B2, o1)) = true$ and $SRRA(SRA(true, A2, B2, o1)) = false$ hold, now how to judge whether $B2$ should have the re-dissemination right on $o1$? For answering this kind of *multi-parent* con ict problems, we propose three policy types of con ict-solution as follows.

- Positive policy. if $\exists\ Ai \in Parent(B2)$, satis es $SRA(true, Ai, B2, o1) = true$, then $B2$ can acquire $o1$ from Ai; if $SRRA(SRA(true, Ai, B2, o1))=true$, then $B2$ can acquire the re-dissemination right on $o1$ from Ai;
- Negative policy. if $\exists\ Ai \in Parent(B2)$, satis es $SRA(true, Ai, B2, o1) = false$, then $B2$ can not acquire $o1$ from any parent; if $SRRA(SRA(true, Ai, B2, o1))=false$, then $B2$ can not acquire the re-dissemination right on $o1$ from any parent;
- Majority-voting policy. if $\exists\ Ai \in Parent(B2)$, $i = 1, 2, \cdots, m$, and $m \geq major(|Parent(B2)|)$ [3], satis es $SRA(true, Ai, B2, o1) = true$, then $B2$ can acquire $o1$ from any positive parent [4]; if $SRRA(SRA(true, Ai, B2, o1)) = true$, then $B2$ can acquire re-dissemination rights on $o1$ from any positive parent.

Beside, dissemination policy can be divided into three types based on the dissemination purpose of the host node that deploys it: Zero-Payment Type (*ZPT*), Payment-Based Type (*PBT*) and Payment-Free Type (*PFT*). The purpose of *ZPT* (deployed by the host node) is to set no control over resource dissemination, and consequently any subject can acquire the resource no matter payment, identi - cation and any other conditions; The purpose of *PBT* is to make pro t for the host node through resource dissemination transactions, and consequently, a payment function is required for monitoring the payment charging process; The purpose of *PFT* is to control the resource dissemination within limited domains, and no payment contact (and payment function) is required but dissemination of resource should be appropriately restricted by using special access mechanisms. Here it should be noted that payment represents more than the notion of money. It can include money (mostly), contracts, agreements, etc. And also the e-payment mechanisms are various within different applications.

In conclusion, the taxonomies of dissemination policy on its priority and on its purpose are connected with each other. For example, *PBT* often takes the *positive* policy but *PFT* always takes the *negative* policy. These elaborate connections should be described system-speci cally.

4 Multi-policy DCON

4.1 De nition

In reality, subjects who require a resource may have different purposes and different ways on its usage and re-dissemination. It results in the complexity of resource dissemination policy management. A multi-policy *DCON* model (*MDCON*) is proposed for solving this problem with convenience.

[3] $major(|Parent(B2)|)$ returns a number that indexes the major part of the whole one.

[4] positive parent denotes a parent node with a policy of allowing resource dissemination.

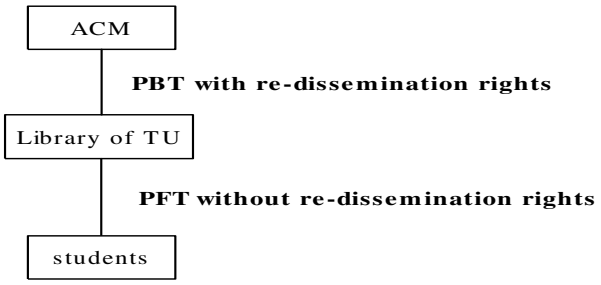


Figure 2. MDCON of PBT-PFT

DEFINITION 2. *MDCON* is a dissemination control model integrating multiple (more than one) dissemination policies within a dissemination tree (chain).

For example, as in the above *Fig*.2, the library of *Tsinghua* University (*TSU*) has brought *ACM*'s digital resource download service by money, and built a mirror site to store these resources, then legally, allows students on campus to share this service of accessing and downloading resource from the mirror site by the authentication on student-card *ID*. But students cannot share their acquired resource with any others. In this example, *ACM* is the originator and uses the *PBT* policy on resource dissemination. The library of *TSU* is a recipient that receives resources and corresponding re-dissemination rights from the originator by monetary contribution, and then takes the *PFT* policy on the resource sharing service oriented to students. But students can not propagate their owner resource any more since they have not re-dissemination rights from the library of *TSU*. In conclusion, there are two policies along the dissemination chain, *PBT* (with re-dissemination rights) and *PFT* (without re-dissemination rights).

4.2 Compatible vs. Con icting

Dissemination policies included in a dissemination tree are *policy-compatible* only if they can be real coexisted with reasonable application functions. For example, *PBT-PFT* is policy-compatible, since existing real applications (as the above example) that use *PBT* in a senior node and *PFT* in a junior node. Another, *ZPT-PFT* is not policy-compatible, since the second policy is meaningless and unnecessary, even illegally.

The policy-compatible analysis can introduce two kinds of compatibility relations: *before* and *after* relation, denoted by $beforeC$ and $afterC$ respectively, which are identi ed to describe the order-depended feature. Further, if $A1 \in beforeC(B1)$ holds, the dissemination relation $A1$-$B1$ is *upward-compatible*; if $B1 \in afterC(A1)$ holds, $A1$-$B1$ is *downward-compatible*; if both $A1 \in beforeC(B1)$ and $B1 \in afterC(A1)$ hold, $A1$-$B1$ is *full-compatible*. Mostly we only consider the full-compatible type and use the following constraint for normalizing $beforeC$ and $afterC$ relations :

- *If existing two policies, P, Q, and* $P \in beforC(Q)$*, then* $Q \in afterC(P)$*, and vice versa.*

From this point, we can de ne the following policy-compatible sets among the dissemination policies of different purposes (without interpretation) and conclude the dissemination policy compatibility theorem naturally. In reality, the general principles for normalizing its compatibility are as follows:
$beforeC(ZPT) = \{ZPT, PBT, PFT\}$;
$beforeC(PBT) = \{PBT, PFT\}$;

$beforeC(PFT) = \{PFT, PBT\}$;
$afterC(ZPT) = \{ZPT\}$;
$afterC(PBT) = \{PBT, PFT, ZPT\}$;
$afterC(PFT) = \{PFT, PBT, ZPT\}$.

THEOREM 1. In *MDCON*, two dissemination policies can *coexist* in a dissemination chain iff they are full-compatible.

[Proof Sketch]
(1) for the $\rightarrow$ direction, if two policies can coexisted in a dissemination chain, it indicates that this integration is meaningful for real *MDCON* application, so they are full-compatible according to the above interpretation;
(2) for the $\leftarrow$ direction, if two policies are full-compatible, it indicates that no con ict between them and so they can coexist in a dissemination chain.

In reality, building a dissemination model should consider the policy-con ict[5] solutions for administration convenience, since some newly inserted policies may be con icted with existing policies in a dissemination chain. We propose three policy-con ict solutions based on the policy priority.

- *senior-priority*, which indicates if two policies are con ict, then change the junior to be compatible with the senior;
- *junior-priority*, which indicates if two policies are con ict, then change the senior to be compatible with the junior;
- *senior-junior-priority*, which indicates the in-between policy should be changed to be compatible with both the senior and the junior policies.

4.3 Rule-based speci cation

The resource dissemination decision making in *DCON* and *MDCON* is complex and complicated. For easing this process, we build a set of rules for automatically making resource dissemination decisions. Now we use rules to specify dissemination policies, chains, trees, formally. For convenience, we don't consider the re-dissemination time point, policy taxonomy on priority.
-S: Subject set;
-O: Object set;
-P: Policy set, supposing $P = \{ZPT, PBT, PFT\}$;
-PF: $O \times S \times S \rightarrow P$, returns a policy over an object dissemination from a senior node to a junior one;
-$PBTDF : O \times S \times S \times Pay \rightarrow \{true, false\}$, denotes a dissemination allowance from a senior node to a junior one over a resource in PBT. Pay denotes a payment contract;
-$PBTDRF : O \times S \times S \times Pay \rightarrow \{true, false\}$, denotes a dissemination right allowance from a senior node to a junior one over a resource in PBT;
-$PFTDF$: $O \times S \times S \rightarrow \{true, false\}$, denotes a dissemination allowance from a senior node to a junior one over a resource in PFT;
-$PFTDRF : O \times S \times S \rightarrow \{true, false\}$, denotes a dissemination right allowance from a senior node to a junior one over a resource in PFT.

Now we take the above de nitions to describe the dissemination policies formally.

- For ZPT policy, $\forall\ s,\ r \in S,\ o \in O$, $PF(o,\ s,\ r) = ZPT \rightarrow SRA(true,\ s,\ r,\ o) = true$, $SSRA(SRA(true,\ s,\ r,\ o)) = true$;
- For PBT policy, $\forall\ s,\ r \in S,\ o \in O$, $PF(o,\ s,\ r) = PBT$, $PBTDF(o,\ s,\ r,\ \triangle)$ [6] $= true$, $PBTDRF(o,\ s,\ r,\ \triangle) = false$ $(true) \rightarrow SRA(true,\ s,\ r,\ o) = true$, $SSRA(SRA(true,\ s,r,\ o)) = false$ $(true)$;
- For PFT policy, $\forall\ s,\ r \in S,\ o \in O$, $PF(o,\ s,\ r) = PFT$, $PFTDF(o,\ s,\ r) = true$, $PFTDRF(o,\ s,\ r) = false$ $(true) \rightarrow SRA(true,\ s,\ r,\ o) = true$, $SSRA(SRA(true,\ s,\ r,\ o)) = false$ $(true)$.

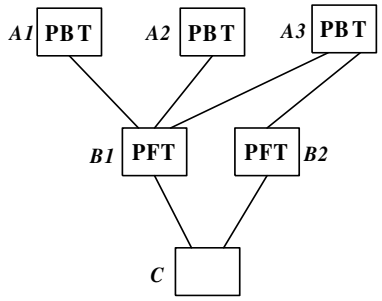


Figure 3. A dissemination tree

Any dissemination chain can be speci ed by combining the above rules. For example, A(PBT with re-dissemination rights according to contract c) - B(PFT without re-dissemination rights) - C can be speci ed as follows:

- On node pair $A(PBT)$-B, $PF(o, A, B) = PBT$, $PBTDF$ (o, A, B, c)=$true$, $PBTDRF(o, A, B, c)$=$true \rightarrow SRA(true, A, B, o) = true$, $SSRA(SRA(true, A, B, o)) = true$;
- On node pair $B(PFT)$-C, $PF(o, B, C)$=PFT, $PFTDF(o, B, C) = true$, $PFTDRF(o, B, C) = false \rightarrow SRA(true, B, C, o)$=$true$, $SSRA(SRA(true, B, C, o)) = false$.

Now we take another instance to demonstrate the dissemination tree speci cation. Supposing $\{A1, A2, A3\}$(PBT with re-dissemination rights propagation according to contract c) - $\{B1, B2\}$(PFT without re-dissemination rights propagation) - $\{C\}$ is a dissemination tree ($Fig.3$) for resource o, and the PBT policy uses the *positive* policy and the PFT policy uses the *negative* policy against the dissemination policy con ict. The dissemination ow of resource o in the whole dissemination tree can be speci ed as follows:

- On node pair $\{A1, A2, A3\}$(PBT & positive policy)-$B1$, If $\exists\ Ai, i \in \{1, 2, 3\}$, $PF(o, Ai, B1) = PBT$, $PBTDF(o, Ai, B1, c) = true$, $PBTDRF(o, Ai, B1, c) = true \rightarrow SRA(true, Ai, B1, o) = true$, $SSRA(SRA(true, Ai, B1, o)) = true$;
- On node pair $A3(PBT)$-$B2$, $PF(o, A3, B2) = PBT$, $PBTDF(o, A3, B2, c) = true$, $PBTDRF(o, A3, B2, c) = true \rightarrow SRA(true, A3, B2, o) = true$, $SSRA(SRA(true, A3, B2, o)) = true$;
- On node pair $\{B1, B2\}$(PFT & negative policy)-C, If $\forall\ Bi, i = 1, 2$; $PF(o, Bi, C) = PFT$, $PFTDF(o, Bi, C) = false \rightarrow \forall\ Bi, i = 1, 2$; $SRA\ (true, Bi, C, o) = false$, $SSRA(\ SRA(true, Bi, C, o)) = false$;

[5] policy-con ict denotes two policies are not full policy-compatible.

[6] $\triangle$ denote a concrete contract value of the corresponding parameter.

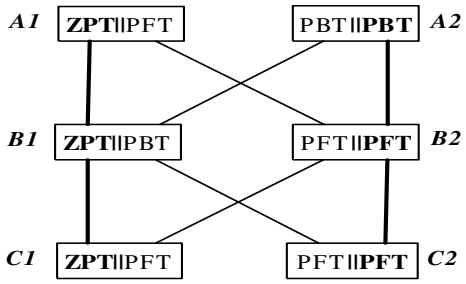


Figure 4. An instance of DMDCON

5 Dynamic Multi-policy DCON

5.1 De nition

In the above section, the dissemination structures (chains and trees) are all static and can not describe dynamical policy mutability, which is necessary and indispensable in real applications. So we propose the dynamic multi-policy *DCON* model (*DMDCON*) to specify continuous policy mutability.

Firstly, we de ne *dynamic dissemination modelling* as the dynamic process of updating the dissemination structure by some operations. From this de nition, some operations can be listed as dynamic dissemination modelling factors, which include inserting, appending a new node, removing an existing node, implementing new policies on existing or new nodes, updating or removing existed dissemination policies on existing nodes etc.

DEFINITION 3. *DMDCON* is an *DCON* model with the following features: (1) Allowing dynamic dissemination modelling; (2) Allowing more than one policy coexisted and activated in the same dissemination chain; (3) Using active time range constraints to express the dynamic features; (4) Using rules to specify the whole resource dissemination process.

The main purpose of *DMDCON* is to describe the dynamic and multi-policy nature of real *DCON* environment. Two attractive features of *DMDCON* are the dissemination decision continuity and dynamic resource dissemination by the activation and inactivation of relative dissemination policies.

5.2 Dynamic features of DMDCON

The dynamic features in our *DMDCON* context refer to the regular activation mutability of dissemination nodes. Factors resulting in this feature include temporal features and special attributes of recipients. For example, in some applications, a dissemination site is active just for some recipients who have special certi cates. Here we take the former factor to demonstrate the dynamic features.

Temporal features of *DMDCON* are expressed mainly by the notion of *active time range*, which is a period of time denoting policy activation and inactivation.

In a dissemination tree, a node is *available* if there is no senior/junior node that is policy-con ict with it. Formally, for node n, its *parent* node set S and *son* node set J, possible dissemination chain set Ch including node n. Node n is available only if, $\forall\, s \in S$, $j \in J$, $\exists\, ch \in Ch, < s, n > \in ch \rightarrow s \in beforeC(n) \cap n \in afterC(s)$; $\forall\, j \in J$, $\exists\, ch \in Ch, < n, j > \in ch \rightarrow j \in afterC(n) \cap n \in beforeC(j)$.

A policy p on node n is *active* if there exist a junior node j, the dissemination of resource o from n to j is available in the current time range tr. Supposing TR denotes time range, and $T(TR, S, S, O, P) \rightarrow \{true, false\}$ denotes an activation test function, $T(tr, A, B, o, p) = true$ denotes node A can disseminate resource o to node B within the time range tr through the active policy p. A node n is *active* only if existing a senior node s and a junior node j are both available, and in the current time range tr, $T(tr, s, n, o, p) = true \cap T(tr, n, j, o, p) = true$ holds. So that function T returns *true* should be a precondition for resource dissemination. Such as in the above example (*Fig*.1), given a dissemination tree T, node $O(p)$-$A1$ is a two-tier model for disseminating resource $o1$ (p is a policy on O). $A1$ can acquire $o1$ iff O has re-dissemination certi cate and $SRA(true, O, A1, o1) = true \cap T(tr, O, A1, o1, p)$ [7] $= true$; $A1$ can acquire the re-disseminate right on $o1$ iff $SRRA(SRA(true, O, A1, o1)) = true \cap T(tr, O, A1, o1, p) = true$.

Furthermore, we take a typical *DMDCON* instance, which contains four multi-policy dissemination chains (*Fig*.4), to demonstrate the above feature.
$A1(ZPT)$ - $B1(ZPT)$ - $C1(ZPT)$;
$A1(PFT)$ - $B2(PFT)$ - $C1(PFT)$;
$A2(PBT)$ - $B1(PBT)$ - $C2(PFT)$;
$A2(PBT)$ - $B2(PFT)$ - $C2(PFT)$.

We de ne that the active time range of an available dissemination chain is the intersection of all of policies within the chain. Supposing the active time range of the ZPT policy on node $A1$ to all junior nodes is (6:00-24:00), it of the PFT policy on $B2$ to $C1$ is (9:00-15:00), and it of the PFT policy on $C1$ to a junior node is (8:00-14:00), So we can calculate easily the active time range of the dissemination chain $A1(PFT)$-$B2(PFT)$-$C1(PFT)$ is (9:00-14:00).

5.3 Revocation: cascade vs. non-cascade

A policy is *revoked* or in the status of *revocation* if it is inactive. Considering the in uence by the revocation of an active policy in a dissemination tree, we divide it into two types:

- *Non-cascade*, which indicates the revocation of a policy just in uences the stand-alone node in a dissemination tree, but not all junior nodes;
- *Cascade*, which indicates the revocation of a policy results in all revocations of junior policies (nodes) in a dissemination tree, which have obtained resource from that node with that policy.

In another way, considering what should be in uenced by a policy revocation, there are two situations: on *resource* or on *re-dissemination rights*. The former indicates that if a policy is revoked, dissemination of both resource and re-dissemination rights [8] should be stopped immediately; the latter indicates what a revoked policy can in uence is re-dissemination rights granting but not resource propagation. In short, there are four types of policy revocation by the combination of the above two taxonomies:
(1) non-cascade revocation on re-dissemination rights;
(2) non-cascade revocation on resource dissemination;
(3) cascade revocation on re-dissemination rights;
(4) cascade revocation on resource dissemination.

[7] p is a speci c policy on O used in this example.

[8] since re-dissemination rights should based on resource dissemination. There are no re-dissemination rights can be granted without resource itself.

Every type has its application domains. For example, Intelligence community needs cascade revocation on resource dissemination: if a le among the community has some error, its copies should not be propagated again. *B2C* e-commerce needs non-cascade revocation on resource: if a senior sale agency has no store of a brand of goods, but a junior has, the senior should not prevent the junior's selling.

5.4 DCON within usage control

Recently, the notion of usage control (*UCON*) is proposed as a comprehensive security service of encompassing traditional access control, trust management, and digital rights management [10, 14]. $UCON_{ABC}$ model family is seen as a new approach for next generation information security solutions [11].

DCON is one of the generic and key concerns of *UCON*, which enables dissemination and re-dissemination outside of a closed system environment where central control authority such as central reference monitor is hard to control.

The *DMDCON* model greatly integrates and expresses the dynamic dissemination conditions in *UCON* with the special mechanisms of the temporal restrictions on dissemination based on active time range, and continuous dissemination management within predened dissemination tree existed in real applications generally. In addition, the *DMDCON* model take a set of rules to describe the complex dissemination decision, which sets a good example for automatized dissemination management within *UCON* model.

6 Conclusion

In this paper, we rstly de ne and extend some basic concepts related with resource dissemination, including dissemination chain, tree. Then, we propose dynamic dissemination modelling and based on this notion, we build the comprehensive *DMDCON* model, which has two attractive features, dissemination decision continuity and dynamic resource dissemination. Finally, we brie y discuss the importance of *DCON* within the usage control domain.

Further research of *DMDCON*, integrating the secure resource initiation and transmission mechanisms, can form our ongoing distributed security model, Secure Resource Management (*SRM*), which elaborately considers trust management [2, 8, 16] and privacy protection [17, 7] as well as the all above mechanisms for achieving a secure resource control (including dissemination and usage separately) lifecycle in open system.

7 References

[1] Abrams, Marshall, and etc. Generalized framework for access control: Towards prototyping the orcon policy. *Proceedings of the 14th National Computer Security Conference*, pages 257 266, 1991.

[2] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. *Proceeding of IEEE Conference on Security and Privacy. Oakland, CA.*, 1996.

[3] Dwork, Cynthia, and etc. The mathematics of information coding, extraction, and distribution. *The IMA Volumes in Mathematics and its applications*, 107:31 47, 1999.

[4] D. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed nist standard for role-based accesscontrol. *ACM Transactions on Information and System Security*, 4(3):224 274, 2001.

[5] R. K.Thomas and R. Sandhu. Towards a multi-dimensional characterization of dissemination control. *Proceedings of the 5th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'04)*, 2004.

[6] R. Lannella and P. Higgs. Driving content management with digital rights management. *IPR systems whitepaper series*, 2003.

[7] K. Lefevre, R. Agrawal, V. Ercegovac, and R. Ramakrishnan. Limiting disclosure in hippocratic databases. *Proceedings of the 30th VLDB conference, Toronto, Canada*, 2004.

[8] N. Li and J. C. Mitchell. Rt: a role-based trust-management framework. *In DARPA Information Survivability Conference and Exposition (DISCEX), Washington, D.C.*, 2003.

[9] J. Park and R. Sandhu. Originator control in usage control. *Proc. 3rd IEEE International Workshop on Policies for Distributed Systems and Networks, Monterey,California*, pages 60 66, 2002.

[10] J. Park and R. Sandhu. Towards usage control models: Beyond traditional access control. *In Proceedings of 7th ACM Symposium on Access Control Models and Technologies*, 2002.

[11] J. Park and R. Sandhu. The $ucon_{ABC}$ usage control model. *ACM Transactions on Information and System Security*, 7(1):128 174, 2004.

[12] J. Park, R. Sandhu, and J. Schifalacqua. Security architectures for controlled digital information dissemination. *IEEE*, 2000.

[13] Ryotuv and T. Neuman. The set and function approach to modeling authorization in distributed systems. *Proceedings of the Workshop on Mathematical Methods and Models and Architecture for Computer Networks Security*, 2001.

[14] R. Sandhu and J. Park. Usage control: A version for next generation access control. *Proc. Mathematical Methods, Models and Architectures for Computer Networks Security, Saint Petersburg, Russia*, 2003.

[15] R. Sandhu, R. S., and etc. Role-based access control models. *IEEE Computer*, 29(2):38 47, 1996.

[16] J.-M. Seigneur and C. D. Jensen. Trading privacy for trust. *iTrust2004, LNCS2995*, pages 93 107.

[17] J. won Byun, E. Bertino, and N. Li. Purpose-based access control of complex data for privacy protection. *SACMAT'05, Stockholm, Sweden*, Jun.

B-tree indexes for high update rates

Goetz Graefe

1 Abstract

In some applications, data capture dominates query processing. For example, monitoring moving objects often requires more insertions and updates than queries. Data gathering using automated sensors often exhibits this imbalance. More generally, indexing streams is considered an unsolved problem.

For those applications, B-tree indexes are good choices if some trade-off decisions are tilted towards optimization of updates rather than towards optimization of queries. This paper surveys some techniques that let B-trees sustain very high update rates, up to multiple orders of magnitude higher than traditional B-trees, at the expense of query processing performance. Not surprisingly, some of these techniques are reminiscent of those employed during index creation, index rebuild, etc., while other techniques are derived from well known technologies such as differential files and log-structured file systems.

2 Introduction

Some applications capture more data than they query them. For example, a fleet management system for a trucking or taxi company might record each vehicle's latest position more often than the vehicles' positions are queried by a fleet supervisor. In those cases, index and B-tree organization should be optimized for insertion and update performance rather than for query performance, as has been the traditional objective.

Another application domain for the techniques discussed in this survey is indexing of continuous data streams. Filtering streams on the fly is reasonably well understood, but streams that contain identifiers of real-world objects often need to be matched by identifier and descriptive attribute against static data as well as other streams. Thus, it is imperative that streams can be captured, typically in the order of data arrival, as well as indexed by attributes other than arrival time, sometimes in multiple indexes with multiple orders. For example, an incoming stream of credit card transactions might require, for efficient and near-instantaneous fraud detection, indexing by card number, customer identity or household (a customer might have lost multiple credit cards at the same time), and merchant (a dishonest employee might fraudulently charge credit cards from many customers).

In the following, we assume that update and insertion performance are more important than query performance. If the reader is not concerned about such applications, traditional B-tree optimizations should be applied rather than the techniques surveyed here. Moreover, we assume that any throttling of the workload, e.g., "best effort" recording of current vehicle locations, has already been applied, such that the remaining update requests indeed must be captured in all indexes under consideration. Finally, we assume that hardware assistance has been considered and exploited to the extent possible and appropriate, e.g., disk striping and solid-state disks or disk buffers.

3 I/O optimizations

As with most database operations, focusing on the efficiency of disk I/O is an effective means for improving performance and scalability. However, one must separate between improvements to the overall system throughput and improvements to the response time of individual transactions, which may or may not be tremendously interesting here.

There are several very generic performance improvement technologies, e.g., data compression [WKH 00]. In update-intensive workloads, relevant compression applies not only to the data but also to the transaction log. Suffice it here to point out that some compression techniques are surprisingly simple, e.g., truncating leading and trailing zeroes or blanks, and aggregating multiple log records from the same transaction into a single log record in order to save the overhead of many record headers in the transaction log.

3.1 Prefetch, read-ahead, and write-behind

Write-behind of log pages and of data pages are well known techniques. By itself,

write-behind does not improve system throughput, because the amount of writing does not decrease. However, write-behind often enables large writes, which is even more efficient that queued I/O. Moreover, they are helpful in the case of spikes in the workload and they permit additional optimizations. For example, modern disk drives support native command queuing and thus perform better if there are tens of I/O operations pending at all times [ADR 03].

Read-ahead (as commonly used in scans) does not apply to append operations, but it applies to merging an entire batch of modifications into an existing B-tree. When merging multiple B-tree partitions (discussed below) into one, read-ahead with forecasting can improve performance, as merging partitions is essentially the same problem as merging runs in an external merge sort [G 03a].

Prefetch based on individual keys apply not only to retrieval operations, e.g., navigation from a non-clustered index into a clustered index, but also to update operations. However, like read-ahead and write-behind, prefetch also does not directly improve system throughput or bandwidth, only response time or latency of individual operations, which might improve system throughput indirectly by reducing concurrency control contention.

3.2 Write-optimized B-trees

In addition to asynchronous I/O, dynamic placement of contents on disk can improve write performance [G 04]. This effect is well known and has been extensively studied for log-structured file systems [OD 89], in particular in the context of RAID storage [PGK 88]. The principal idea of write-optimized B-trees is to allocate a new location on disk each time a page is written to disk, and to do so as part of the write operation, i.e., subsequent to the buffer manager's replacement decision, and to allocate a page's new location in such a way that multiple concurrent write operations all target the same area on disk.

In order to avoid subsequent updates of neighboring pages, the traditional page chain using physical page identifiers is replaced by a logical page chain using separator keys, i.e., each page carries as lower and upper fences the separator key propagated to the page's parent node when the page was split from its neighbors. In addition to supporting the same consistency checks and other maintenance operations supported by traditional physical page chains, fence keys simplify and improve key range locking, because it is never required to navigate to a neighboring leaf page in order to find the right key to lock. After physical page chains have been replaced by logical fence keys, the only role for physical page identifiers is in child pointers, and only those have to be updated when a node moves to a new location on disk.

In traditional B-tree algorithms, a new location is allocated as part of the B-tree manager's decision to split a node, such that subsequent log records can refer to the page identifier. In write-optimized B-trees, a new page is given a temporary identifier that log records may refer to, and the page is moved as part of the write operation in a way very similar to a page move during B-tree defragmentation. Thus, proven concurrency control and recovery mechanisms apply.

The performance effect of write-optimized B-trees is such that random write operations are converted to large sequential write operations, with a bandwidth advantage of factor 10 or more, at the expense of added maintenance of each node's parent each time a node is written to a new location on disk.

4 Buffering insertions

There are multiple ways to buffer and group new insertions in order to modify each B-tree node less often, with the advantage of less disk I/O, fewer faults in the CPU cache, etc. Query operations either need to search the buffer structure in addition to the B-tree index or they force some or all buffered records into the B-tree index.

For correct transactional execution, both insertion and deletion in the buffer must be logged; thus the log volume in these methods may exceed the traditional log volume by a factor of three or more. However, only the initial insertion into the first buffer is a user transaction, whereas all subsequent movements of a record can be system transactions that can commit inexpensively without forcing the tail of the transaction log to stable storage.

4.1 Buffering within tree nodes

Several researchers have explored data structures and algorithms that add a large buffer to each interior tree node [A 96,

AHV 02, VSW 97]. Often the size of this buffer exceeds the size of the area dedicated to traditional key-pointer pairs, not only because each buffered new record is larger than a key-pointer pair but also because the number of retained records should be larger than the number of key-pointer pairs. When the buffer fills up, appropriate records are pushed down to the child with the most retained records.

It seems that records should be retained only for those children not immediately available in the I/O buffer. Given that most B-trees have a fan-out of 100 or more, and given that in most database servers the memory size exceeds 1% of the disk size, and given that the B-trees discussed here are among the most active and performance-critical indexes within the database, one may infer that such buffering applies only at the nodes immediately above the leaves. In other words, there may be additional improvement possible beyond published methods that permit buffering in nodes of all B-tree levels.

4.2 Buffering in separate structures

An alternative to buffering insertions in tree nodes is to create a separate data structure to buffer new insertions [LJB 95, MOP 00, OCG 96]. This data structure can be another B-tree or it can be a different type of in-memory data structure, e.g., a hash table. In fact, it can also be a collection of data structures, forming a hierarchy or cascade of staging areas. Interestingly, this organization is reminiscent both of generational garbage collection [U 84].

New structures imply new mechanisms for concurrency control and recovery. Thus, a standard index structure that is already implemented might be the preferred mechanism. Otherwise, new locking modes or protocols require correctness arguments, implementation, testing, etc. Perhaps the most desirable implementation avoids both separate structures and modifications of existing structures, and instead only uses existing mechanisms in different ways.

4.3 Buffering in B-tree partitions

One design motivated by the desire to avoid special-case code employs the main B-tree as its own buffer data structure by introducing partitions within each B-tree [G 03a]. By introducing an artificial leading key column, the traditional B-tree structure is retained. The "main" B-tree is defined by a common value for the artificial leading key column, say 0 or null, and one or more "buffers" are defined by different values in that column, say 1, 2, etc.

Traditional buffer management together with a size limit on newly added partitions can ensure that data insertions by user transactions can be absorbed entirely in memory. In the extreme case, partitions of new insertions can be as small as a single record, i.e., each new insertion defines a new partition and can thus proceed with hardly any search or page reorganization within the B-tree. Thus, insertion rates and throughput by user transactions are maximized, at the expense of more effort for index optimization and reorganization.

Queries have to search in each partition, using traditional methods for queries that restrict some index columns but not the leading one [LJB 95], possibly augmented with optimizations to exploit the fact that successive integer values are used as partition identifiers. Alternatively, query activities may force some merge activities, executed prior to actual data retrieval and implemented using system transactions. Thus, B-tree maintenance work that traditionally is part of update operations is shifted to query operations or reorganization that may happen any time between insertion and query. In the extreme case, a query may force complete merging and optimization of all partitions, maybe excepting one partition targeted by current insertions.

Some interesting aspects of such B-trees are (i) that the reorganization operation that combines multiple partitions into one is very similar to a merge step in a traditional external merge sort, (ii) that such merge operations can execute as system transactions and commit a very small key range at a time, (iii) that merge and reorganization operations can pause and resume at any time in response to load spikes etc., and (iv) the same technique can aid bulk deletions, i.e., B-tree entries to be deleted are moved by small system transactions into one dedicated partition and then deleted in one fast user transaction that cuts multiple full pages from the B-tree.

4.4 Graceful degradation

In addition to raw performance improvements, buffering insertions also enables

graceful degradation after errors in cardinality estimation during query optimization. Today, query optimization can choose between row-by-row update processing and index-by-index update processing. Updating row-by-row implies maintenance of all appropriate indexes immediately for each row. Updating index-by-index means that all changes are applied to one index at a time, possibly after splitting each update into a deletion and an insertion, sorting the changes on the index key, and recombining changes if appropriate; a generalized version of techniques described in [GKK 01] implemented in Microsoft SQL Server since release 7.0. Row-by-row updates are most appropriate for small changes, e.g., in online transaction processing, whereas index-by-index updates are more efficient for large updates, in particular if there are more individual changes than leaf pages in an index, e.g., in bulk insertion or bulk deletion. For graceful degradation, a query execution plan may prescribe row-by-row update processing due to an anticipated small update set, yet the actual execution may determine that the update set is rather large and switch to index-by-index updates.

Buffering insertions as described above using partitioned B-trees is a third way to apply updates to a B-tree index, and it thus opens up another option for graceful degradation. Row-by-row processing targeting a new partition promises I/O pattern and efficiency better than index-by-index processing, albeit with the disadvantage of non-optimal indexes left behind. For graceful degradation, an update plan may apply updates row-by-row in the main partition until the actual size of the update set becomes apparent and then switch to buffered or partitioned updates. While it is possible to implement graceful degradation from row-by-row to index-by-index updates using conditional execution in a traditional query execution plan, assigning a new partition identifier (artificial leading key column) to index changes is much simpler and it promises even faster update performance.

5 Differential files and indexes

While the designs discussed in the prior section are able to buffer insertions, they cannot buffer other update operations, i.e., modifications or deletions. However, they can be extended to do so, by adapting ideas from differential files [SL 76] to B-tree indexes. Interestingly, some B-tree adaptations for multi-version concurrency control and for historical indexes are very similar, including the logic required during query processing.

The basic approach is to append records that invalidate prior records without actually modifying those prior records. In an update, a new record supersedes the prior B-tree entry with the same key. In a deletion, the newly appended record simply indicates the end of the history for a particular key, or at least the end of the history until a subsequent new insertion with the same key.

Query evaluation needs to search the history for each particular key, either for the most current state (for traditional query semantics) or for the state at a particular time (for point-in-time historical queries). Merge operations may condense the history of keys depending on the desired future query capabilities.

In other words, like buffering insertions, buffering updates and deletions in differential B-trees trades query performance in favor of update performance. Turning random single-record insertions, deletions, and updates into append operations with large sequential write operations promises to improve the sustained update throughput by two orders of magnitude.

Of course, there is also a relationship between differential files and the implementation of multi-version snapshot isolation. The main difference, however, is that differential files retain the oldest version plus the deltas forward in time, whereas implementations of multi-version snapshot isolation are typically tuned for access to the most recent versions, i.e., they usually retain the most recent version plus deltas backward in time.

6 Transaction guarantees

Another opportunity for performance improvement may be to weaken transactional guarantees for some indexes, in particular for redundant non-clustered indexes. We consider three techniques that do so, one that dilutes the separation of individual transactions by batching, one that weakens guarantees in case of system failures, and one that records changes only in the transaction log without even attempting to apply them to the index, with the implicit danger that the attempt to apply such changes later might fail. Obvi-

ously, these techniques apply only if the remaining transactional guarantees are still strong enough for the application at hand.

6.1 Log-only operations

If the index maintenance cannot keep up with the update stream, maybe at least the transaction log can. In that case, one could write logical *redo* records to the transaction log and apply them later, essentially using *redo* recovery. Of course, this process violates multiple traditional assumptions about logging, e.g., that *redo* operations are always physical operations that already happened, that *redo* operations cannot fail, etc. However, depending on the application, such failures might not be total disasters and could be ignored, for example, when some individual location reports in a vehicle tracking application cannot be recorded in the historical index. Clearly, this idea might apply, but details need to be worked out, e.g., what transaction commit truly promises and what it guarantees, how checkpoints work and what they guarantee, etc.

6.2 Non-logged B-trees

Some database systems employ special techniques during index creation such that the contents of the new index do not appear in the transaction log. Instead, only catalog changes and page allocation are logged. Failure during index creation results in deallocation of those pages and erasure of the new index in the catalogs. Index creation ends with flushing all newly allocated and filled pages to disk, and subsequent backup operations of the database or even of the transaction log capture those new pages. Subsequent user transactions log their changes to the new index in the usual way.

This idea can be extended in the following way. If an index is truly redundant similar to a traditional cache, and if erasing the index during media or system recovery is acceptable, then all operations on this index may be non-logged, i.e., only space allocation is logged. This specifically includes user transactions running after index creation is complete. Rollback of a user transaction is driven by virtual log records attached to the transaction descriptor in memory, similar to virtual log records used in other transaction processing designs [G 04, GK 85]. Details of this technique have not been published at this point, but the technique seems promising for some applications, in particular for temporary caches and for indexes that exist only in memory.

6.3 Batching updates

Finally, one may group multiple update operations and transactions into a single transaction. However, it seems important to separate the transaction semantics from the data structure. For example, many small user transactions may all insert into a single buffer as described above, leaving it to a subsequent system transaction (or series of small system transactions) to merge such insertions into the main B-tree. In other words, it might not be necessary or advantageous to modify or weaken the boundaries and semantics of user transactions in order to achieve the desired advantages in performance and scalability.

7 Summary and conclusions

In summary, if one is willing to accept deterioration of query performance by an order of magnitude, e.g., due to searching multiple partitions, update and insertion performance can be improved by two orders of magnitude or more, e.g., by turning insertions into append operations and by turning random in-place writes into large sequential writes to newly allocated disk space. Less dramatic tradeoffs also exist. While most applications issue more queries than update requests and thus demand a query-optimized database organization, some applications (e.g., tracking moving objects) record more data changes than they answer queries (e.g., about current object location). For those applications, numerous techniques are readily available for implementation by database vendors. Some are even available to database users, e.g., by introducing an artificial leading key column in the visible database schema and exploiting it for index creation and possibly for index maintenance during bulk operations [G 03b].

This survey attempts to list a variety of possible techniques. New techniques include write-optimized B-trees, partitioned B-trees using partitions to buffer insertions or all modifications in the manner of differential files, and non-logged B-trees. However, this intuitive

appraisal requires validation using prototyping or even product implementations.

Numerous open questions present themselves, including the question for additional or better trade-offs between update and query performance, a comparative performance evaluation of the methods described above based on an appropriate benchmark, adaptation of the techniques discussed above to other index structures, in particular to multi-dimensional indexes such as UB-trees and R-trees and to materialized and indexed views, and integration of query and update processing with database maintenance operations such as consistency checks, defragmentation, and statistics refresh for query optimization. Maybe the present survey will stimulate and structure such research.

8 Acknowledgments

Theo Härder and Bernhard Mitschang read earlier incomplete drafts and contributed multiple very helpful suggestions.

9 References

[A 96] Lars Arge: Efficient External-Memory Data Structures and Applications. University of Aarhus (Denmark), 1996.

[ADR 03] Dave Anderson, Jim Dykes, Erik Riedel: More Than an Interface - SCSI vs. ATA. Conference on File and Storage Technology (FAST), March 2003.

[AHV 02] Lars Arge, Klaus Hinrichs, Jan Vahrenhold, Jeffrey Scott Vitter: Efficient Bulk Operations on Dynamic R-Trees. Algorithmica 33(1): 104-128 (2002).

[G 03a] Goetz Graefe: Sorting and Indexing with Partitioned B-Trees. Conference on Innovative Data Systems Research, 2003.

[G 03b] Goetz Graefe: Partitioned B-trees - a user's guide. Datenbanksysteme für Business, Technologie und Web (BTW) 2003: 668-671.

[G 04] Goetz Graefe: Write-Optimized B-Trees. VLDB Conference 2004: 672-683.

[GK 85] Dieter Gawlick, David Kinkade: Varieties of Concurrency Control in IMS/VS Fast Path. IEEE Data Eng. Bulletin 8(2): 3-10 (1985).

[GKK 01] Andreas Gärtner, Alfons Kemper, Donald Kossmann, Bernhard Zeller: Efficient Bulk Deletes in Relational Databases. IEEE ICDE 2001: 183-192.

[JNS 97] H. V. Jagadish, P. P. S. Narayan, S. Seshadri, S. Sudarshan, Rama Kanneganti: Incremental Organization for Data Recording and Warehousing. VLDB Conference 1997: 16-25

[LJB 95] Harry Leslie, Rohit Jain, Dave Birdsall, Hedieh Yaghmai: Efficient Search of Multi-Dimensional B-Trees. VLDB Conference 1995: 710-719.

[MOP 00] Peter Muth, Patrick E. O'Neil, Achim Pick, Gerhard Weikum: The LHAM Log-Structured History Data Access Method. VLDB J. 8(3-4): 199-221 (2000).

[OCG 96] Patrick E. O'Neil, Edward Cheng, Dieter Gawlick, Elizabeth J. O'Neil: The Log-Structured Merge-Tree (LSM-Tree). Acta Inf. 33(4): 351-385 (1996).

[OD 89] John K. Ousterhout, Fred Douglis: Beating the I/O Bottleneck: A Case for Log-Structured File Systems. Operating Systems Review 23(1): 11-28 (1989).

[PGK 88] David A. Patterson, Garth A. Gibson, Randy H. Katz: A Case for Redundant Arrays of Inexpensive Disks (RAID). ACM SIGMOD Conference 1988: 109-116.

[SL 76] Dennis G. Severance, Guy M. Lohman: Differential Files: Their Application to the Maintenance of Large Databases. ACM Trans. Database Syst. 1(3): 256-267 (1976).

[U 84] David Ungar: Generation Scavenging: A Non-Disruptive High Performance Storage Reclamation Algorithm. Software Development Environments (SDE), ACM SIGPLAN Notices 19(5): 157-167 (1984).

[VSW 97] Jochen Van den Bercken, Bernhard Seeger, Peter Widmayer: A Generic Approach to Bulk Loading Multidimensional Index Structures. VLDB Conference 1997: 406-415.

[WKH 00] Till Westmann, Donald Kossmann, Sven Helmer, Guido Moerkotte: The Implementation and Performance of Compressed Databases. ACM SIGMOD Record 29(3): 55-67 (2000).

Report on the 10th International Symposium on Database Programming Languages (DBPL 2005)

Gavin Bierman
Microsoft Research Cambridge, UK

Christoph Koch
Saarland University, Germany

DBPL 2005 was held on August 28–29, 2005, in the charming surroundings of Trondheim, Norway, and was one of the eleven meetings that were co-located with VLDB. DBPL meets every two years and presents the very best work at the intersection of database and programming language research. DBPL 2005 is the tenth symposium in the series.

The program committee selected 17 papers from a total of 63 submissions—an acceptance rate of 27%—and invited Giuseppe Castagna to open the symposium with a lecture. The 18 talks were given over two days, and informal proceedings were distributed at the meeting. In DBPL tradition, the authors of accepted papers were able to improve their papers following discussions and feedback at the symposium, and these papers have been collected and are published by Springer in volume 3774 of the LNCS series.

Invited lecture Giuseppe Castagna opened DBPL 2005 with an invited lecture entitled *Patterns and Types for Querying XML Documents*. During this lecture he surveyed the various approaches for deconstructing XML data from both the database and programming language communities. He identified two main categories: the vertical approach embodied in database query languages such as XPath; and the horizontal approach as found in programming languages such as ℂDuce (a language which he helped design). As the latter is less well-known to the database community, he gave a concise but detailed tutorial on regular expression patterns; a key feature of ℂDuce. His conclusion was that future research should be directed towards developing languages with a unification, or tight integration, of both approaches.

Research papers

The first paper by Claus Brabrand, Anders Møller and Michael Schwartzbach, *Dual Syntax for XML Languages*, describes XSugar which is a system for managing dual syntax for XML languages. Given a specification, the system can both translate between the syntactic alternatives, and check that the transformations are reversible and valid.

The second paper by J. Nathan Foster, Michael Greenwald, Christian Kirkegaard, Benjamin Pierce and Alan Schmitt, *Exploiting Schemas in Data Synchronization*, concerns the authors' synchronization framework, Harmony, which can generate state-based synchronizers for a variety of tree-structured data formats. The authors formalize the synchronization algorithm and show how the synchronization process is driven by the schema of the structures involved.

The third paper by Benny Kimelfeld and Yehoshua Sagiv, *Efficiently Enumerating Results of Keyword Search*, considers the keyword search problem. The authors identify a common class of algorithms and show that they are provably efficient, i.e. they run with polynomial delay.

The next paper by Dario Colazzo and Carlo Sartiani, *Mapping Maintenance in XML P2P Databases*, considers P2P systems where peers manage their own data and where schema mappings exist between peers. The authors consider the problem of corrupted mappings and

provide a technique for maintaining schema mappings based on a semantic notion of correctness.

The paper by Diego Calvanese, Giuseppe de Giacomo, Domenico Lembo, Maurizion Lenzerini and Riccardo Rosati, *Inconsitency Tolerance in P2P Data Integration: an Epistemic Logic Approach*, also considers unstructured P2P systems. The authors extend their previously introduced multimodal epistemic semantics for such systems and study the problem of dealing with inconsistencies in such P2P data integration scenarios.

XML Data Integration with Identification by Antonella Poggi and Serge Abiteboul considers the problem of data integration of XML data where two major issues arise: first, that the global schema can be expressed as a constraint set, and secondly, the complications of node identity. The authors provide a formal framework and consider various problems including globally identifying nodes and answering queries under different mapping assumptions.

The paper by Floris Geerts and Wenfei Fan, *Satisfiability of XPath Queries with Sibling Axes*, considers the impact of sibling axes to the satisfiability problem for XPath fragments. The authors show that in many cases the presence of sibling axes significantly complicates the satisfiability analyses and give bounds for a number of XPath fragments with and without axes.

XML Subtree Queries: Specification and Composition by Michael Benedikt and Irini Fundulaki considers the problem of queries that filter an input document and return a subdocument. Such queries are useful but cannot be naturally specified in either XPath or XQuery. The authors identify a fragment of XPath with an alternative subtree selection semantics and study the query composition problem.

The paper by Jan Hidders, Stefania Marrara, Jan Paredaens and Roel Vercammen, *On the Expressive Power of XQuery Fragments*, provides a broad analysis of the expressive power of various fragments of XQuery. The authors identify 64 different XQuery fragments and classify them into 17 equivalence classes such that two fragments can express the same fragments iff they are in the same equivalence class.

Peter Thiemann's paper, *A Type Safe DOM API*, considers the W3C recommended, language neutral, API for XML document manipulation, DOM. The assumed type system for DOM is quite simple and a number of constraints are not made explicit at the type level. The author proposes a refinement of the Java type system that makes these constraints explicit and subject to compile-time checking.

The paper by Michael Levin and Benjamin Pierce, *Type-based Optimization for Regular Patterns*, considers pattern matching mechanisms based on regular expressions (such as those described in the invited lecture). The authors consider the problem of compiling pattern matching and propose a method that utilizes the schema of the input value to generate efficient code. They show that generating optimal code is decidable for finite patterns.

The paper by Giorgio Busatto, Markus Lohrey and Sebastian Maneth, *Efficient Memory Representation of XML Documents*, addresses the problem that many query processors use memory to represent XML data whose size far exceeds the size of the XML document itself. The authors present a way of compressing XML trees such that basic tree operations, such as edge traversal, are preserved in the compressed representation.

The paper by Joachim Niehren, Laurent Planque, Jean-Marc Talbot and Sophie Tison, *N-ary Queries by Tree Automata*, generalizes previous work on node-selecting tree automata, and proposes and studies notions of tree automata that select tuples of nodes from trees and their power to express n-ary queries, for both ranked and unranked trees.

The paper by Wim Martens and Joachim Niehren, *Minimizing Tree Automata for Unranked Trees*, considers the problem of efficiently minimizing automata for unranked trees (such as those that form a foundation of XML schema and various query and pattern languages). They show that, surprisingly, unranked tree automata contribute an additional

form of nondeterminism that renders minimial unranked tree automata in general not unique. The paper studies the complexity of minimization and introduces a new model of tree automata that may be of interest in its own right.

Solmaz Kolahi's paper, *Dependency-Preserving Normalization of Relational and XML Data*, considers the familiar problem of normalization for both relational and XML data. First, techniques from information theory are introduced to characterize the amount of redundancy present in 3NF schemas. Then the problem of preserving dependencies while eliminating redundancies is studied for XML and a new normal form is introduced.

The final two papers, *Complexity and Approximation of Fixing Numerical Attributes in Databases Under Integrity Constraints* by Leopoldo Bertossi, Loreto Bravo, Enrico Franconi and Andrei Lopatenko and *Consistent Query Answers on Numerical Databases under Aggregate Constraints* by Sergio Flesca, Filippo Furfaro and Francesco Parisi, both address the consistent query answering problem, i.e., the problem of extracting all consistent answers (w.r.t. a set of integrity constraints) from a database possibly violating these integrity constraints. The first paper provides several interesting complexity results for the general problem while the second introduces and studies the problem under aggregate constraints which are inequalities of queries involving aggregate-sum constructs.

Panel discussion DBPL 2005 closed with a panel discussion that was jointly organized with the co-chairs of The Third International XML Database Symposium (XSym 2005), Ela Hunt and Zachary Ives; and to which the attendees of both symposia were invited. The invited panel discussed *Whither XML, c. 2005?*, and consisted of experts on various aspects of XML: Gavin Bierman (Microsoft Research), Peter Buneman (University of Edinburgh), Dana Florescu (Oracle), H.V. Jagadish (University of Michigan) and Jayavel Shanmugasundaram (Cornell University).

Acknowledgements We owe thanks to a large number of people for making DBPL 2005 such a great success. We are grateful to the hard work and diligence of the 21 distinguished researchers who served on the program committee. We also thank Peter Buneman, Georg Lausen and Dan Suciu, who offered us much assistance and sound counsel. Svein Erik Bratsberg provided flawless local organization. Finally, DBPL 2005 was generously supported by Microsoft Research.

The WS-DAI Family of Specifications for Web Service Data Access and Integration

Mario Antonioletti,
Amy Krause
EPCC, University of Edinburgh,
Edinburgh EH9 3JZ, UK
(mario, amrey)@epcc.ed.ac.uk

Norman W. Paton
School of Computer Science,
University of Manchester,
Manchester M13 9PL, UK
norm@cs.manchester.ac.uk

Andrew Eisenberg
IBM, Westford, MA 01886
andrew.eisenberg@us.ibm.com

Simon Laws
IBM, Hursley Park
Winchester, SO21 2JN, UK
simon_laws@uk.ibm.com

Susan Malaika
IBM, San Jose, CA 95141, USA
malaika@us.ibm.com

Jim Melton
Oracle Corp., Sandy, UT 84093
jim.melton@acm.org

Dave Pearson
Oracle Corp,Thames Valley Park,
Reading, RG6 1RA, UK
Dave.Pearson@oracle.com

Guest Column Introduction

This month, we are pleased to provide to our readers a column that addresses an important aspect of grid computing: data access.

Grid computing is important and relevant because it provides middleware that supports secure and managed sharing of networked computational resources. This is valuable because many activities involve collaborations that stand to benefit from more efficient and systematic access to computational and data resources across management domains. The GGF is important because the development of grids depends on shared abstractions and consistent interfaces; the GGF is the principal standards body for grid computing. For the most part, the GGF is developing web service standards for resource management and use that can be used relatively independently or as part of a wider service-based architecture.

The authors of the note belong to two overlapping groups: the chairs of the GGF Data Access and Integration Services Working Group, and the members of the Design Team that was responsible for writing the specifications, and for evolving them in the light of input from other members of the working group and the wider community.

Introduction

The WS-DAI (Web Service Data Access and Integration) family of specifications defines web service interfaces to data resources, such as relational or XML databases. The specifications have been developed by the Database Access and Integration Services Working Group [1] of the Global Grid Forum [2], and can be used independently, or as part of a wider service-based grid architecture. The specifications include properties that can be used to describe a data service or the resource to which access is being provided, and define message patterns that support access to (query and update) data resources. The specifications include a data model-independent specification (WS-DAI), which is extended in two *realizations* to include model-dependent properties and operations in relational (WS-DAIR) and XML (WS-DAIX) specifications. It is anticipated that further realizations will be developed in due course, for example to support access to RDF and object databases.

The specifications do not provide fully transparent access to existing resources; various messages convey requests written using existing query languages, and the specifications do not impose any requirements for implementers to parse such language statements. Thus, for example, the WS-DAIR and WS-DAIX specifications implement similar message patterns and share the properties defined in WS-DAI, but users of the relational specification express requests in SQL and users of the XML specifications express requests in XPath, XQuery (http://ww.w3.org/CR/) or XUpdate (http://xmldb.org.sourceforge.net/xupdate).

The WS-DAI specifications define web services; as such, they are described using WSDL, and messages are sent to WS-DAI services using SOAP. The specifications have few additional dependencies on web service standards, although they have been designed to be able to be used in

conjunction with emerging standards from the grid and web services communities. In particular, the WS-DAI specifications are part of a wide ranging activity to develop the Open Grid Services Architecture (OGSA) [3] within the GGF. Data can be expected to exist in many forms in grids [4]; file access, movement and replication are central to many grid applications, and database systems are widely deployed in grids both for managing application data and for storing information of relevance to the middleware itself [5].

Many of the other web service specifications for security (e.g. WS-Security [6]), resource description and identification (e.g. the WS-Resource Framework [7]), and transactions (e.g. WS-AtomicTransaction [8]) are likely to be used extensively within service-oriented grids. The current state of play with OGSA is that the first version of a top-level architecture has been defined [3], and working groups are developing specifications that fit into the architecture. The WS-DAI specifications [9][10][11] form part of the OGSA Data Architecture, which will also include services for data movement, replication and storage management.

WS-DAI: A Framework for Data Access and Integration

The WS-DAI specification defines concepts, properties and messages that can be used in the definition of data services. A *data service* is a web service that implements one or more of the DAIS-WG specified interfaces to provide access to data resources. A *consumer* is an application that exploits the interface provided by a data service to access a data resource.

In principle, a *data resource* represents any system that can act as a source or sink of data. In practice, two kinds of data resource are distinguished in WS-DAI. An *externally managed data resource* is one in which the data is stored using an existing data management system. For example, a relational database of protein sequences stored using an installation of MySQL would be an externally managed data resource. An externally managed data resource:

1. Normally has an existence outside the DAIS service.
2. Has its lifetime managed in ways that are not specified in the DAIS specifications.

The DAIS-WG specifications do not provide operations for carrying out database administration functions on a database management system; instead they provide access to data managed by such systems using the capabilities of a service-based middleware. We note that the specifications are silent about how a service is implemented, so different properties and operations can be supported in different ways. For example, an *external data management system* could be a centralized database or a federation, and a federation could be constructed using WS-DAI services to access the federated resources.

A Service Managed Data Resource, by contrast:

1. Does not normally have an existence outside the service-oriented middleware.
2. Has its lifetime managed in ways that are specified in the DAIS specifications.

For example, the result of a query over a protein sequence database could give rise to a result set that contains information on all the proteins found in yeast. Such a result set could be made available as a data resource in its own right through a data service. As such a data resource has been created by a data service, the service is considered to have responsibility for providing access to the data resource and for enabling the destruction of the resource when it is no longer required; messages are provided to support such capabilities.

Following the OGSA naming scheme [3], data resources may be identified using *abstract names* or *concrete names*. An *abstract name* is a location-independent persistent name, which is a URI. A *concrete name* specifies the location of a data resource. As such, a single abstract name may be used to refer to a data resource accessible through multiple data services, but the concrete name is specific to the location of the service through which the data resource is being accessed.

In WS-DAI, a concrete name may consist of a combination of a service address plus an abstract name, or it may consist of a WS-Addressing endpoint reference as used in the WS-Resource Framework (WSRF) [7]. The WSRF specifies both an approach to resource identification in which resources are identified explicitly in the header of a message, and a range of associated behaviors, for example for representing the properties or the lifetime of a resource. In WS-DAI, messages to a data service must include within their SOAP body the abstract name of the resource being accessed by way of the service; where WSRF is being used to identify a data resource, the address to which a message is sent contains enough information to identify the resource within the data service, and the abstract name is also included in the SOAP body only for consistency with non-WSRF data resources. Both approaches are supported because the WSRF family of specifications is only now completing its standardisation process within OASIS (http://www.oasis-open.org/), and the level of adoption it will experience is not yet clear,

partly due to the presence of competing specifications for resource representation.

The properties of a data service and the principal messages that can be sent to a data service form the core of the specification, and are available whether or not data resources are represented using WSRF. Where data resources are represented using WSRF, the functionalities associated with WSRF for soft state lifetime management and fine-grained property access become available for managing the lifetime of (service managed) data resources and for accessing the properties defined in WS-DAI and in the realizations. For non-WSRF resources, properties are only available through retrieval of the entire property document, and lifetime management is restricted to an explicit destruction of the data service-data resource relationship.

Properties

It is important that a consumer of a data service can interrogate the service to: (i) determine whether the service is suitable for use in a given setting; and (ii) obtain the information required to enable valid requests to be sent to the service. To support this, the WS-DAI specification defines a collection of properties and provides a *GetDataResourceProperty-Document* operation, which, given a resource name, returns an XML document that describes all the properties associated with a resource on a service. If the data resource is represented using WSRF, the fine-grained operations provided by WSRF can be used to access the properties individually.

The properties in the WS-DAI specification are applicable to any type of data resource; individual realizations extend the properties listed here to include paradigm-specific features, such as the XML Schemas associated with the collections in an XML database. The following properties are defined for all data services:

DataResourceAbstractName: URI representing the abstract name of the data resource.

ParentDataResource: If this resource was derived from another, this has the abstract name of the parent data resource.

DataResourceManagement: An enumeration indicating if the data resource is *ServiceManaged* or *ExternallyManaged*.

DatasetMap: A mapping between the QName of a message and the URI of a dataset type representing the result types supported for the messages. For example, as there are many different XML representations for relational result sets, this property allows a service provider to indicate to a consumer the representations that the service can return. A consumer can then specify the format in which data should be returned to them. The dataset URIs are not defined by DAIS.

ConfigurationMap: A mapping between the QName of a message and the URI of an expression language. For example, a message which supports XPath queries may support XPath Version 1.0 expressions or XPath Version 2.0 expressions. DAIS does not define new expression languages but wraps those defined by others in DAIS messages. Expression languages are expected to evolve independently of the DAIS specifications. For example, the development of the DAIS specifications has no impact on the further development of the SQL or XQuery specifications. DAIS uses, but does not define, URIs to identify supported expression languages.

Suitably precise identifiers for languages do not always exist. In this regard, it is hoped that vendors and language standards bodies will develop schemes that allow the language supported by a data resource management system to be unambiguously identified.

LanguageMap: A mapping between the QName of a message and the URI of an expression language. For example, a message which supports SQL queries may allow such queries as SQL:1999 expressions or SQL:2003 expressions. DAIS does not define the URIs of supported languages.

DataResourceDescription: A human readable description of a data resource.

Readable and *Writeable:* These properties indicate whether or not the data resource is able to be read from or written to from the data service.

ConcurrentAccess: Has the value true if a data service is able to process more than one message concurrently otherwise it has the value false.

TransactionInitiation: Describes under what circumstances a transaction is initiated in response to messages. The values are as follows: *NotSupported* – does not support transactions; *Automatic* – transaction initiated automatically for the duration of each message; *Manual Transaction* – context under control of the consumer, for example using an existing transaction specification, such as WS-AtomicTransaction [8].

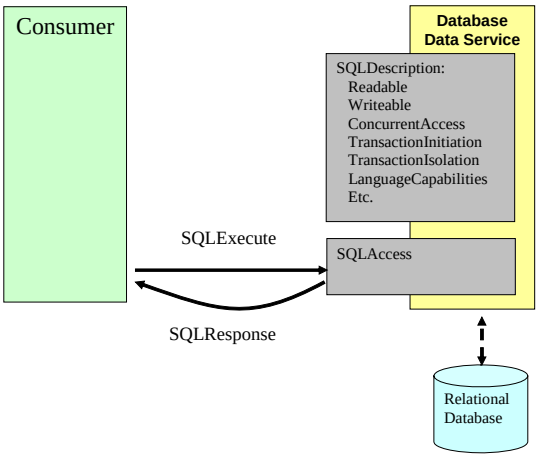
TransactionIsolation: describes how transactions behave with respect to other ongoing transactions.

ChildSensitiveToParent and *ParentSensitiveToChild:* indicates whether a parent or child data resource is sensitive to changes made to the other.

Direct Access

The DAIS-WG specifications, for the most part, rely on the existing query facilities of database systems for extracting data from and modifying the contents of data resources. Operations are provided that pass query statements as strings, which in turn are passed on to the underlying data resource management system; as such, the specifications can be seen as defining web service wrappers for the underlying databases.

The WS-DAI specification supports two patterns for obtaining the results of requests directed at a data resource, referred to as *direct data access* and *indirect data access*. Direct data access follows the typical web service type of interaction where a consumer expects the data or status of a query in the response to a request. For example, passing a message containing a SQL query to a data service will result in a response message containing the rowset representing the result of the query as an XML document, as illustrated below.



For direct data access, the WS-DAI specification defines a single query-language-independent message, and a template for realizations to follow in defining language-dependent operations.

Generic Query

A message for passing generic queries to a data resource. The actual query language payload is implicit in the QName of the message, and is specified by a *LanguageMap* property. Operation descriptions are not given in WSDL here; the WSDL is provided in the specification [9]. The following terminology is used to indicate multiplicity in the examples below: a "?" indicates that a parameter is optional; "*" denotes 0 or more; "+" denotes one or more; and no modifiers signifies that one such element must be present.

Input. *GenericQueryRequest*:

1. *DataResourceAbstractName*: abstract name of the target resource.
2. *DatasetFormatURI*: An element that can be used to define the type of the response message. This element must contain a URI from the set that appears in the *MessageDatasetMap* properties defined by the data service. When only one URI is advertised this element may be omitted, in which case the format of the response message will follow that of the type reference by the advertised value.
3. *GenericExpression*: the query expression document.

Output. *GenericQueryResponse*:

1. The *DatasetFormatURI* used to format the response.
2. The response document formatted according to *DatasetFormatURI*

GenericQuery is an extensibility point for a service; the specifications are not prescriptive with regard to the languages that might be supported in different contexts, and thus, like other extensibility points, interoperability is traded for flexibility.

Query Template

The WS-DAI specification does not provide any query-language-specific operations; these are the responsibility of the realizations. Rather, WS-DAI defines a structure for direct data access request messages that is followed by the realizations. The structure is as follows:

```
<RequestMessage>
  <wsdai:DataResourceAbstractName/>
  <wsdai:DatasetFormalURI/>?
  <RequestDocument/>
</RequestMessage>
```

The elements within the template are as follows:

1. *RequestMessage*: This is the root element for a request message. The type of this element is specific to each message. For example, the relational realization defines *SQLExecuteRequest* as a *RequestMessage*.
2. *DataResourceAbstractName*: abstract name of the target resource
3. *DatasetFormatURI*: as in Generic Query.
4. *RequestDocument*: The request statement. The structure of this document is specific to the statement being used. For example, the relational

realization defines a *SQLExpression* element as a *RequestDocument*.

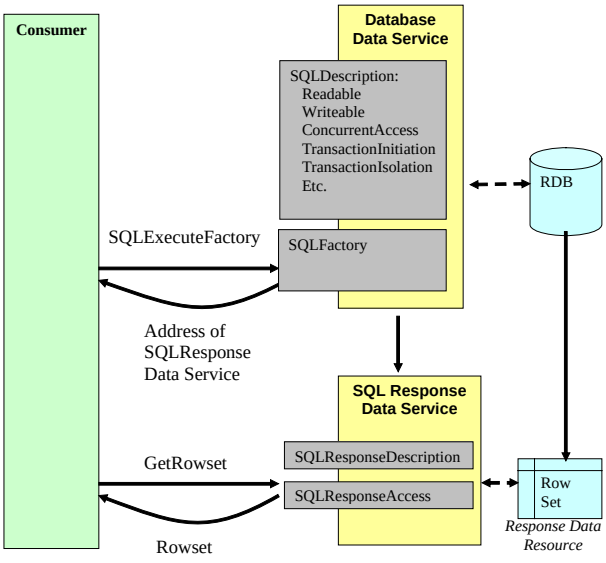
The structure of a direct access response message is:

```
<ResponseMessage>
  <wsdai:DatasetFormat/>
  <wsdai:DatasetData>
    Data formatted according to the
    DatasetFormatURI goes here.
  </wsdai:DatasetData>
</ResponseMessage>
```

Indirect Access

Indirect data access essentially implements the factory pattern, whereby the result of a request is not returned directly to the user, but rather made available as a data resource in its own right, for access through a data service. The consumer thus gets an end-point reference in the response message through which the data may be accessed. The WS-DAI specification does not define any generic indirect access operations, but it defines a template that must be followed by the realizations that implement this type of operation.

To see an example of indirect access usage we thus turn to the relational realization example.



A consumer sends a *SQLExecuteFactory* message to a *Relational Data Service*. As a result of the message, the data service makes available a service managed *Response Data Resource* through a *SQL Response Data Service*. The result of the *SQLExecuteFactory* message is the concrete name of the associated *Response Data Resource*. Subsequent messages from the same consumer, or a different consumer, can be used to access the contents of the *Response Data Resource*.

This pattern has been introduced into the DAIS-WG specifications in part to support third party delivery (and also to avoid unnecessary data movement). The specifications support third-party-pull delivery directly – a consumer *C1* can invoke a *SQLExecuteFactory* operation and pass the concrete name of the result data resource to consumer *C2*, which can then access the result directly, avoiding the need to transfer the result to *C2* via *C1*. Other patterns of third party delivery, such as third-party-push delivery, are expected to be supported by implementing portTypes from OGSA data movement services on the data services used to provide access to the results of factory operations.

As in the case of direct data access, the specification defines a template that is implemented by the realizations. The structure of a message implementing the factory pattern is:

```
<RequestMessage>
  <wsdai:DataResourceAbstractName/>
  <wsdai:PortTypeQName/>?
  <wsdai:ConfigurationDocument/>?
  <wsdai:PreferredTargetService/>?
  <RequestDocument/>
</RequestMessage>
```

The components of the template are as follows:

1. *RequestMessage*: This is the root element for a request message. The type of this element is message-specific. For example, the relational realization defines *SQLExecuteFactoryRequest* as a RequestMessage.
2. *DataResourceAbstractName*: The abstract name of the target resource.
3. *PortTypeQName*: The QName of the port type that the resulting resource will be accessed through. This must correspond to one of the QNames defined in the *ConfigurationMap* property of the service. If no *PortTypeQName* is specified the port type specified by the first *ConfigurationMap* property is assumed.

6. *RequestDocument*: This is the request part of the message. The structure of this document is message-specific. For example, the relational realization defines a *SQLExpression* element as a *RequestDocument*.

The structure of the factory pattern response message is:

```
<wsdai:DataResourceAddressList>
  <wsa:EndPointReference>
    <wsa:ReferenceParameters>
     <wsdai:DataResourceAbstractName/>?
    </wsa:ReferenceParameters>
  </wsa:EndPointReference>*
</wsdai:DataResourceAddressList>
```

The components of the template are as follows:

1. *wsdai:DataResourceAddress*: This is the root element for the response message. For example, the relational realization defines a *SQLExecuteFactoryResponse* element.
2. *wsa:EndPointReference*: A list of end points of the service(s) that provide access to the newly created data resource(s). The EPR *ReferenceParameters* element contains the abstract name of the data resource to which the address refers.

WS-DAIR: The Relational Realization

The relational realization [10] extends the properties defined in WS-DAI, instantiates the templates for direct and indirect data access, and defines interfaces for accessing the results of requests directed at a relational data service. The relational realization builds on the SQL standard throughout.

Properties

A relational data service defines a single additional property, namely *CIMDescription*: the description of the database accessible through the service, described using the model of the Database Technical Committee of the Distributed Management Task Force (DMTF – http://www.dmtf.org).

Direct Access

The *SQLExecute* operation directs a SQL statement to a relational data resource, instantiating the template defined in WS-DAI.

Input: *SQLExecuteRequest*. The elements are as defined in the WS-DAI direct access template, where the *RequestDocument* becomes *SQLExpression* which contains the actual SQL query plus optional parameters.

Output: *SQLExecuteResponse*.

1. *DatasetFormatURI*: The format in which the data is being returned.
2. *DatasetData*: Any data returned in response to a query.
3. *SQLUpdateCount**: The number of rows that were affected by an SQL update if this was the type of SQL statement used.
4. *SQLOutputParameter**: Any output from a SQL stored procedure output parameter.
5. *SQLReturnValue?*: The return value from any stored procedure.
6. *SQLCommunicationsArea+*: Any output from the SQL Communications Area.

The following is an example of an input message to *SQLExecuteRequest*:

```
<SQLExecuteRequest>
  <DataResourceAbstractName>
     urn:dais:dataresource27
  </DataResourceAbstractName>
  <SQLExpression>
    <Expression>
      SELECT * FROM P
   </Expression>
  </SQLExpression>
</SQLExecuteRequest>
```

Indirect Access

The *SQLExecuteFactory* operation directs a SQL statement to a relational data resource, instantiating the template defined in WS-DAI. As such, the result of the SQL statement is made available to the consumer as a data resource by way of a data service.

Input: *SQLExecuteFactoryRequest*. The elements are as defined in the WS-DAI indirect access template, where the *RequestDocument* becomes *SQLExpression*.

Output: *SQLExecuteFactoryResponse*. A list of addresses of the data resources for the result(s).

The following is a fragment of an example *SQLExecuteFactoryRequest*, which includes both the query and the properties used to configure the data service that will provide access to the result:

```
<SQLExecuteFactoryRequest>
  <DataResourceAbstractName>
   urn:dais:mydataresource1234
  </DataResourceAbstractName>
  <PortTypeQName>
      dair:SQLResponsePT
  </PortTypeQName>
  <ConfigurationDocument>
    <Readable>true</Readable>
    <Writeable>false</Writeable>
    <ConcurrentAccess>
      false
    </ConcurrentAccess>
    <TransactionInitiation>
      NotSupported
```

```
    </TransactionInitiation>
     …
  </ConfigurationDocument>
  <SQLExpression>
    <Expression>
     SELECT * FROM P
    </Expression>
  </SQLExpression>
</SQLExecuteFactoryRequest>
```

Although the DAIS-WG specifications allow other interfaces to be used to provide access to the results of a *SQLExecuteFactoryRequest*, the *SQLResponse* and *SQLRowset* data services have been defined for this purpose.

SQLResponse

A *SQLExecuteFactoryRequest* may return the concrete address of a data resource based on the result of any SQL statement. As such, the *SQLResponse* data service provides properties that describe the actual result and operations for accessing those results.

Space does not allow a detailed description to be provided, but the names of the properties are fairly self-explanatory, and are: *NumberOfSQLRowSets*, *NumberOfSQLUpdateCounts*, *NumberOfSQLReturnValues*, *NumberOfSQLOutputParameters* and *NumberOfSQLCommunicationsAreas*.

Associated with these properties are a collection of data access operations, namely *GetSQLResponseItem*, *GetSQLRowSet*, *GetSQLUpdateCount*, *GetSQLReturnValue*, *GetSQLOutputParameter* and *GetSQLCommunicationsArea*.

In a third-party pull data delivery, the results of a query can be obtained by a consumer using the collection of operations defined in *SQLResponse*.

SQLRowSet

A *SQLRowSet* data service essentially provides access to a single table. It has the properties: *AccessMode*, which indicates if the rowset can be read only sequentially or whether access by position is supported; and *NumberOfRows*, which indicates how many tuples are stored in the rowset. An operation, *GetTuples*, provides access to a group of tuples from the rowset, indexed by position.

WS-DAIX: The XML Realization

The XML realization [11] extends the properties defined in WS-DAI, instantiates the templates for direct and indirect data access, and defines interfaces for accessing the results of requests directed at a XML data service. Space restrictions preclude a fuller description of WS-DAIX here, but it follows a similar pattern to that described for WS-DAIR. Key features include: properties and operations for manipulating collections of documents, direct and indirect access using both XQuery and XPath, and data modification using XUpdate.

Conclusions

The development of standards for accessing databases from programming languages is well established, and has saved countless hours of developer effort though improved portability and interoperability. Such standards are widely deployed for relational databases, where Embedded SQL is part of the SQL standards process (ISO/IEC 9075 at http://www.iso.org/) and JDBC® is part of the Java Community Process (http://jcp.org/en/jsr/-detail?id=054). Programming language APIs used to access XML databases are less well established, although several native XML database systems support XML:DB (http://xmldb-org.source-forge.net/xapi/), and the development of the XQJ standard within the Java Community Process is now well advanced (http://www.jcp.org/en/jsr/-detail?id=225). The WS-DAI family of specifications should bring similar benefits to service-based computing, by making data resources available though consistent WSDL interfaces. Benefits that result from the provision of data access standards as web services include: (i) no need to deploy database connectivity software on clients; (ii) seamless integration with other web service standards, for example, for security, transaction management and data movement.

The argument here is not that databases should always, or even normally, be made available in a service-based setting using the interfaces described in this paper; indeed most access to databases in service-based applications will be completely transparent, hidden behind domain-specific services. However, at some level in any distributed application it becomes necessary to direct requests at resources, whether data or computational resources. Service-oriented grid middleware seeks to make the secure and coordinated interaction with distributed computational resources more systematic, more consistent and more compositional, and thus more cost effective. The WS-DAI family of standards seeks to contribute to this process by making databases first class citizens in service-based grids. At the time of writing, the specifications have been submitted by the DAIS-WG to the public comment phase of the GGF standardization process; as such it is anticipated that the specifications will be refined in

the light of feedback for adoption as standards during 2006.

Availability: Prototype implementations of WS-DAIR and WS-DAIX will be made available from http://www.ogsadai.org.uk during the first half of 2006.

Acknowledgements: Many people have contributed at different stages to the development of the DAIS-WG specifications, including Malcolm Atkinson, Brian Collins, Shannon Hastings, Stephen Langella, James Magowan and Greg Riccardi; fuller acknowledgements are provided on the specifications themselves.

References

[1] Database Access and Integration Services Working Group (DAIS-WG), http://forge.gridforum.org/projects/dais-wg

[2] Global Grid Forum (GGF), http://www.ggf.org

[3] Open Grid Services Architecture Version 1.0, I. Foster *et al.*, Technical Report GFD-I.30, Global Grid Forum, 2004.

[4] Data Access, Integration and Management, M.P. Atkinson, *et al.*, in The Grid: Blueprint for a New Computing Infrastructure (Second Edition), I. Foster and C. Kesselman, Morgan-Kaufmann, 391-430, 2004.

[5] Grid Database Access and Integration: Requirements and Functionalities M.P. Atkinson, V. Dialani, L. Guy, I. Narang, N.W. Paton, D. Pearson, T. Storey, and P. Watson. Technical Report GFD-I.13, GGF, 2003.

[6] Web Services Security: SOAP Message Security 1.0 (WS-Security), A. Nadalin, C. Kaler, P. Hallam-Baker, and R. Monzillo. OASIS, 2004.

[7] Web Services Resource Framework (WSRF) Primer, T. Banks, http://docs.oasis-open.org/wsrf/wsrf-primer-1.2-primer-cd-01.pdf, OASIS, 2005.

[8] Web Services Atomic Transaction (WS-AtomicTransaction) 1.1. E Newcomer, I Robinson (eds), Working Draft, http://www.oasis-open.org/committees/download.php/15719/WS-AT%20Working%20Draft.pdf, 2004.

[9] Web services data access and integration – the core (WS-DAI) Specification, Version 1.0. M. Antonioletti, M. Atkinson, A. Krause, S. Laws, S. Malaika, N.W. Paton, D. Pearson, G. Riccardi. GGF, 2005.

[10] Web services data access and integration – the relational realization (WS-DAIR), Version 1.0. M. Antonioletti, B. Collins, A. Krause, S. Laws, S. Malaika, J. Magowan, N.W. Paton. GGF, 2005.

[11] Web services data access and integration – the XML realization (WS-DAIX), Version 1.0, M. Antonioletti, A. Krause, S. Hastings, S. Langella, S. Laws, S.Malaika, N.W. Paton. GGF, 2005

Moshe Vardi Speaks Out
on the Proof, the Whole Proof, and Nothing But the Proof

by Marianne Winslett

Moshe Vardi
http://www.cs.rice.edu/~vardi/

Welcome to ACM SIGMOD Record's series of interviews with distinguished members of the database community. I'm Marianne Winslett, and I have here with me Moshe Vardi, who holds an endowed professorship at Rice University and is a former chair of their Computer Science Department. Before joining Rice, Moshe was a manager at IBM Almaden Research Center. Moshe is an ACM Fellow, a AAAI Fellow, a co-winner of the Goedel Prize, and a member of the U.S. National Academy of Engineering and the European Academy of Sciences. His research interests include databases, verification, complexity theory, and multi-agent systems, and his PhD is from the Hebrew University of Jerusalem. So Moshe, welcome!

Thank you very much. It is a pleasure to be here.

Moshe, after you finished your PhD, you worked on Datalog for a number of years---as did many, if not all, database theoreticians. At the time, there was vocal opposition to some aspects of this activity from some of the more practically oriented members of the database research community. In hindsight, do you think that the criticism was justified?

It surprised me that people get so emotional about certain research areas. The work on integrity constraints in the late 70s and early 80s also received scathing criticism as not being at all relevant to the practice of database systems, only to reemerge later as being of central importance. I heard recently a question that someone asked Stephen Hawking: what is the best idea in physics this year? And Hawking said that we won't know for many years. When you do an exciting piece of research, it is very hard to know whether it will be relevant to the field in the long term. This is true both for theory and for experimental work. The vast majority of theory research results will be forgotten, as will be the vast majority of experimental work. The fact that something is done experimentally does not guarantee any lasting impact.

We can look back now at Datalog, and ask what its impact was. I don't know of any database that implements Datalog per se, but SQL3 does include recursion. A case can be made that the Datalog work did contribute to realizing the importance of recursion. Georg Gottlob has a company that does Web integration using Datalog as its internal language. The Datalog concept of rules has had an impact on active databases. So yes, Datalog did have an impact. That doesn't

mean that every single technical result, every PSPACE-completeness result, had much impact; but the same is true for most scientific work. Most scientific work will be forgotten.

Datalog has been very influential in the area I work in, in security.

Languages for security policies?

A very good language for security policies. Although it is not necessarily the direction that the industry is going in, perhaps they should *be going in that direction. Certainly in security research, you want to be able to reason about the correctness of what you are doing.*

What do you view as the most important open problems in database theory today?

Have you tried to use databases? They are darned hard to use. We have built a marvelous piece of middleware that is incredibly powerful and incredibly difficult to use. The challenge for the database field is how to have more usable database systems.

I use databases all the time. I use them every time I go to an ATM machine, and it is very easy. So why are you complaining that they are hard to use?

For you it is easy, but it is very expensive to make them easy for the consumer to use.

So you are not worried about ease of use, you are worried about the expense of making them easy to use.

Someone actually has to write the application. At an ATM, you don't deal directly with the database, you are interacting with an application. It is very expensive to write an application and maintain the database---I'm not going to be the first person or the last person who will say that. What we lack (and this is true both on the implementation side and the theoretical side) is a good set of abstractions that will make life easier for people who build databases and people who build and maintain applications.

The relational model overall was a wonderful abstraction, a beautiful abstract model. It cleared away a lot of bushes from the pathway, and life became easy for a while. Now the bushes are growing across the pathway again. If you talk to people in forestry, they will tell you that fires are good because they clear the brush from the forest. The relational model was a fire that cleared a lot of brush away, but we haven't had a good fire for a while.

Is there a role for the theory community there? Basically, you are talking about a revolution in the API?

Well, the API is one way to look at it, but the relational model first and foremost was a theoretical contribution. The experimental work came later to show that this theoretical contribution is realizable and can result in practically and pragmatically usable databases. In our time, we have tried another abstraction. Everybody claimed that object-oriented databases were the way things would go. But it didn't work out that way, so we can ask, why not?

There is no doubt that we are looking for new abstractions, and in that task theory plays a critical role. It is not enough to have new theory, it must also be paired with implementations and experimental work. In the database research community culture, I don't think we have a very healthy ecosystem where there is a constant interplay between theory and practice. Instead, for

example, the theoreticians come and say that integrity constraints are important, and the more experimental people say that no, they are not important. After a while the theory people stop working on integrity constraints. Then the experimental people say, "No, no, we need integrity constraints!" And the theory people say, "Oh, we are not interested now, we looked at them ten years ago and now we're not interested." In other areas of my research, verification for example, there is a much more healthy interchange between more theoretical research and more applied research.

I think I misunderstood you earlier. You don't want a better API, you want a new model*?*

It is not necessarily just the model; there is a whole set of abstractions that goes into building a system. The relational model was one such abstraction. Another abstraction was logic and relational algebra as a way to query. Another abstraction was the concept of keys, as a way to describe integrity constraints, and normalization. Transaction management was another kind of abstraction. So we are not tied to one single abstraction; we need a mix of abstractions. We can't just replace tables by trees and be done. It's more than that. But the complexity of today's systems suggests that there is a prime opportunity for the theory community to develop and reason about new abstractions.

Do you have any proposed directions to find these new abstractions?

I don't. If I had, maybe I wouldn't be here! There is some work happening in the XML world, where people are trying to figure out a more abstract way of looking at XML. Maybe that will have lasting impact. Interestingly, we see theory playing a bigger role in the XML world than it did in the relational world. If XML becomes the primary way of storing data, then perhaps we will see some impact there.

In that earlier fiasco with dependency theory, I think that the integrity constraints that the theory people were looking at were not the kinds of integrity constraints that come up in practice, so that is where the mismatch happened.

I think that was part of it. At the time we theoreticians didn't get feedback saying, "This is very nice, but here is the constraint that we really care about, and please work on it instead." Instead, what we got was a scathing response that nobody needs anything more than keys. And then later, "Well, maybe we'll eventually need referential integrity, but nothing else." Today if you look at SQL, you find that there are keys, referential integrity, and assertions. Assertions are incredibly powerful and we never did develop a good theory of how to handle them well. So we don't have a healthy interchange between theory and experimentalists. I'm perfectly okay if somebody says, "Very nice work, but if you can change it a little bit, maybe it will better fit my needs," rather than "You're wasting our time, you're wasting your time, stop bothering us," which is, I have to say, very often what we were hearing.

There is no Nobel Prize in computer science, but there is a Nobel Prize in economics. I have heard that database theoreticians have switched to working on game theory for this reason. Is that rumor true?

People work on game theory because games are a powerful paradigm. For example, I use games in my verification work. I don't think game theory work will get a Nobel Prize. I think one should not structure one's work for Nobel Prizes. Too few people get them; it is not a good career driver. One should strive to make impact, to have success, for other considerations. I

know people who work on bioinformatics because they think they might get the Nobel Prize in medicine. I think it is a bad strategy for designing one's career.

A Nobel Prize is *a bit of a long shot, but maybe one can pick up some other prizes along the way.*

Game theory already received the Nobel Prize twice in economics, which surprised people. Some people thought after the first prize that game theory had had its time, and then there was a second prize for game theory. So I wouldn't bet on a third one.

How about bioinformatics?

Bioinformatics is a less of a long shot in some sense. Again, one would have to do something fairly dramatic to win a Nobel in that area.

You could argue whether the contribution by some people to the genome project was worthy of the Nobel Prize in medicine. I suspected that that project was so controversial that nobody from that project would get the Nobel Prize, because the Nobel Prize committee would not want to sort out the intense rivalries there. If the medicinal contributions of the genome project would materialize, which so far has not happened, then the genome project would be one of the most successful contributions of computer science to medicine.

Moshe, you were one of the main players in the area of finite model theory. Many of our readers will not be familiar with this fascinating subject. Can you tell us a little bit about it? Perhaps some of your favorite results in the field, and how they may relate to the database world?

Finite model theory emerged at the confluence of databases, complexity theory, and logic. The realization that several people had independently was that classical mathematical logic, and model theory in particular, deal with infinite structures. They deal with infinite structures because arithmetic is infinite and real numbers are infinite; logicians almost looked down on finite structures. *Finite* was almost boring; it was too trivial to study. We realized that there is motivation coming from computer science to study finite structures, and a beautiful theory emerged. Questions and issues come up that do not arise at all when you look at infinite structures.

You can ask what the implications of finite model theory are for databases and complexity theory and perhaps for other areas. The answer is that we don't know yet. So far, in my opinion, the implications of finite model theory for any other area are somewhat modest. We have a beautiful mathematical theory and there are some textbooks. We hope that the beauty can eventually lead to applications, just like in any other area of mathematics, but that can take a while.

From the finite model theory work, we learned about the limits of expressivity of first-order logic. When SQL was developed, Ted Codd and other people thought that first-order logic was as expressive as one would ever need. The work on finite model theory led us to understand the limits on expressivity of first-order logic and therefore of SQL. This contributed to the eventual inclusion of recursion in SQL, because we now had formal proof that recursion does add expressive power. This is probably the most measurable impact of finite model theory on databases.

There was some hope that finite model theory work on random models might tell us something about average query processing behavior. I don't think that has been borne out. People were hoping that finite model theory would lead to significant progress in complexity theory; that so

far has not been borne out. There is some recent work that uses finite model theory to give us new techniques in query optimization, and that might become significant.

What are the hot topics in computational logic today?

I think that one of the major successes in computational logic has been the applications of logic to theoretical formal methods. (I just discovered that [the Department of Computer Science here at UIUC] has a very large group in formal methods.) The issue of how to design better systems has been one of the challenges of computer science for the last 50 years. Just in the last decade, formal methods have been coming to the fore as a major body of techniques to do better systems design, programming, debugging, and other tasks related to creating more reliable computer systems. Formal methods have had major success in industry.

I would love to see more of my database work having impact on database practice, but most of my work has been theoretical, without technology transfer. I think that some of my theoretical verification work will have an impact on industrial practice. That is where I spend most of my time: not in database research, but in trying to transfer ideas from computational logic to industrial practice. I think it is a very exciting area nowadays.

You served as department chair at Rice for many years.

Too many!

Nowadays, many departments are working hard to move up in the US News & World Report *rankings---so much so that a department has to continually improve itself just to maintain its current position. Finite model theory tells us that this cannot go on forever. So where will it all end?*

The goal of improving a department's ranking is not a realistic goal. You can say that you want to have more graduate students; that is something you can measure: how many graduate students do you admit per year? You can look at the average GRE score of the students, and say that you would like to have better graduate students. You can say that you would like to see your department getting more funding. There are all kinds of things you can measure and you can control, but you cannot control the ranking that *US News* will generate with their complex formulas. Since everybody is trying to improve, it would be nice if you could squeeze more departments into the top ten. But since only ten departments can be in the top ten, I think it is not a useful goal for departments to have. I would advise departments to focus on measurable goals and attainable goals. I never felt that the goal of ranking improvement was useful or attainable.

You can only improve your graduate student population so far, and you can only increase your funding so much. If everyone else is also trying to compete for the same pool of excellent graduate students and to compete for the same pool of research dollars, how is it all going to end?

I will tell you where it is going. Everything is going down, unfortunately. There is a biblical story that says that there were seven fat cows, and there were seven lean cows. This meant that there were seven good years, and they were followed by seven bad years. Right now, I look at the global situation and see that we are in for a tough period. Now our job is to survive in this tough period and do the best we can.

Well, what the biblical story tells us is that during the fat years, plan well for the lean years. That was the wisdom of Joseph. I don't think we did it. We enjoyed the good years and we thought they would last forever. And they didn't last forever.

What should we have done that we didn't do? Go to Congress and talk up the field?

We have not communicated effectively the contributions of our field. The public as a whole knows very little about it. When you open the newspaper, you find stories almost once a month about black holes. Somehow, the stories excite people about black holes. When you think about it, it is utterly bizarre; why would be people be so interested in black holes in the center of the galaxy? But people *are* interested in black holes. The authors made the topic interesting. We computer scientists have not told our stories well to make them interesting to the public. Maybe our topics are more difficult, more abstract; it is hard to get people excited about theoretical algorithms. But still, how many books do you know that tell exciting stories of computer science to the public? Compare that to physics, even to string theory: there are books for the public about string theory!

There are books about computer science for the public, but they often concentrate on things like startups and successful companies in computer science.

So the public does not have an appreciation for computer science. Now, how does public appreciation translate into Congressional support and funding? That is not so obvious, but when you go to Congress it does help to have background appreciation for what your discipline has done. Computer science has not built that background appreciation; we have hunkered in our own little corner. We have done great research and we built amazing infrastructure, but if you ask the public what they know about computer science, they will list the Internet, and very little other than that. They know very little about research challenges in computer science.

So does that mean we'll be seeing stories in the New York Times *about finite model theory soon?*

It is an art to tell the story in a way that people find interesting. I heard David Harel give a wonderful talk to the public about what computers cannot do. And he is one of the few people that have been able to write very cogently about theoretical computer science, about computability, about complexity theory. Now, of course, he doesn't talk about the complexity of Ehrenfeucht games. When people write about quantum theory for the public, they don't present all the gory details of Schroedinger's equations. We need to know how to tell our story. We have not developed this set of skills. I think it is more difficult for us than for other disciplines, because there is something about the natural world that people inherently find more interesting than abstract and artificial worlds such as ours.

People are starting to think about how to tell our story. We rarely publish in *Science* and *Nature*. Should we try to publish there? Is it a good idea? How do we do it? I am hearing a conversation that I haven't heard until very recently, partly driven by the current crisis.

What words of advice do you have for database groups in academia who would like to improve their group's standing?

We now have interesting tools to evaluate the success of work in the long term. Things like Citeseer and Google Scholar suddenly give us a view that we could not have had before. One thing that we discovered from these tools is that we are actually very poor in predicting the long

term impact of work. There is very little correlation, for example, between the best paper awards that we give and the test-of-time awards that we give. Sometimes, miraculously, we have the same paper win the best paper and the test-of-time awards. But that is the exception rather than the rule. So I think people should focus on just doing the best work they can.

What you just said implies that the low acceptance rates at conferences now are actually a problem, because we may be pruning out those papers that would win the ten-year paper award.

I think the low acceptance rate is a terrible problem. I think that the idea that we are raising the quality is nonsense. I think that actually the quality goes down. I think we are very good at selecting about one third to one fourth of the papers; we do a pretty good job there. As we become more selective, the program committee gets larger, the whole discussion gets more balkanized, and the whole process gets more random. Nobody gets a global view. The program committee used to look at the whole set of papers and there was some consensus; people would argue only about the marginal papers. Now the whole enterprise is so large that it is effectively broken into several subcommittees, and I have no confidence that we are really selecting the best papers. I have heard other people say that we are encountering the problem that we deal with in our own research: scalability. Conferences are not scalable. They work nicely with up to roughly 300 submissions and a certain size of program committee. When you try to scale it up, very good papers get lost. It becomes more political. I think we are being held back by our own success.

What would you propose as a remedy?

We are very unique among all the sciences in how we go about publication. We have these selective conferences. (People have stopped calling them "refereed conferences." They are not really refereed. You don't get good referee reports.) Our conferences worked well in the world we used to inhabit. We assume that because they worked, they are scalable; but there are reasons to doubt the scalability of this model.

I don't have a good solution to this problem. We don't even have a good forum in which to discuss the problem. It's not just a problem for one part of computer science, it is a problem for all of computer science. How can computer science go about changing its publication culture? Are there areas that move just as fast as we do, and have journal papers and conferences, but conferences are not the primary vehicle? I have questions about the basic model of scholarly publications. And I find it fascinating that it is difficult to have a conversation about this on a big scale, and make changes on a big scale. We are very conservative. It is interesting that computer science has been one of the slowest disciplines to move to open access publications. Other disciplines are way ahead of us in using online publications.

So you must not be thinking of ACM SIGMOD's DiSC and Anthology?

Those are online repositories of printed publications. When you go to the SIGMOD conference, you get a monster proceedings volume. It is expensive to produce it. Do you need it?

We don't do that anymore. Now you get a DVD, and you can buy the printed version if you want.

Right, actually, I find I cannot store DVDs. I am not organized enough to store DVDs. All I care about…

I want it on the Internet. If I have it there, I don't need a DVD. At least *books* are thick enough that you can put them back on the shelf and you can find them later. I have yet to see anybody organize DVDs in such a way that you can find a DVD when you want it. So I just want the proceedings on line. I don't know why conferences give people a DVD of the proceedings. Maybe during the conference you can use it, and during a talk you can open the paper. But long term, I don't even keep the DVDs any more.

To back up just a moment, it sounds like you are talking about moving us more towards a journal culture?

I'm raising the issues. I think we had a model that worked successfully for about 30 years, but now we see cracks in the foundations. We ought to rethink how we do it. Right now, people try to fix things incrementally by having a larger conference with a bigger program committee, a two-level PC, a three-level PC. Maybe we need to rethink the way we do scholarly communication in our discipline.

As the number of really good database researchers continues to grow exponentially in the US (because we all graduate more than one PhD student in our lifetime), while the available funds for research remain constant, what changes do you foresee in the way research is carried out?

Exponential growth is never sustainable---we know this. During the period of exponential growth, we all get very excited, and we think this will go on forever. Nothing goes on forever. Exponential growth always hits some kind of a ceiling. We have been hitting our ceiling.

So funding is going to be scarcer, and that will translate to fewer graduate students. If the graduates cannot find good jobs, that will translate to fewer graduate students. Fields go through periods of growth and then some fields decline after that. We are too young to have seen this ebb and flow of things, so we think our field is always in growth. We had a period of very heady years: we had the late 90s, which were a very atypical period. There was an Internet bubble outside, but in some sense there was also a bubble inside our discipline. We still have to come to grips with that. What is the realistic size of our discipline? That will depend on the interest by students; that will depend on the funding available. The fact that we want to have more students ultimately is not the only factor that will determine what is going to happen. There is a big world out there, and we have to learn to roll with the punches.

Do you have any words of advice for fledgling or mid-career researchers or practitioners?

I find that the things ultimately that I have success with are the things that I find at the time just to be an enjoyable piece of research. When I did research that I really enjoyed and that I thought was beautiful, very often it became the piece of research that had long term impact. I have to admit I had some beautiful papers that only I loved and nobody else cared about; generally we are not very good at predicting the ultimate success of our own work. There is a famous story about a party at the Cavendish Lab in Cambridge around 1900, where the physicists toasted, "To the electron: may it be of no use ever!" Even technical people, scientists, are not good at predicting the ultimate impact of their own work. So do the work that you think has lasting power; some will last, and some will not, but at least you will have fun in the process.

Among all your past research, what is your favorite piece of work?

Paul Erdos, the great discrete mathematician, had a concept of *God's Book of Proofs*. He said that mathematicians produce many proofs, but once in a while there is a proof so beautiful that

God says, “Ah, this one I didn’t see coming. This one is so beautiful that I am going to put it in my own book.” Long term, the things that I appreciate are the things that are the most beautiful, the things that have the most aesthetic value---sometimes these are the things that also have the most long term impact, but not necessarily. I have a few proofs that I would consider submitting to God for his book of proofs.

My work in verification was about translating from linear temporal logic to automata; it established a new connection, and I like that work very much. The thing that drove me in the beginning was the aesthetics of it. There are some results in finite model theory that have not had as much impact, but I think they are very elegant aesthetically. I recently put on my web page two slogans. One says, “Theorems Are Forever,” and the other says, “The proof, the whole proof, and nothing but the proof.”

If you magically had enough time to do one additional thing at work that you are not doing now, what would it be?

I would like to write a textbook on teaching logic in computer science.

Oh, that would be great!

There is a course I have been teaching for many years, about logic from a computer science perspective. It is very algorithmic; I make a lot of effort to convey to students why logic is important in computer science. I would love to have the time to write such a textbook.

When you say it is algorithmic, do you mean that you spend a lot of time on automated proof theory?

For example, we talk about propositional logic, and we talk about satisfiability, both from a complexity point of view and algorithmically. The course has a lot of programming, and for the course project the students write a satisfiability solver. We talk about databases and first-order logic as a query language. We spend quite a lot of time thinking about query evaluation. We don’t do SQL, but the students understand the concept of formulas as queries. There is a deep connection between logic and computer science. There are some textbooks that try to make this connection, with a focus mostly on verification, but I think the connection between logic and computer science is much deeper than that.

If you could change one thing about yourself as a computer science researcher, what would it be?

I would be better organized. I am not a very organized person. My office continually looks like a hurricane just passed through it. My time management skills are not very good. I would love to be better organized.

But a hurricane did *just pass through your office, didn’t it?*

Both Rita and Katrina went a little north of us.

Thank you very much for talking with me today.

It has been a pleasure.

Query Reformulation with Constraints*

Alin Deutsch
University of California at San Diego
deutsch@cs.ucsd.edu

Lucian Popa
IBM Almaden Research Center
lucian@almaden.ibm.com

Val Tannen
University of Pennsylvania
val@cis.upenn.edu

1 Introduction

Let Σ_1, Σ_2 be two schemas, which may overlap, $\mathcal{C}$ be a set of constraints on the joint schema $\Sigma_1 \cup \Sigma_2$, and q_1 be a Σ_1-query. An **(equivalent) reformulation** of q_1 in the presence of $\mathcal{C}$ is a Σ_2-query, q_2, such that q_2 gives the same answers as q_1 on any $\Sigma_1 \cup \Sigma_2$-database instance that satisfies $\mathcal{C}$. In general, there may exist multiple such reformulations and choosing among them may require, for example, a cost model.

In 1999 we published an algorithm, called Chase and Backchase (C&B), for enumerating the reformulations of a query under constraints [11]. Our main motivation was query optimization, in which Σ_1's role is played by the *logical schema* and Σ_2's role by the *physical schema*. We found that the assertions used for integrity constraints (a.k.a. dependencies), by relating the elements of the logical and physical schemas constitute a flexible tool for modeling ideas such as "semantic" optimization [4], and the use of cached data or materialized views [33, 3].

The 1999 paper did not limit itself to the standard relational model and instead, following [30] and more distantly [6, 23], covered complex values and OO classes with extents. A comprehensive approach to query optimization for this model, including join (usual and dependent) reordering, appeared in [28], see also[29].

Query reformulation is also essential for data publishing [32, 12] where Σ_1 is the public schema and Σ_2 the proprietary (storage) schema. It is equally essential in schema evolution where Σ_1 respectively Σ_2 is the old, respectively new schema.

Since views can be modeled as a pair of inclusion constraints, the C&B algorithm provided a new technique for rewriting with views [25] and hence was also applicable to information integration. In fact, we had already shown in [11] that C&B will find all reformulations of conjunctive queries using conjunctive views, if such reformulations exists. However, we should emphasize that C&B finds *equivalent* reformulations while in information integration, when equivalent reformulations may not exist, one is also very much interested in reformulations that produce some (as many as possible) of the answer tuples [26, 1, 21, 7].

As its name suggest, C&B is using the *chase*, a technique developed 25+ years ago for the purposes of deciding logical consequence for most types of integrity constraints used in databases [27, 5]. Many papers have used the chase since then [1] . It seemed surprising that there would still exist fundamental properties of the chase left undiscovered. Nonetheless, we thought that the C&B algorithm provided such a property. This was formally verified in [13] where we proved that with constraints to which the chase applies, whenever the chase terminated, C&B would find all minimal reformulations of conjunctive (select-project-join) queries.

This completeness property holds also for the complex values and OO model, using a generalization of the chase developed in [30]. Moreover, the C&B algorithm was used also for the reformulation of XML queries, via a compilation from XML to relational queries and constraints [14, 12, 10]. These early successes encourage us to think that C&B could become a versatile tool for query processing. This survey will attempt to provide an introduction to the why, when, and especially how, of C&B.

2 What is C&B?

From the beginning it was observed that the chase can also be used to decide containment (hence equivalence) of conjunctive queries in the presence of constraints. Indeed, if the chase of q_1 with $\mathcal{C}$ terminates producing a query q_c then $q_1 \subseteq_{\mathcal{C}} q_2$ iff $q_c \subseteq q_2$ and the latter can be checked by finding a containment mapping from q_2 to q_c [9, 2]. (Here, $q_1 \subseteq_{\mathcal{C}} q_2$ means that when q_1 and q_2 are applied to any instance that satisfies $\mathcal{C}$ the answers of q_1 are contained in those of q_2. Similarly for $\equiv_{\mathcal{C}}$.)

In the reformulation problem, however, we are only given $\mathcal{C}$ and q_1 and we must decide whether there exists a q_2 such that $q_1 \equiv_{\mathcal{C}} q_2$. Since q_2 is among infinitely many queries of the same type as q_1 deciding this isn't obvious. Moreover, in practice we want to actually compute a q_2 when it exists, in fact we probably want to enumerate the q_2's that provide solutions and choose among them based on cost criteria. But it's easy to see that queries can be syntactically "padded" with redundant joins while conserving equivalence, ad infinitum. We are therefore led to searching for solutions that satisfy some syntactically determined *minimality* condition. (See the definition of minimality under constraints in section 4.) As a consequence, we shall

*Database Principles Column. Column editor: Leonid Libkin, Department of Computer Science, University of Toronto, Toronto, Ontario M5S 3H5, Canada. E-mail: libkin@cs.toronto.edu.

[1] In this survey we assume familiarity with conjunctive queries, homomorphisms and the chase procedure which are all covered extensively in [2]. To keep the paper self-contained we review these definitions in the appendix.

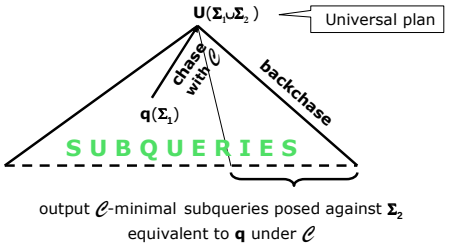


Figure 1: Chase and Backchase.

solve both the reformulation problem and a generalization of the query minimization problem [9, 2].

The C&B algorithm applies to the case when q_1 is a *conjunctive query* and when the constraints in $\mathcal{C}$ are either a *tuple-generating dependency (tgd)* of the form

$$\forall \mathbf{x}(\phi(\mathbf{x}) \rightarrow \exists \mathbf{y}\psi(\mathbf{x}, \mathbf{y}))$$

or an *equality-generating dependency (egd)* of the form

$$\forall \mathbf{x}(\phi(\mathbf{x}) \rightarrow (x_1 = x_2))$$

(see [5]). Here, $\phi(\mathbf{x})$ and $\psi(\mathbf{x}, \mathbf{y})$ are conjunctions of atomic formulas over $\Sigma_1 \cup \Sigma_2$, all of the variables in $\mathbf{x}$ must appear in $\phi(\mathbf{x})$, and x_1, x_2 must be among the variables in $\mathbf{x}$.

These two classes (tgds and egds) together comprise the (embedded) implicational dependencies [16], which seem to include essentially all of the naturally-occurring constraints on relational databases. Furthermore, tgds, which were originally meant as a generalization of integrity constraints such as join and inclusion dependencies turn out to be ideally suited for describing schema mappings in data exchange [18] and data integration [24], as well as for capturing physical structures typically used in query optimization (views, indexes, join indexes, gmaps, etc.) [11]. As a whole, the class of embedded implicational dependencies is remarkably well-suited for representing most intra- and inter-schema relationships that are of importance in practice.

C&B proceeds in two phases. In the **chase phase** it uses $\mathcal{C}$ to chase q_1 until (and if) no more chase steps are possible. We call the resulting query U, a **universal plan**, see Figure 1

Now it's time to recall that q_1 is Σ_1-query, that we are looking for a Σ_2-query as reformulation, and that the constraints in $\mathcal{C}$ are on the joint schema $\Sigma_1 \cup \Sigma_2$. The universal plan, U, resulting from the chase of q_1 with $\mathcal{C}$ will (in general) be a $\Sigma_1 \cup \Sigma_2$-query. We can think of the universal plan as incorporating *all* possible alternative ways to answer q_1 in the presence of the constraints $\mathcal{C}$. This intuition is fully justified by the following [13]:

Theorem 1 *If q_m is a minimal conjunctive $\Sigma_1 \cup \Sigma_2$-query equivalent to q_1 under $\mathcal{C}$, i.e., $q_1 \equiv_{\mathcal{C}} q_m$, then q_m is (isomorphic to) a* subquery *of the universal plan U.*

It is now possible to effectively enumerate all minimal reformulations. Indeed, we need only search the finite space of subqueries of U. This is done in the **backchase phase**, so called because we check for equivalence with q_1 by chasing subqueries of U with $\mathcal{C}$. These chase sequences go "backwards", toward the U we already have. For each such candidate reformulation we can stop (equivalence holds) whenever we have a containment mapping from U into an intermediate chase result or (no equivalence) when the chase terminates without such a containment mapping. In fact, as we shall see (Section 4), it is enough to check the existence of a containment mapping from the original query q_1 into any intermediate result of chasing the candidate subquery of U.

We see that in both the chase and the backchase phase the algorithm (and Theorem 1) needs the chase sequences to terminate. In [5] it was shown that this is always the case if the tgds are *total* or *full* [2] (they cannot have $\exists$) while the egds can be arbitrary. While full tgds cannot in general model the physical structures or the integration/exchange mappings we have become interested in, Deutsch and Popa have recently discovered a significantly larger and remarkably useful class of tgds that can. Chase sequences with such sets of dependencies, called *weakly acyclic* in [17, 18] and *stratified-witness* in [13, 14] are guaranteed to terminate. The set of constraints from Example 2 in Section 3 is weakly acyclic.

Finally, note that the subqueries of U are in general $\Sigma_1 \cup \Sigma_2$-queries. Some of them may in fact be Σ_1-queries (q_1 itself is one!) and some may be Σ_2-queries. The theorem above guarantees that if Σ_2-reformulations exist, then we shall find all minimal ones among the subqueries of U.

3 Schemas and Constraints, Queries and Rewritings

In this article we focus our presentation on a scenario where query reformulation is applied to a distributed heterogeneous environment, with multiple schemas that are interconnected by complex relationships. The problem is that of finding alternative (and equivalent) reformulations of a query that is initially formulated in terms of one of the schemas. Our running example will show the challenges (and opportunities) for query reformulation in such an environment. The example will depict constraints that fall into one of four categories:

1. **(Traditional) single-database constraints** (e.g., key and foreign key constraints.)
2. **Relationships (mappings) between schemas.** These constraints are a consequence of how these repositories have been created and subsequently maintained.
3. **Domain knowledge constraints.** These constraints are assertions that are true about a specific situation, for example, the fact that a customer id has a unique nation code across repositories.
4. **Constraints capturing materialized views.** These constraints express the fact that data is redundantly stored in both base tables and materialized views.

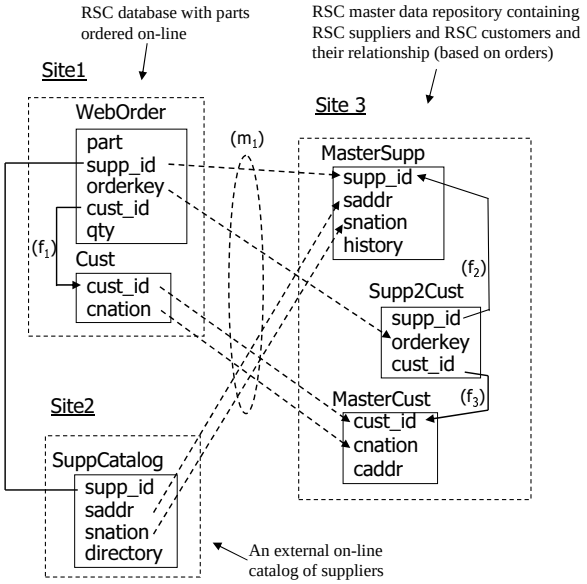


Figure 2: Retail Store Chain Example

Example 1 (Running) Consider a large retail store chain (call it RSC) maintaining and accessing several repositories with data about its suppliers, customers and parts.

One of the repositories (located at Site 2) is an external, read-only, on-line directory of suppliers. The other repositories are internal but distributed across Sites 1 and 3, with different structure, and with different although possibly overlapping data. The repository at Site 1 is a database containing parts ordered on-line and some of the associated customer and supplier information. Additional repositories like this may exist (not shown here for simplicity). The repository at Site 3 is a central repository intended to contain all the information about RSC suppliers, customers and the orders that relate them. Figure 2 illustrates the schemas of these repositories; it also depicts some of the intra- and inter-schema constraints that hold.

Example 2 (Constraints) We illustrate next some of the constraints associated with the schemas in the running example. These constraints fall under the first three categories mentioned earlier. We shall illustrate constraints in the fourth category, describing views, later in Section 5.

1. (Traditional) single-database constraints. The following egds can be used to express that $cust\_id$ plays the role of a key in the each of the tables $Cust$ and $MasterCust$ and similarly, $supp\_id$ is a key in $MasterSupp$. (As a notational convenience, we will drop the the universal quantifiers in front of a dependency, and implicitly assume such quantification.)

$$k_1: \quad Cust(c, cn) \wedge Cust(c, cn') \rightarrow cn = cn'$$
$$k_2: \quad MasterCust(c, cn, ca) \wedge MasterCust(c, cn', ca') \rightarrow (cn = cn') \wedge (ca = ca')$$
$$k_3: \quad MasterSupp(s, sa, sn, h) \wedge MasterSupp(s, sa', sn', h') \rightarrow (sa = sa') \wedge (sn = sn') \wedge (h = h')$$

The following tgds describe formally the foreign key constraints f_1, f_2, and f_3 shown in Figure 2.

$$f_1: \quad WebOrder(p, s, o, c, q) \rightarrow \exists cn\ Cust(c, cn)$$
$$f_2: \quad Supp2Cust(s, o, c) \rightarrow \exists sa \exists sn \exists h\ MasterSupp(s, sa, sn, h)$$
$$f_3: \quad Supp2Cust(s, o, c) \rightarrow \exists cn \exists ca\ MasterCust(c, cn, ca)$$

2. Relationships (mappings) between schemas. The mapping m_1 from Sites 1 and 2 to Site 3 reflects the fact that the master data repository will be refreshed with data from Site 1 and Site 2, for instance due to a periodic process that takes customer and supplier info from Site 1, joins with Site 2 to get extra supplier information (e.g., $saddr$ and $snation$) and updates appropriate tables of Site 3. Such a mapping can be specified using schema mapping tools (e.g., Clio [31]). In Figure 2, the mapping is shown informally via the dotted arrows grouped under m_1. The link between $supp\_id$ in Site 1 and $supp\_id$ in Site 2 reflects the join. Formally, the meaning of mapping m_1 is expressed by the following tgd (universal quantifiers are again dropped):

$$\begin{aligned} m_1: \quad & WebOrder(p, s, o, c, q) \wedge Cust(c, cn) \\ & \wedge\ SuppCatalog(s, sa, sn, d) \\ & \rightarrow \exists h \exists ca\ (MasterSupp(s, sa, sn, h) \\ & \wedge\ Supp2Cust(s, o, c) \wedge MasterCust(c, cn, ca)) \end{aligned}$$

Another example of a mapping between schemas (not shown in Figure 2 to avoid cluttering) is the following tgd, expressing that $SuppCatalog$ is an "authority" for supplier information, and every supplier in $MasterSupp$ at Site 3 can be found in $SuppCatalog$ at Site 2. (The converse may not be true.)

$$m_2: MasterSupp(s, sa, sn, h) \rightarrow \exists d\ SuppCatalog(s, sa, sn, d)$$

3. Domain knowledge constraints. The fact that a customer id has a unique nation code (across all repositories) is expressed by adding the following egd to the earlier key constraints:

$$e: Cust(c, cn) \wedge MasterCust(c, cn', ca) \rightarrow cn = cn'$$

Note that e is more general than a functional dependency, as it states a property about tuples in different tables.

Example 3 (Reformulations) Consider the following query (expressed in conjunctive query notation [2]):

$$\begin{aligned} q(p, c, sa, sn) \quad :- \quad & WebOrder(p, s, o, c, q), \\ & SuppCatalog(s, sa, sn, d) \end{aligned}$$

The query q retrieves all parts that were ordered at Site 1, with the addresses and nations of suppliers and with the customer ids. The query needs to access Site 1 and Site 2, to be executed in its current form.

Given the overall configuration, q is equivalent to the following (non-obvious) rewriting:

$$\begin{aligned} q'(p, c, sa, sn) \quad :- \quad & WebOrder(p, s, o, c, q), \\ & MasterSupp(s, sa, sn, h) \end{aligned}$$

The query q' accesses Site 1 and Site 3 (all within RSC) and avoids the external catalog (which could be slower, less available, may require subscription, etc). Thus, q' is potentially more efficient with respect to execution time or cost.

If for Example 1 we have that Σ_1 contains the union of the schemas at all sites and $\Sigma_1 = \Sigma_2$, then Example 3 shows that we need to consider at least two candidates for evaluation: q' and q itself. As the configuration of the system grows larger (e.g., additional databases, cached queries, materialized views, etc.), the number of equivalent rewritings increases as well (as we shall also see in a later example). This increases the potential for improvement in performance but at the same time poses the challenge of finding such reformulations in a systematic and complete way.

Section 4 describes how the C&B algorithm can be used for systematic enumeration of available reformulations. This enumeration is based on constraints such as the ones described above. In Section 5 we modify the running example by adding materialized views (one in the example). We then describe how the same C&B algorithm is able to find extra, view-based reformulations, by using additional constraints describing the views.

4 The C&B Algorithm

Given a conjunctive query q and a set $\mathcal{C}$ of constraints, the C&B algorithm finds reformulations q' of q under $\mathcal{C}$ (i.e. $q' \equiv_{\mathcal{C}} q$) which are *minimal under* $\mathcal{C}$ (in short $\mathcal{C}$*-minimal*). The notion of minimality of a conjunctive query in the absence of constraints is well-known [9, 2]: q is minimal if dropping any of its atoms compromises equivalence to q. For minimality under constraints, we require a subtle modification:

Definition 1 (Minimality under constraints) *A conjunctive query q is $\mathcal{C}$-minimal if there are no queries s_1, s_2 where s_1 is obtained from q by replacing zero or more variables with other variables of q, and s_2 by dropping at least one atom from s_1 such that s_2 and s_1 remain equivalent to q: $q \equiv_{\mathcal{C}} s_1 \equiv_{\mathcal{C}} s_2$.*

Intuitively, the variable replacement reflects the equalities between replaced and replacing variables as implied by the equality-generating dependencies (egds) in $\mathcal{C}$.

Example 4

$$\begin{aligned} q_{nm}(cn, cn') &:- Cust(c, cn), MasterCust(c, cn, ca), \\ &\quad Cust(c, cn'), MasterCust(c, cn', ca') \\ s_1(cn, cn) &:- Cust(c, cn), MasterCust(c, cn, ca), \\ &\quad MasterCust(c, cn, ca') \\ s_2(cn, cn) &:- Cust(c, cn), MasterCust(c, cn, ca) \end{aligned}$$

The query q_{nm} above yields pairs of nations of customers listed with the same customer id. The query is minimal in the absence of constraints: we cannot drop any atom, as proven by the absence of containment mappings. However, for constraint e from Example 2, q_{nm} is *not* $\{e\}$-minimal, as witnessed by queries s_1 and s_2 above. Intuitively, replacing cn' with cn preserves $\{e\}$-equivalence of s_1 to q_{nm}, since $cn = cn'$ is implied by e (the duplicate atom *Cust(c,cn)* is removed). It is easy to check that the removal of the second *MasterCust* atom preserves equivalence to s_1 even in the absence of constraints.

As illustrated in Figure 1, the C&B algorithm proceeds in two phases. In the **chase phase**, the original query q is chased with the constraints in $\mathcal{C}$, yielding the query U called a *universal plan*. The **backchase phase** enumerates all $\mathcal{C}$-minimal subqueries sq of U which are formulated against Σ_2 and are $\mathcal{C}$-equivalent to q ($sq \equiv_{\mathcal{C}} q$). (A *subquery* is obtained by dropping one or more atoms in the original query with the condition that the head variables continue to appear in the new query body.)

Example 5 (Chase) Recall the query q from Example 3:

$$\begin{aligned} q(p, c, sa, sn) &:- WebOrder(p, s, o, c, q), \\ &\quad SuppCatalog(s, sa, sn, d) \end{aligned}$$

A chase step of q with f_1 yields

$$\begin{aligned} q_1(p, c, sa, sn) &:- WebOrder(p, s, o, c, q), Cust(c, cn), \\ &\quad SuppCatalog(s, sa, sn, d) \end{aligned}$$

which chases with m_1 to

$$\begin{aligned} U(p, c, sa, sn) &:- WebOrder(p, s, o, c, q), Cust(c, cn), \\ &\quad SuppCatalog(s, sa, sn, d), \\ &\quad MasterSupp(s, sa, sn, h), \\ &\quad Supp2Cust(s, o, c), \\ &\quad MasterCust(c, cn, ca) \end{aligned}$$

which is the universal plan since no further chase step applies.

For each subquery sq of U, to check $sq \equiv_{\mathcal{C}} q$ it suffices to check $sq \subseteq_{\mathcal{C}} q$. Indeed, by construction U is contained in sq, $U \subseteq sq$, and $U \equiv_{\mathcal{C}} q$ because the chase preserves equivalence under constraints [2]. Checking $sq \subseteq_{\mathcal{C}} q$ reduces according to classical results to finding a containment mapping from q into the result of chasing sq with $\mathcal{C}$ [2]. Finding a containment mapping into an *intermediate* result of chasing also suffices to show containment.

Pruning Property. An immediate yet naive backchase implementation would exhaustively enumerate all subqueries. We can however avoid this by using the following key observation (called the *pruning property*) which follows from the definition of $\mathcal{C}$-minimality: given a subquery sq of U with $sq \equiv_{\mathcal{C}} q$, every subquery sq' of U corresponding to a superset of sq's atoms (we say that sq' is a *superquery* of sq) cannot be both $\mathcal{C}$-equivalent to q and $\mathcal{C}$-minimal.

The pruning property enables an efficient backchase implementation which enumerates subqueries of U *bottom-up*, starting with all subqueries generated by one atom of U, continuing with those generated by all pairs of atoms, then all triplets, and so on. As soon as a subquery is $\mathcal{C}$-equivalent to q, it is output and all its superqueries are pruned from subsequent consideration. This enumeration discipline avoids even generating non-minimal reformulations. We will discuss alternate backchase implementations shortly.

Example 6 (Backchase) The rewriting q' of q from Example 3 corresponds to the subquery of U generated by the *WebOrder* and *MasterSupp* atoms. Since q' has no smaller subquery, it is one of the starting points in the bottom-up backchase. To check $q' \subseteq_{\mathcal{C}} q$, we chase q' with $\mathcal{C}$. A first chase step with constraint m_2 yields

$$\begin{aligned} q_1'(p, c, sa, sn) \quad :- \quad & WebOrder(p, s, o, c, q), \\ & MasterSupp(s, sa, sn, h), \\ & SuppCatalog(s, sa, sn, d) \end{aligned}$$

Although we could continue to chase with f_1 and then m_1, it is not necessary. We can already find a containment mapping from q to q_1' (the identity mapping), thus proving that q' is equivalent under $\mathcal{C}$ to q. It can be checked that q' is $\mathcal{C}$-minimal. In fact, it is a property of the C&B algorithm that it outputs only $\mathcal{C}$-minimal reformulations.

The backchase will enumerate additional subqueries and will possibly output other minimal reformulations. For instance, the subquery of U generated by its *WebOrder* and *SuppCatalog* atoms is q itself.

In the example, the retrieval of rewriting q' as a subquery of the universal plan is not merely a happy coincidence: by Theorem 1, we have that whenever the chase is guaranteed to terminate, all minimal reformulations of a query will be found by the C&B algorithm:

Theorem 2 (Sound and complete C&B [11, 13]) *Let q be a conjunctive query and $\mathcal{C}$ a set of tgds and egds such that the chase of any query with $\mathcal{C}$ terminates. Then the C&B algorithm outputs precisely all $\mathcal{C}$-minimal reformulations of q (up to isomorphism).*

The C&B algorithm relies on the termination of the chase. This property is in general undecidable for conjunctive queries and constraints given by tgds and egds. However, the notion of *weak acyclicity* of a set of constraints, is sufficient to guarantee that any chase sequence terminates. This is the least restrictive sufficient termination condition we are aware of, holding in all scenarios we encountered in practice (including our example).

Definition 2 (Weakly acyclic set of constraints) Let $\mathcal{C}$ be a set of tgds over a fixed schema. Construct a directed graph, called the *dependency graph*, as follows: (1) there is a node for every pair (R, A) with R a relation symbol of the schema and A an attribute of R; call such pair (R, A) a *position*; (2) add edges as follows: for every tgd $\phi(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y})$ in $\mathcal{C}$ and for every x in $\mathbf{x}$ that **occurs** in ψ:

- For every occurrence of x in ϕ in position (R, A_i):
  (a) for every occurrence of x in ψ in position (S, B_j), add an edge $(R, A_i) \rightarrow (S, B_j)$.
  (b) in addition, for every existentially quantified variable y and for every occurrence of y in ψ in position (T, C_k), add a *special edge* $(R, A_i) \stackrel{*}{\rightarrow} (T, C_k)$.

Note that there may be two edges in the same direction between two nodes, if exactly one of the two edges is special. Then $\mathcal{C}$ is *weakly acyclic* if the dependency graph has no cycle going through a special edge. We say that a set of tgds and egds is weakly acyclic if the set of all its tgds is weakly acyclic.

Theorem 3 ([17, 13]) *If $\mathcal{C}$ is a weakly acyclic set of tgds and egds, then the chase with $\mathcal{C}$ of any conjunctive query q terminates.*

By Theorems 3 and 2, the C&B algorithm is sound and complete for weakly acyclic sets of constraints:

Corollary 1 *If $\mathcal{C}$ is a weakly acyclic set of tgds and egds, then the C&B algorithm outputs precisely the $\mathcal{C}$-minimal reformulations of its input query.*

The complexity of the chase. For fixed schemas and set $\mathcal{C}$ of constraints, if $\mathcal{C}$ is weakly acyclic then any chase sequence terminates in polynomial time in the size of the query being chased (as shown in [17, 13]). The fixed size assumption about schemas and constraints is often justified in practice, where one is usually interested in repeatedly reformulating incoming queries for the same setting with schemas and constraints. Nonetheless, the degree of the polynomial depends on the size of the dependencies and care is needed to implement the chase efficiently. Successive implementations have shown that in practical situations the chase is eminently usable [29, 28, 10, 12].

The complexity of reformulation. Assume that the chase of any query with $\mathcal{C}$ terminates in polynomial time. Then checking whether a conjunctive query q admits a reformulation is NP-complete in the size of q. Checking whether a given query r is a $\mathcal{C}$-minimal reformulation of q is NP-complete in the sizes of q and r. Note that for arbitrary sets of dependencies (for which the chase may not even terminate), the above problems are undecidable.

Alternative strategies for backchase. The complexity of the backchase is an even more delicate issue since even though the size of the universal plan is (with weakly acyclic dependencies) polynomial in that of the original query, in the worst case the backchase enumerates exponentially many minimal solutions [29]. Above, we presented the backchase as a bottom-up procedure that generates subqueries of the universal plan U by starting from the smallest subqueries and extending them with additional atoms from U. The algorithm stops extending a subquery sq as soon as sq becomes *equivalent* to the original query q. Such sq is guaranteed to be a $\mathcal{C}$-minimal reformulation of q.

Symmetrically, another way of implementing the backchase minimization is a *top-down*, decremental, procedure that goes from the universal plan down to its subqueries by eliminating one relational atom at a time in a systematic way, starting at every step a new branch per available atom. The algorithm stops descending on a branch whenever a *non-equivalent* subquery is found. The last equivalent query on that branch is a $\mathcal{C}$-minimal reformulation.

The top-down and the bottom-up algorithms are dual and produce the exact same output. However, the bottom-up ap-

proach has a crucial advantage in that it can be mixed with *cost-based pruning* (when cost information is available). The resulting procedure is as follows. When we find the first minimal reformulation, we estimate its cost (for example, based on traditional methods that include join reordering). This cost becomes the *best cost* so far and it will be subsequently replaced by the cost of every minimal reformulation that we may find later. Once the best cost is in place, for every explored subquery, even before checking for equivalence, we compute its cost. If the cost is higher than the best cost so far, then we can prune the subquery *together with all of its superqueries* without checking equivalence. This pruning still guarantees that the least-cost reformulation is found under the (typically true[2]) assumption that queries become more expensive by adding extra atoms (joins). The improvement in performance of the overall method is then substantial, sometimes over an order of magnitude [28]. Bottom-up backchase minimization with cost-based pruning is further extended in [28] to deal also with cases in which the above assumption may be violated (for example, due to the presence of indexes). Additional exploration strategies for subqueries of the universal plan are investigated in [29]. There it was shown that in various practical situations, the C&B method can be *stratified*, which means essentially, that the universal plan can be decomposed into independent fragments (smaller universal plans). For each fragment the backchase minimization is applied in the usual way. The minimal reformulations that result for each fragment can then be put together, by joining, as minimal reformulations for the entire process. The net effect is a significant reduction in the exponent of the search space, and hence considerable improvement in the performance of the method.

5 Adding Views

We show next that materialized views defined by conjunctive queries can be captured using tgds, and hence the C&B algorithm serves in particular as a complete algorithm which finds all minimal rewritings of a conjunctive query using conjunctive query views under integrity constraints. All we need to do is add the constraints (tgds) capturing the views, and reformulate the query against a schema containing the view names.

In detail, let $\mathcal{V}$ be a set of views defined by conjunctive queries against Σ_1. The views define a relationship between schemas Σ_1 and the schema $\mathcal{V}$, in which each view name V denotes the table with the materialized result of the homonymous view. We express this relationship equivalently using the set of dependencies $\mathcal{C}_\mathcal{V}$ constructed as follows. For each view $V \in \mathcal{V}$, assume w.l.o.g. that it is defined by the query

$$V(\mathbf{x}) :- body(\mathbf{x}, \mathbf{y})$$

where *body* is a conjunctive query body and $\mathbf{x}, \mathbf{y}$ are its variables. Let d_V^1, d_V^2 be the dependencies (over schema $\Sigma_1 \cup \mathcal{V}$):

$$d_V^1 :\ body(\mathbf{x}, \mathbf{y}) \rightarrow V(\mathbf{x}) \qquad d_V^2 :\ V(\mathbf{x}) \rightarrow \exists \mathbf{y}\ body(\mathbf{x}, \mathbf{y})$$

[2] And in fact, the very idea of minimization is based on such assumption.

which state the inclusions between the result of the query defining the view, and the materialized table V. Set

$$\mathcal{C}_\mathcal{V} := \{d_V^1 \mid V \in \mathcal{V}\} \cup \{d_V^2 \mid V \in \mathcal{V}\}.$$

We consider two flavors of rewriting using views: rewritings using exclusively the views (also called *total* rewritings in [25]), and rewritings using both the views and the base tables in Σ_1 (called *partial* rewritings in [25]). Thus, given conjunctive Σ_1-query q and set of constraints $\mathcal{C}$ over Σ_1, the problem of finding all total conjunctive query rewritings of q reduces to finding all minimal reformulations of q against schema $\Sigma_2 := \mathcal{V}$ under constraints $\mathcal{C} \cup \mathcal{C}_\mathcal{V}$. For partial rewritings, we set $\Sigma_2 := \Sigma_1 \cup \mathcal{V}$. According to Theorem 2, both flavors are completely solved by the C&B algorithm.

Example 7 Continuing our example, consider the following query launched at Site 1, which retrieves all parts provided by Japanese suppliers and ordered by US customers.

$$\begin{aligned} j2us(p) \quad :- \quad & WebOrder(p, s, o, c, q), Cust(c, \text{“}US\text{”}), \\ & SuppCatalog(s, sa, \text{“}Japan\text{”}, d) \end{aligned}$$

In general, the query would need to access Sites 1 and 2. Assume however that the previously answered query q from Example 3 is cached at Site 1, in cache entry $cache_q$. Then $j2us$ has a partial rewriting which reuses the pre-computed join of $WebOrder$ and $SuppCatalog$, performing only the remaining join between the cache entry and the customer table, both located at Site 1:

$$j2us'(p) :- cache_q(p, c, sa, \text{“}Japan\text{”}), Cust(c, \text{“}US\text{”})$$

This is more efficient as it avoids network access to Site 2, and saves the time to recompute the join of $WebOrder$ and $SuppCatalog$.

The C&B algorithm discovers this rewriting when called with $\Sigma_2 := \Sigma_1 \cup \mathcal{V}$ and $\mathcal{C} \cup \mathcal{C}_\mathcal{V}$, where $\mathcal{V}$ contains the names of all active cache entries, Σ_1 is the union of the schemas at all sites, and $\mathcal{C}_\mathcal{V}$ is constructed as described above. For instance, entry $cache_q$ can be seen as the materialized view

$$\begin{aligned} cache_q(p, c, sa, sn) \quad :- \quad & WebOrder(p, s, o, c, q), \\ & SuppCatalog(s, sa, sn, d) \end{aligned}$$

and $\mathcal{C}_\mathcal{V}$ includes the constraints:

$$\begin{aligned} d^1_{cache_q} &: WebOrder(p, s, o, c, q) \wedge SuppCatalog(s, sa, sn, d) \\ &\qquad \rightarrow cache_q(p, c, sa, sn) \\ d^2_{cache_q} &: cache_q(p, c, sa, sn) \rightarrow \exists s \exists o \exists q \exists d \\ &\qquad WebOrder(p, s, o, c, q) \\ &\qquad \wedge SuppCatalog(s, sa, sn, d) \end{aligned}$$

Now $j2us$ chases with m_1, then $d^1_{cache_q}$, to

$$\begin{aligned} j2us_1(p) \quad :- \quad & WebOrder(p, s, o, c, q), Cust(c, \text{“}US\text{”}), \\ & SuppCatalog(s, sa, \text{“}Japan\text{”}, d), \\ & MasterSupp(s, sa, \text{“}Japan\text{”}, h), \\ & Supp2Cust(s, o, c), \\ & MasterCust(c, \text{“}US\text{”}, ca), \\ & cache_q(p, c, sa, \text{“}Japan\text{”}) \end{aligned}$$

This is the universal plan, and it contains the equivalent (and minimal) rewriting $j2us'$ as the subquery given by the second and last atoms. In fact, the universal plan includes even more. The following two subqueries of the universal plan are also $\mathcal{C}$-minimal reformulations of $j2us$:

$$\begin{aligned} j2us''(p) \quad &:- \quad WebOrder(p,s,o,c,q), Cust(c, \text{“}US\text{”}), \\ &\qquad MasterSupp(s, sa, \text{“}Japan\text{”}, h) \\ j2us'''(p) \quad &:- \quad WebOrder(p,s,o,c,q), \\ &\qquad MasterSupp(s, sa, \text{“}Japan\text{”}, h), \\ &\qquad MasterCust(c, \text{“}US\text{”}, ca) \end{aligned}$$

While the first of the two rewritings above is similar to the earlier rewriting q' (Example 3), the second of the two rewritings is slightly different and less obvious. Its equivalence to the original query (which can be proven by chasing) depends essentially on the existence of several of the constraints in the system (specifically, m_2, f_1, and even the egd e). The last two reformulations do not include the view ($cache_q$) but they can be equally good candidates for execution.

This example shows the versatility of the C&B method as a rewriting tool that unifies several different concepts (e.g., views, constraints, mappings) under one umbrella, that of rewriting under constraints.

6 Other Considerations

Dictionaries. An interesting property of the query and dependency languages used in [30, 11] is the use of *dictionary* structures. In conjunction with complex values, dictionaries can be used (see [11]) to model OO classes with extents, primary and secondary indexes on either relations or class extents and gmaps [33]. On one hand this allows one to express and optimize arbitrary OQL queries [8]. On the other hand, the explicit presence of indexes allows an optimizer that uses C&B to automatically discover non-trivial execution plans that would not be found by traditional optimizers (including the ones that perform rewriting using materialized views [21]).

More expressive queries and constraints. For simplicity, we have presented the C&B only for conjunctive queries and constraints given by tgds and egds. However, the soundness and completeness of the C&B carries over to unions of conjunctive queries with inequalities, and tgds and egds extended with disjunction and inequalities (using an appropriate generalization of the chase) [10, 13, 14].

XML query reformulation. We were able to apply the C&B method to XML query reformulation, by using a relational encoding of queries, views and constraints that are originally written against a schema which models the XML tree. Relationships between XML elements (such as parent-child and ancestor-descendant) are captured by relational tables satisfying certain constraints (e.g. each child has at most one parent, the descendant table is transitive, etc.). We could show that for a significant class of XML queries, the minimal reformulations are found by running the C&B algorithm on the relational encoding [10, 13, 14]. The encoding turned out to lead to queries and universal plans of significantly larger size than encountered in any real-life relational scenarios (as a typical data point, universal plans of 300 atoms were obtained by chasing queries of 20 atoms). Both the chase and the backchase implementation were engineered to scale, and the feasibility of the method was proven in a battery of experiments [10, 12].

Relationship to data integration and non-equivalent rewritings. The C&B algorithm looks for rewritings that are equivalent (retrieve the same answers as the original query). Under this semantics, it is more general than previously known algorithms for rewriting using views, because it additionally takes into account general constraints (tgds and egds). In many integration scenarios however, there is no equivalent rewriting and one is content to approximate the original query q by finding a *maximally-contained rewriting*. Significant research has been carried out on algorithms which find such rewritings for conjunctive queries. Contained rewritings are unions of conjunctive queries expressed exclusively in terms of the views and contained in q [15, 21]. Maximally-contained rewritings are contained rewritings which contain any other contained rewriting of q, thus being the best “under-approximation” of q using the views. The problem was generalized in [7] to replace views with schema mapping constraints from the source schema to the target schema, also allowing constraints on the target schema. [22] generalizes the setting even further, allowing schema mappings in both directions, and settling the problem by charting its decidability boundaries and providing tight complexity bounds.

Although in its basic form the C&B algorithm returns only *equivalent* rewritings, it turns out that a simplified version acts as a dual to the algorithms for finding maximally-contained rewritings [21], by providing an alternate approximation: the *minimally-containing rewriting*. A containing rewriting of q is a conjunctive query against the views which contains q. A minimally-containing rewriting is a containing rewriting which is contained in any other containing rewriting of q. It is thus the best “over-approximation” of q. The simplified algorithm is the following:

1. Chase q and obtain the universal plan U.
2. Restrict the body of U only to the vocabulary of views, obtaining a query M.
3. If M is safe (i.e., its head variables appear in the body), output M, otherwise output “no containing rewriting of q exists”.

This simplification of C&B skips the backchase minimization stage. The following result states that the algorithm is sound and complete for finding the minimally-containing rewriting, which is unique up to equivalence:

Theorem 4 *Assume that the chase of q terminates. Then q admits a minimally-containing rewriting if and only if the simplified C&B algorithm outputs such a rewriting. Moreover, the minimally-containing rewriting is unique up to equivalence.*

Relationship to data exchange There are several interesting parallels (and differences) between the C&B method and the formalism for data exchange that was developed in [18, 19].

The data exchange problem is the problem of materializing an instance of a target schema based on an instance of a source schema, and based on a set of source-to-target constraints, representing the mapping between the two schemas.

First of all, both methods make use of the chase in a fundamental way. The C&B method applies the chase to construct the universal plan, while in data exchange, the chase is applied on the source instance to construct a *universal solution*. Philosophically, the concepts of universal plan and universal solution are somewhat similar and play equally important roles. The universal plan defines the space of all minimal reformulations while the universal solution is the "best" representative for the space of all possible target instances (or, solutions).

Second, both methods use minimization: in C&B, to generate all the minimal reformulations, in data exchange, to compute the smallest universal solution (the *core* of the universal solutions [19]). In C&B, minimization is performed under constraints and we look for multiple and non-isomorphic reformulations that are minimal under constraints. In contrast, in data exchange there is only one core of the universal solutions (up to isomorphism). This core is defined independently of the constraints and represents the minimal form of a universal solution, under homomorphisms which preserve the values that appear in the source instance. Finally, another (important) difference is the complexity of the minimization process in the two cases. In data exchange, computing the core of the universal solutions has polynomial-time algorithms in several cases of practical relevance [19, 20]. In the more general setting of C&B, minimization is exponential (NP-hard even without constraints, when it becomes tableau minimization [9]).

7 Conclusion

Many classical database problems such as semantic optimization (i.e. rewriting using semantic constraints), minimization, rewriting using views, equivalent query reformulation in data publishing and integration, are particular instances of query reformulation under constraints. While the general reformulation problem is undecidable, the least restrictive known conditions which are sufficient to guarantee decidability (namely weak acyclicity of the constraint set) hold in numerous practical scenarios. Under these conditions, C&B is a sound and complete algorithm, thus providing a uniform solution to the above problems (with applicability to object-oriented and XML settings). Our experiments show that, with careful engineering of the chase and backchase phases, the C&B method is viable in practice. An online demo of the C&B method can be found at `http://cb.ucsd.edu`.

References

[1] S. Abiteboul and O. M. Duschka. Complexity of Answering Queries Using Materialized Views. In *PODS*, pages 254–263, 1998.

[2] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[3] S. Adali, K. S. Candan, Y. Papakonstantinou, and V. Subrahmanian. Query Caching and Optimization in Distributed Mediator Systems. In *SIGMOD*, pages 137–148, 1996.

[4] C. Beeri and Y. Kornatzky. Algebraic Optimisation of Object Oriented Query Languages. *TCS*, 116(1):59–94, 1993.

[5] C. Beeri and M. Y. Vardi. A Proof Procedure for Data Dependencies. *JACM*, 31(4):718–741, 1984.

[6] P. Buneman, S. Naqvi, V. Tannen, and L. Wong. Principles of Programming with Collection Types. *TCS*, 149(1):3–48, 1995.

[7] A. Calì, D. Calvanese, G. D. Giacomo, and M. Lenzerini. Data Integration under Integrity Constraints. In *CAiSE*, pages 262–279, 2002.

[8] R. G. G. Cattell, D. K. Barry, M. Berler, J. Eastman, D. Jordan, C. Russell, O. Schadow, T. Stanienda, and F. Velez, editors. *The Object Data Standard: ODMG 3.0*. Morgan Kaufmann, 2000.

[9] A. K. Chandra and P. M. Merlin. Optimal Implementation of Conjunctive Queries in Relational Data Bases. In *STOC*, pages 77–90, 1977.

[10] A. Deutsch. *XML Query Reformulation Over Mixed and Redundant Storage*. PhD thesis, Dept. of Computer and Information Sciences, University of Pennsylvania, 2002.

[11] A. Deutsch, L. Popa, and V. Tannen. Physical Data Independence, Constraints and Optimization with Universal Plans. In *VLDB*, pages 459–470, 1999.

[12] A. Deutsch and V. Tannen. MARS: A System for Publishing XML from Mixed and Redundant Storage. In *VLDB*, pages 201–212, 2003.

[13] A. Deutsch and V. Tannen. Reformulation of XML Queries and Constraints. In *ICDT*, pages 225–241, 2003.

[14] A. Deutsch and V. Tannen. XML Queries and Constraints, Containment and Reformulation. *TCS*, 336(1):57–87, 2005.

[15] O. M. Duschka and M. R. Genesereth. Answering recursive queries using views. In *PODS*, pages 109–116, 1997.

[16] R. Fagin. Horn Clauses and Database Dependencies. *JACM*, 29(4):952–985, Oct. 1982.

[17] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. In *ICDT*, pages 207–224, 2003.

[18] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. *TCS*, 336(1):89–124, 2005.

[19] R. Fagin, P. G. Kolaitis, and L. Popa. Data Exchange: Getting to the Core. *ACM TODS*, 30(1):174–210, 2005.

[20] G. Gottlob. Computing Cores for Data Exchange: New Algorithms and Practical Solutions. In *PODS*, 2005.

[21] A. Halevy. Answering Queries Using Views: A Survey. *VLDB Journal*, pages 270–294, 2001.

[22] C. Koch. Query Rewriting with Symmetric Constraints. In *Proceedings of FoIKS (LNCS 2284)*, pages 130–147, 2002.

[23] K. Lellahi and V. Tannen. A calculus for collections and aggregates. In E. Moggi and G. Rosolini, editors, *LNCS 1290: Category Theory and Computer Science (Proceedings of CTCS'97)*, pages 261–280, 1997.

[24] M. Lenzerini. Data Integration: A Theoretical Perspective. In *PODS*, pages 233–246, 2002.

[25] A. Y. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering Queries Using Views. In *PODS*, pages 95–104, 1995.

[26] A. Y. Levy, A. Rajamaran, and J. J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *VLDB*, pages 251–262, 1996.

[27] D. Maier, A. O. Mendelzon, and Y. Sagiv. Testing Implications of Data Dependencies. *ACM TODS*, 4(4):455–469, 1979.

[28] L. Popa. *Object/Relational Query Optimization with Chase and Backchase*. PhD thesis, Dept. of Computer and Information Sciences, University of Pennsylvania, 2000.

[29] L. Popa, A. Deutsch, A. Sahuguet, and V. Tannen. A Chase Too Far? In *SIGMOD*, pages 273–284, 2000.

[30] L. Popa and V. Tannen. An Equational Chase for Path-Conjunctive Queries, Constraints, and Views. In *ICDT*, pages 39–57, 1999.

[31] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernandez, and R. Fagin. Translating Web Data. In *VLDB*, pages 598–609, 2002.

[32] J. Shanmugasundaram, J. Kiernan, E. Shekita, C. Fan, and J. Funderburk. Querying XML Views of Relational Data. In *VLDB*, pages 261–270, 2001.

[33] O. Tsatalos, M. Solomon, and Y. Ioannidis. The GMAP: A Versatile Tool for Physical Data Independence. *VLDB Journal*, 5(2):101–118, 1996.

A Some Definitions

We review the standard definitions of conjunctive queries, homomorphisms, containment mappings and chase.

A *conjunctive query* q over a schema Σ is an expression of the form $q(\mathbf{x}) :- \phi(\mathbf{x}, \mathbf{y})$ where $\phi(\mathbf{x}, \mathbf{y})$ is a conjunction of atomic formulas (i.e., relational atoms, also called *subgoals*) over Σ. We follow the usual notation and separate the atoms in a query by commas. We call $q(\mathbf{x})$ the *head* and $\phi(\mathbf{x}, \mathbf{y})$ the *body*. We use a notation such as $\mathbf{x}$ for a vector of variables $x_1, \ldots, x_k$ (not necessarily distinct). Every variable in the head must appear in the body(i.e., the query must be *safe*). The set of variables in $\mathbf{y}$ is assumed to be existentially quantified.

Given two conjunctions $\phi(\mathbf{u})$ and $\psi(\mathbf{v})$ of atomic formulas, a *homomorphism* from $\phi(\mathbf{u})$ to $\psi(\mathbf{v})$ is a mapping h from the set of variables in $\mathbf{u}$ to the set of variables in $\mathbf{v}$ such that for every atom $R(u_1, \ldots, u_n)$ of ϕ, the atom $R(h(u_1), \ldots, h(u_n))$ is in ψ. Given two conjunctive queries $q_1(\mathbf{x}) :- \phi(\mathbf{x}, \mathbf{y})$ and $q_2(\mathbf{x}') :- \psi(\mathbf{x}', \mathbf{y}')$, a *containment mapping* from q_1 to q_2 is a homomorphism h from $\phi(\mathbf{x}, \mathbf{y})$ to $\psi(\mathbf{x}', \mathbf{y}')$ such that $h(\mathbf{x}) = \mathbf{x}'$. A classical result [9] states that a necessary and sufficient condition for the containment (under all instances) of a conjunctive query q_1 into a conjunctive query q_2 is the existence of a containment mapping from q_2 to q_1.

Assume a conjunctive query $q(\mathbf{x}) :- \phi(\mathbf{x}, \mathbf{y})$ and a tgd t of the form $\forall \mathbf{u}(\alpha(\mathbf{u}) \rightarrow \exists \mathbf{v} \beta(\mathbf{u}, \mathbf{v}))$. Assume without loss of generality that $\mathbf{v}$ and the query have no variables in common. The chase of q with t is applicable if there is a homomorphism h from $\alpha(\mathbf{u})$ to the body of q, and moreover, if h cannot be extended to a homomorphism h' from $\alpha(\mathbf{u}) \wedge \beta(\mathbf{u}, \mathbf{v})$ to the body of q. In that case, a *chase step* of q with t and h is a rewrite of q into $q'(\mathbf{x}) :- \phi(\mathbf{x}, \mathbf{y}) \wedge \beta(h(\mathbf{u}), \mathbf{v})$.

Similarly, we can define a chase step with an egd. Assume a conjunctive query q as before and an egd e of the form $\forall \mathbf{u}(\alpha(\mathbf{u}) \rightarrow (u_1 = u_2))$. The chase of q with e is applicable if there is a homomorphism h from $\alpha(\mathbf{u})$ to $\phi(\mathbf{x}, \mathbf{y})$ so that $h(u_1)$ and $h(u_2)$ are not the same variable. In that case, a *chase step* of q with e and h is a rewrite of q into a query q' which is the same as q except that all occurrences of the variable $h(u_1)$ (in the head and in the body) are replaced by the variable $h(u_2)$.

Computing Reviews

WWW.REVIEWS.COM

BECOME A REVIEWER FOR COMPUTING REVIEWS

Computing Reviews is the authoritative publication of reviews in computing literature, and we're inviting you to apply to become a reviewer.

As a reviewer, you will communicate your expertise and insight to Computing Reviews' readers – hundreds of thousands of academics and professionals in universities and corporate research facilities worldwide. With reviews published daily online and monthly on paper, Computing Reviews tracks the latest developments and discoveries across all areas of computer science, and gives its readers the overview needed to identify the most essential books and articles.

To apply to become a reviewer, go to www.reviews.com/reviewer, then click on Become a Reviewer. At Computing Reviews, we are committed to excellence. Our reviewers are authorities in their fields, and, through their reviews, they provide the timely commentary needed to find out what is new and worth reading. Our Editorial Board evaluates and approves reviewers on numerous criteria, including educational background, technical knowledge and professional experience.

Computing Reviews is a collaboration between the Association for Computing Machinery and Reviews.com and can be read daily at www.reviews.com.

Association for Computing Machinery

Reviews.com