# Table of Contents

Proceedings of the Twenty-Second Australasian Database Conference (ADC 2011), Perth, Australia, 17-20 January 2011

Preface .................................................................................................................. vii
Programme Committee ....................................................................................... viii
Organising Committee ......................................................................................... x
Welcome from the Organising Committee ....................................................... xi
CORE - Computing Research & Education ....................................................... xii
ACSW Conferences and the Australian Computer Science Communications ................................................................. xiii
ACSW and ADC 2011 Sponsors ........................................................................... xv

## Invited Papers

Processing Spatio-Temporal Queries in a Streaming Fashion ........................... 3
  
  Rai Zhang

## Contributed Papers

Semi-Skyline Optimization of Constrained Skyline Queries .......................... 7
  
  Markus Endres and Werner Kießling

Schema-less XML in Columns ........................................................................... 17
  
  Zuzana Částková and Jaroslav Pokorny

Analyzing and Improving Table Space Allocation ......................................... 27
  
  Hong Min, Hubertus Franke, Bala Iyer, Frances Villafrue and Julie Watts

Potentiality of Power Management on Database Systems with Power Saving Function of Disk Drives .......................... 37
  
  Norifumi Nishikawa, Miyuki Nakano and Masaru Kitsuregawa

Optimizing Queries for Web Generated Sensor Data .................................... 47
  
  Mark Roantree, Gerard Marks and Dominick Smyth

User Preference Representation Based on Psychometric Models .................. 57
  
  Biyun Hu, Zhoujun Li, Wen Han Chao, Xia Hu and Jun Wang

A System for Managing Data Provenance in In Silico Experiments ............... 65
  
  Jarrod Trevathan, Ian Atkinson, Wayne Read, Nigel Sim and Chris Christensen

Privacy and Anonymity in Untrusted Data Stores ......................................... 75
  
  Jarrod Trevathan, Wayne Read, Hossein Ghodosi and Ian Atkinson

An Empirical Study of Learning from Imbalanced Data ............................... 85
  
  Xiuzhen Zhang and Yuxuan Li
Effective Scheduling Algorithm for On-Demand XML Data Broadcasts in Wireless Environments

Yongrui Qin, Hua Wang and Jitian Xiao

PartSS: An Efficient Partition-based Filtering for Edit Distance Constraints

Zhixu Li, Laurianne Sitbon and Xiaofang Zhou

A Triangular Decomposition Access Method for Temporal Data - TD-tree

Bela Stantic

Accelerating Spatial Join Operations using Bit-Indices

Elizabeth Antoine, Kotagiri Ramamohanarao, Jie Shao and Rui Zhang

Scheduling with Freshness and Performance Guarantees for Web Applications in the Cloud

Yingying Zhu, Mohamed Sharaf and Xiaofang Zhou

Discovering Conditional Functional Dependencies in XML Data

Loan T.H Vo, Jinli Cao and Wenny Rahayu

20 Years of Data Quality Research: Themes, Trends and Synergies

Shazia Sadiq, Naiem Yeganeh and Marta Indulska

Author Index
Preface

The series of Australasian Database Conference is an annual forum for exploring novel technical developments and applications of database systems. The 22nd Australasian Database Conference, ADC 2011, is held in Perth, Australia, as part of Australasian Computer Science Week.

ADC 2011 invited submissions of original contributions in all research areas of databases and its applications. The program committee received thirty six submissions of full research papers; each was thoroughly reviewed by at least three PC members or external reviewers. Sixteen papers have been selected for presentation at the conference. In addition, the program committee invited an active researcher Dr Rui Zhang for the traditional ADC invited talk.

The ADC PC chairs have also looked at all the accepted papers to select a paper to be awarded the conference’s Best Paper. This year’s Best Paper award goes to the following paper “Potentiality of Power Management on Database Systems with Power Saving Function of Disk Drives” by Norifumi Nishikawa, Miyuki Nakano and Masaru Kitsuregawa. Congratulations to the above authors!

We would like to take this opportunity to thank all the authors who submitted papers and conference participants for the fruitful discussions. We are grateful to the members of the program committee and external referees for their timely expertise and effort in carefully reviewing the papers.

Heng Tao Shen  
University of Queensland  

Yanchun Zhang  
Victoria University  

ADC 2011 Programme Chairs  
January 2011
Programme Committee and Review Panel

Program Chairs

Heng Tao Shen (University of Queensland, Australia)
Athman Bouguettaya (CSIRO, Australia)

Programme Committee

Walid Aref (University of Purdue, USA)
Boualem Benatallah (University of New South Wales, Australia)
K. Selcuk Candan (Arizona State University, USA)
Jinli Cao (La Trobe University, Australia)
Hong Cheng (Chinese University of Hong Kong, Hong Kong)
Dario Colazzo (LRI - Universite Paris Sud, France)
Gao Cong (Nanyang Technological University, Singapore)
Bin Cui (Peking University, China)
Emiran Curtmola (Teradata Corporation, USA)
Gill Dobbie (University of Auckland, New Zealand)
Xiaoyong Du (Renmin University, China)
Giovanna Guerrini (DISI - University of Genova, Italy)
Wook-Shin Han (Kyungpook National University, Korea)
Takahiro Hara (Osaka University, Japan)
Zi Huang (University of Queensland, Australia)
Yoshiharu Ishikawa (Nagoya University, Japan)
Mizuho Iwaihara (Kyoto University, Japan)
Jialie Shen (Singapore Management University, Singapore)
Sang-goo Lee (Seoul National University, Korea)
Xue Li (University of Queensland, Australia)
Guoliang Li (Tsinghua University, China)
Chengfei Liu (Swinburne University of Technology, Australia)
Zaki Malik (Wayne State University, USA)
Miyuki Nakano (Tokyo University, Japan)
Mourad Ouzzani (Purdue University, USA)
Hye-young Paik (University of New South Wales, Australia)
Uwe Roehm (University of Sydney, Australia)
Shazia Sadiq (University of Queensland, Australia)
Falk Scholer (RMIT, Australia)
Jie Shao (University of Melbourne, Australia)
Michael Sheng (CSIRO ICT Centre, Australia)
Bela Stantic (Griffith University, Australia)
Markus Stumptner (University of South Australia, Australia)
Egemen Tanin (University of Melbourne, Australia)
Zahir Tari (RMIT, Australia)
Wei Wang (University of New South Wales, Australia)
Guoren Wang (Northeast University, China)
Quanqing Xu (National University of Singapore, Singapore)
Jian Yang (Macquarie University, Australia)
Qi Yu (Rochester Institute of Technology, USA)
Rui Zhang (University of Melbourne, Australia)
Xiuzhen Zhang (RMIT University, Australia)
Shichao Zhang (University of Technology, Sydney, Australia)
Yu Zheng (Microsoft Research Asia, China)
Xiangyang Xue (Fudan University, China)
Additional Reviewers

Amal Alhosban
Jiajie Xu
Jianzhong Qi
John Shepherd

Khayyam Hashmi
Peter Darcy
Tao Liu
Organising Committee

Chair
Assoc. Prof. Mihai Lazarescu

Co-Chair
Assoc. Prof. Ling Li

Finance
Mary Simpson
Mary Mulligan

Catering and Booklet
Mary Mulligan
Dr. Patrick Puersum
Assoc. Prof. Mihai Lazarescu

Sponsorship and Web
Dr. Patrick Puersum
Dr. Aneesh Krishna

Registration
Mary Mulligan
Dr. Patrick Puersum

DVD and Signage
Dr. Patrick Puersum
Mary Mulligan

Venue
Dr. Mike Robey

Conference Bag
Dr. Sieteng Soh
Welcome from the Organising Committee

On behalf of the Australasian Computer Science Week 2011 (ACSW2011) Organising Committee, we welcome you to this year’s event hosted by Curtin University. Curtin University’s vision is to be an international leader shaping the future through its graduates and world class research. As Western Australia’s largest university, Curtin is leading the state in producing high quality ICT graduates. At Curtin Computing, we offer both world class courses and research. Our Computing courses cover three key areas in IT (Computer Science, Software Engineering and Information Technology), are based on the curricula recommendations of IEEE Computer Society and ACM, the largest IT professional associations in the world, and are accredited by the Australian Computer Society. Curtin Computing hosts a top level research institute (IMPCA) and offers world class facilities for large scale surveillance and pattern recognition.

We welcome delegates from over 18 countries, including Australia, New Zealand, USA, U.K., Italy, Japan, China, Canada, Germany, Spain, Pakistan, Austria, Ireland, South Africa, Taiwan and Thailand. We hope you will enjoy the experience of the ACSW 2011 event and get a chance to explore our wonderful city of Perth. Perth City Centre is located on the north bank of the Swan River and offers many fun activities and a wealth of shopping opportunities. For panoramic views of Perth and the river, one can visit Kings Park or enjoy a relaxing picnic in one of the many recreational areas of the park.

The Curtin University campus, the venue for ACSW2011, is located just under 10km from the Perth City Centre and is serviced by several Transperth bus routes that travel directly between Perth and Curtin University Bus Station, as well as several other routes connecting to nearby train services.

ACSW2011 consists of the following conferences:

– Australasian Computer Science Conference (ACSC) (Chaired by Mark Reynolds)
– Australasian Computing Education Conference (ACE) (Chaired by John Hamer and Michael de Raadt)
– Australasian Database Conference (ADC) (Chaired by Heng Tao Shen and Athman Bouguettaya)
– Australasian Information Security Conference (AISC) (Chaired by Colin Boyd and Josef Pieprzyk)
– Australasian User Interface Conference (AUIC) (Chaired by Christof Lutteroth)
– Australasian Symposium on Parallel and Distributed Computing (AusPDC) (Chaired by Jinjun Chen and Rajiv Ranjan)
– Australasian Workshop on Health Informatics and Knowledge Management (HIKM) (Chaired by Kerryn Butler-Henderson and Tony Sahama)
– Computing: The Australasian Theory Symposium (CATS) (Chaired by Taso Viglas and Alex Potanin)
– Australasian Computing Doctoral Consortium (ACDC) (Chaired by Rachel Cardell-Oliver and Falk Scholer).

The nature of ACSW requires the co-operation of numerous people. We would like to thank all those who have worked to ensure the success of ACSW2011 including the Organising Committee, the Conference Chairs and Programme Committees, our sponsors, the keynote speakers and the delegates. Many thanks go to Alex Potanin for his extensive advice and assistance and Wayne Kelly (ACSW2010 chair) who provided us with a wealth of information on the running of the conference. ACSW2010 was a wonderful event and we hope we will live up to the expectations this year.

Assoc. Prof. Mihai Lazarescu and Assoc. Prof. Ling Li
Department of Computing, Curtin University
ACSW2011 Co-Chairs
January, 2011
CORE welcomes all delegates to ACSW2011 in Perth. CORE, the peak body representing academic computer science in Australia and New Zealand, is responsible for the annual ACSW series of meetings, which are a unique opportunity for our community to network and to discuss research and topics of mutual interest. The original component conferences ACSC, ADC, and CATS, which formed the basis of ACSW in the mid 1990s now share this week with six other events - ACE, AISC, AUIC, AusPDC, HIKM, ACDC, which build on the diversity of the Australasiain computing community.

In 2011, we have again chosen to feature a small number of plenary speakers from across the discipline: Heng To Shen, Gene Tsudik, ans Dexter Kozen. I thank them for their contributions to ACSW2011. I also thank the keynote speakers invited to some of the individual conferences. The efforts of the conference chairs and their program committees have led to strong programs in all the conferences again, thanks. And thanks are particularly due to Mihai Lazarescu and his colleagues for organising what promises to be a strong event.

In Australia, 2009 saw, for the first time in some years, an increase in the number of students choosing to study IT, and a welcome if small number of new academic appointments. Also welcome is the news that university and research funding is set to rise from 2011-12. However, it continues to be the case that place funding for computer science students has fallen relative to that of other physical and mathematical sciences, and, while bodies such as the Australian Council of Deans of ICT seek ways to increase student interest in the area, more is needed to ensure the growth of our discipline.

During 2010, CORE continued to negotiate with the ARC on journal and conference rankings. A key aim is now to maintain the rankings, which are widely used overseas as well as in Australia. Management of the rankings is a challenging process that needs to balance competing special interests as well as addressing the interests of the community as a whole.

COREs existence is due to the support of the member departments in Australia and New Zealand, and I thank them for their ongoing contributions, in commitment and in financial support. Finally, I am grateful to all those who gave their time to CORE in 2010; in particular, I thank Alex Potanin, Jenny Edwards, Alan Fekete, Aditya Ghose, Leon Sterling, and the members of the executive and of the curriculum and ranking committees.

Tom Gedeon
President, CORE
January, 2011
The Australasian Computer Science Week of conferences has been running in some form continuously since 1978. This makes it one of the longest running conferences in computer science. The proceedings of the week have been published as the *Australian Computer Science Communications* since 1979 (with the 1978 proceedings often referred to as Volume 0). Thus the sequence number of the Australasian Computer Science Conference is always one greater than the volume of the Communications. Below is a list of the conferences, their locations and hosts.

**2012.** Volume 34. Host and Venue - RMIT University, Melbourne, VIC.

**2011. Volume 33.** Host and Venue - Curtin University of Technology, Perth, WA.

**2010.** Volume 32. Host and Venue - Queensland University of Technology, Brisbane, QLD.

**2009.** Volume 31. Host and Venue - Victoria University, Wellington, New Zealand.

**2008.** Volume 30. Host and Venue - University of Wollongong, NSW.

**2007.** Volume 29. Host and Venue - University of Ballarat, VIC. First running of HDKM.

**2006.** Volume 28. Host and Venue - University of Tasmania, TAS.

**2005.** Volume 27. Host - University of Newcastle, NSW. APBC held separately from 2005.


**2002.** Volume 24. Host and Venue - Monash University, Melbourne, VIC.

**2001.** Volume 23. Hosts - Bond University and Griffith University (Gold Coast). Venue - Gold Coast, QLD.

**2000.** Volume 22. Hosts - Australian National University and University of Canberra. Venue - ANU, Canberra, ACT. First running of AUIC.


**1998.** Volume 20. Hosts - University of Western Australia, Murdoch University, Edith Cowan University and Curtin University. Venue - Perth, WA.


**1996.** Volume 18. Host - University of Melbourne and RMIT University. Venue - Melbourne, Australia. CATS joins ACSW.

**1995.** Volume 17. Hosts - Flinders University, University of Adelaide and University of South Australia. Venue - Glenelg, SA.


**1993.** Volume 15. Hosts - Griffith University and Queensland University of Technology. Venue - Nathan, QLD.

**1992.** Volume 14. Host and Venue - University of Tasmania, TAS. (ADC held separately at La Trobe University).

**1991.** Volume 13. Host and Venue - University of New South Wales, NSW.

**1990.** Volume 12. Host and Venue - Monash University, Melbourne, VIC. Joined by Database and Information Systems Conference which in 1992 became ADC (which stayed with ACSW) and ACIS (which now operates independently).

**1989.** Volume 11. Host and Venue - University of Wollongong, NSW.

**1988.** Volume 10. Host and Venue - University of Queensland, QLD.

**1987.** Volume 9. Host and Venue - Deakin University, VIC.

**1986.** Volume 8. Host and Venue - Australian National University, Canberra, ACT.


**1984.** Volume 6. Host and Venue - University of Adelaide, SA.

**1983.** Volume 5. Host and Venue - University of Sydney, NSW.

**1982.** Volume 4. Host and Venue - University of Western Australia, WA.

**1981.** Volume 3. Host and Venue - University of Queensland, QLD.

**1980.** Volume 2. Host and Venue - Australian National University, Canberra, ACT.

**1979.** Volume 1. Host and Venue - University of Tasmania, TAS.

**1978.** Volume 0. Host and Venue - University of New South Wales, NSW.
### Conference Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Conference Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACDC</td>
<td>Australasian Computing Doctoral Consortium</td>
</tr>
<tr>
<td>ACE</td>
<td>Australasian Computer Education Conference</td>
</tr>
<tr>
<td>ACSC</td>
<td>Australasian Computer Science Conference</td>
</tr>
<tr>
<td>ACSW</td>
<td>Australasian Computer Science Week</td>
</tr>
<tr>
<td>ADC</td>
<td>Australasian Database Conference</td>
</tr>
<tr>
<td>AISC</td>
<td>Australasian Information Security Conference</td>
</tr>
<tr>
<td>AUIC</td>
<td>Australasian User Interface Conference</td>
</tr>
<tr>
<td>APCCM</td>
<td>Asia-Pacific Conference on Conceptual Modelling</td>
</tr>
<tr>
<td>AusPDC</td>
<td>Australasian Symposium on Parallel and Distributed Computing (replaces AusGrid)</td>
</tr>
<tr>
<td>CATS</td>
<td>Computing: Australasian Theory Symposium</td>
</tr>
<tr>
<td>HIKM</td>
<td>Australasian Workshop on Health Informatics and Knowledge Management</td>
</tr>
</tbody>
</table>

Note that various name changes have occurred, which have been indicated in the Conference Acronyms sections in respective CRPIT volumes.
ACSW and ADC 2011 Sponsors

We wish to thank the following sponsors for their contribution towards this conference.

CORE - Computing Research and Education, www.core.edu.au


Australian Computer Society, www.acs.org.au

Curtin University of Technology, www.curtin.edu.au

Victoria University, www.csiro.au

University of Queensland, www.uq.edu.au
INVITED PAPERS
Processing Spatio-Temporal Queries in a Streaming Fashion

Rui Zhang

Department of Computer Science and Software Engineering
The University of Melbourne

Email: rui@csse.unimelb.edu.au

Abstract

In the past few years, the wide application of online map applications and location based services have significantly changed the ways of our life. Today, it is typical for us to search for directions or businesses on the Internet and find our ways through cell phones. Lots of the location based services require to provide answers to users continuously so that the users can interact with the applications. For example, a tourist may ask for the nearest three restaurants to be reported continuously while travelling in a city, so that he or she may choose to go to one at any time. As the answers may change over time, the continuous nature of the applications poses new challenges on methods to process the queries efficiently.

We look at a few key strategies to process spatio-temporal queries in a streaming fashion such as incremental and shared computation, safe regions, and time-constraint processing. We illustrate these strategies through the algorithms to several important types of continuous spatio-temporal queries.
Contributed Papers
Semi-Skyline Optimization of Constrained Skyline Queries

Markus Endres  Werner Kießling
Institute for Computer Science
University of Augsburg,
86135 Augsburg, Germany,
Email: {endres, kiessling}@informatik.uni-augsburg.de

Abstract
Skyline evaluation techniques (also known as Pareto preference queries) follow a common paradigm that eliminates data elements by finding other elements in a data set that dominate them. Nowadays already a variety of sophisticated skyline evaluation techniques are known, hence skylines are considered a well-researched area. On the other hand, the skyline operator does not stand alone in database queries. In particular, the skyline operator may commute with the selection operator which may express hard constraints on the skyline. In this paper, we address skyline queries that satisfy some hard constraints, so-called constrained skyline queries. We will present novel optimization techniques for such queries, which allow more efficient computation. For this, we propose semi-skylines which can be used effectively for algebraic optimizations of skyline queries having a mixture of hard constraints and soft preference conditions. All our efficiency claims are supported by a series of performance benchmarks.

Keywords: Constrained skyline queries, Skyline optimization, Semi-Skylines, Preference

1 Introduction

The skyline operator has emerged as an important and very popular summarization technique for multi-dimensional data sets. The skyline, or Pareto operator selects those objects from the database that are not dominated by any others. An object dominates another object, if it is as good or better in all dimensions and better in at least one dimension. Therefore, skyline queries have shifted retrieval models from exact matching of attribute values to the notion of best matching database objects.

But skyline queries may return too many objects if the dimensionality of the data space is large, or, the dimensions are anti-correlated. Therefore, one may put constraints on some dimensions (or attributes of a database relation) to express hard restrictions, i.e., one defines a skyline query that satisfies some hard constraints. These hard constraints could be expressed by the selection operator in a database query. There exists two different types of such queries in the literature: constrained skyline queries (Hafenrichter & Kießling 2005, Papadias et al. 2005, Dellis et al. 2006) and skyline queries with constraints (Zhang & Alhajj 2010). Given a set of hard constraints, a constrained skyline query returns the most interesting objects in the data space defined by these constraints. In contrary, skyline queries with constraints return the objects from the skyline restricted by the constraints. In general the result of skyline queries with constraints is different from the result reported by constrained skyline queries. Thus, an object in the former result is not necessarily an object in the latter.

Constrained skyline queries often occur in real-world applications, e.g., in a diet planning service. The United States Department of Agriculture (USDA, http://www.nal.usda.gov/fnic/) published a food database which contains nutritional facts for more than 7000 types of food. A user may be interested in finding meals that satisfy nutritional requirements such as a restriction on the number of calories (Cal), the amount of vitamin C (Vc), and the amount of total lipid fat (Fat). However, in the context of a diet, each user has preferences concerning its meal. Such a constrained preference query is shown in Figure 1, using the PreferenceSQL language from Kießling and Köstler (2002).

```
SELECT *
FROM Soup S, Meat M, Beverage B
WHERE S.Cal + M.Cal + B.Cal < 1100
AND S.Vc + M.Vc + B.Vc ≥ 38
AND S.Fat + M.Fat + B.Fat ≤ 9
AND S.Name IN ('Chicken', 'Noodle') AND M.Name IN ('Beef')
```

Figure 1: Sample Preference SQL query

In our running example a user expresses his or her preferences after the keyword PREFERENCES. It is a Pareto preference (AND in the PREFERENCES-clause) consisting of preferences on soups, meat, and beverages. The keyword IN denotes a preference for members of a given set, a POS-preference. Hence, the user prefers Chicken and Noodle soups over all others. Furthermore, the user wants Beef and a drink with a maximum of vitamin C (B.Vc HIGHEST). The preferences are evaluated after the selection operator as stated by constrained skyline queries, following the operational semantics of skyline queries from Börzsönyi et al. (2001) and Hafenrichter & Kießling (2005), where the skyline operator is logically applied after the selection operation and joins, respectively.

The result of a preference query consists of best matches only (BMO-set, (Kießling 2002)). Skyline queries are a special case of this BMO approach: Basically, they only allow HIGHEST (MAX) and LOWEST (MIN) preference constructors to participate in...
a Pareto preference (Kießling 2002). Therefore all results of this paper apply to skyline queries as well.

The optimization of constrained skyline queries as given in Figure 1 is essential to support fast result computation (Börzsönyi et al. 2001, Papadias et al. 2005, Hafnerichter & Kießling 2005, Zhang & Alhaji 2010). However, the optimization of such queries is far from being a trivial task. The query in Figure 1 contains hard constraints referring several relations and user preferences on some attributes. Conventional approaches implement such queries by a set of binary join operators and evaluate the hard constraints. Afterwards the user preferences as soft selection combined with the Pareto operator are evaluated by a skyline algorithm to retrieve the best matching objects. This can be seen in Figure 2.

![Figure 2: Unoptimized operator tree.](image)

In this operator tree \( \sigma[P] \) denotes Pareto, or skyline, evaluation and \( \bowtie_H \) the join with the hard constraints from the query in Figure 1. The query evaluation process must evaluate the cartesian product of all tuples of all relations (about 120-10^6 combinations using the USDA database), which leads to high memory and computation costs, particularly for large relations.

It would be attractive to apply the skyline operator before the selection operator, or the join, because the skyline operator reduces the size of the intermediate results. Therefore, a skyline operator before a join makes the join cheaper. Also, non-reductive joins tend to increase the size of intermediate results, so that the skyline operator itself becomes cheaper if it is pushed through the join. Furthermore, pushing the skyline operator through a join might make it possible to use an index to compute the skyline, as described by Börzsönyi et al. (2001).

In this paper we present semi-skylines which allow an algebraic optimization of constrained skyline queries. Semi-skylines eliminate tuples from relations which definitely can never be in the result set before the skyline join can be applied. Also, Raghavan and Rundensteiner (2010) consider such queries. They present a progressive algorithm by translating a tuple-level processing into a job-sequencing problem. Jin et al. (2007 and 2010) integrate different join methods into skyline computation. Jin et al. (2007) covers the skyline operator on joins and aggregates. In Jin et al. (2010) only equijoins are considered. Nevertheless, no tuple reduction can be done before the join is evaluated. Cui et al. (2008) propose the PaDSkyline algorithm in distributed environments. However, PaDSkyline does not address algebraic optimization issues, which are the main contribution of Semi-Skylines.

There are many publications on traditional database query optimization involving joins, but none of them deal with Skylines, e.g. (Agarwal et al. 1998, Ghu et al. 2003, Liu et al. 2005, Nestorov et al. 2007).

### 3 Semi-Skylines

Skyline queries, in particular preference queries and their integration into databases have been in focus for some time, leading to diverse approaches, e.g. (Kießling 2002, Chomicki 2003). We shortly review the preference model from Kießling (2002). Afterwards we define the Semi-Pareto preference which is the key idea of this paper.

#### 3.1 Preference Background

A preference \( P = (A, <_P) \), where \( A \) is a set of attributes, is a strict partial order on the domain of \( A \). The term \( x <_P y \) is interpreted as "\( x \) like \( y \) more than \( x \)". Two tuples \( x \) and \( y \) are indifferent, if \( -x <_P y \land -y <_P x \). The skyline of a preference \( P = (A, <_P) \) on an input relation \( R \) are all tuples that are not dominated w.r.t. the preference. It is computed by the preference selection operator \( \sigma[P](R) \) (called winnow by Chomicki (2003), BMO-set by Kießling (2002)):

\[
\sigma[P](R) := \{ t \in R \mid \exists t' \in R : t <_P t' \}
\]

It finds all best matching tuples \( t \) for the preference \( P \) with \( A \subseteq \text{attr}(R) \), where \( \text{attr}(R) \) denotes all attributes of a relation \( R \).

Preferences on single attributes are called base preferences. There are base preference constructors for continuous and for discrete domains. The discrete overhead associated with database updates, e.g. Kossmann et al. (2002), or Papadias et al. (2005). Nested-loop algorithms are the generic way of computing a skyline, e.g. Godfrey et al. (2005). Despite lower performance compared to index algorithms, they are capable of processing arbitrary data without any preparations necessary. Recently, algorithms have been developed exploiting the structure of the lattice imposed by the skyline operator on the data space to identify the skyline, but they keep this lattice structure in main memory and are only applicable on low-cardinality domains, see the works of Preisinger and Kießling (2007), and Morse et al. (2007).

Although, the evaluation of skyline queries is a well researched area, constrained skyline queries have not been intensively researched in the last years. There exists some algorithms for constrained skyline computation, e.g. BBS by Papadias et al. (2005), STA introduced by Delli et al. (2006), or a modification of SaLSa by Sun et al. (2008). But all of them check the hard constraints ‘inside’ their algorithms, i.e., they are modified join operators. All these algorithms have the disadvantage that in the case of a cartesian product, or a join, still all tuple combinations have to be computed before the skyline join can be applied. Also, Raghavan and Rundensteiner (2010) consider such queries. They present a progressive algorithm by translating a tuple-level processing into a job-sequencing problem. Jin et al. (2007 and 2010) integrate different join methods into skyline computation. Jin et al. (2007) covers the skyline operator on joins and aggregates. In Jin et al. (2010) only equijoins are considered. Nevertheless, no tuple reduction can be done before the join is evaluated. Cui et al. (2008) propose the PaDSkyline algorithm in distributed environments. However, PaDSkyline does not address algebraic optimization issues, which are the main contribution of Semi-Skylines.
POS-preference $\text{POS}(\text{A}, \text{POS-set})$ for example states that the user has a set of preferred values, the POS-set, in the domain of A. The sample query in Figure 1 shows two POS preferences with POS-sets for Soup and Meat. For example, Chicken and Noodle soups are preferred to all other soups.

Continuous numerical domains need a different type of preferences. For this purpose Kießling (2005) defined many numerical preference constructors, e.g., HIGHEST, LOWEST, ANTICHAIN. The extremal preferences HIGHEST and LOWEST allow users to express easily their desire for values as high or as low as possible. ANTICHAIN considers each tuple as equally good. Formally:

- $\text{P:=HIGHEST}(\text{A})$: A tuple $x$ is worse than a tuple $y$ if the value of $x$ is lower than $y$:
  \[ x <_{P} y \iff x < y \]

- $\text{P:=LOWEST}(\text{A})$: A tuple $x$ is worse than a tuple $y$ if the value of $x$ is higher than $y$:
  \[ x <_{P} y \iff x > y \]

- $\text{P:=ANTICHAIN}(\text{A})$: All values are considered as equally good, i.e., $<_P = \emptyset$.

The query in Figure 1 shows a HIGHEST preference for the amount of $\text{Ve}$ in the Beverage products.

There is the need to combine several base preferences into more complex preferences. One way is to list a number of preferences that are all equally important to the user. This is the concept of Pareto preferences.

**Definition 1. Pareto Preference (Kießling 2002)**

For preferences $P_1 = (A_1, <_{P_1})$ and $P_2 = (A_2, <_{P_2})$, a Pareto preference $P := P_1 \otimes P_2 = (A_1 \times A_2, <_P)$ is defined as:

\[ (x_1, x_2) <_P (y_1, y_2) \iff \begin{cases} \left( x_1 <_{P_1} y_1 \land (x_2 <_{P_2} y_2 \lor x_2 = y_2) \right) \lor \\ \left( x_2 <_{P_2} y_2 \land (x_1 <_{P_1} y_1 \lor x_1 = y_1) \right) \end{cases} \]

A generalization to more than two preferences is straightforward.

If we restrict the attention to LOWEST (MIN), HIGHEST (MAX) and ANTICHAIN (DIFF) as input preferences for a Pareto preference $P$, then Pareto preference queries coincide with the traditional skyline queries.

An important subclass of preferences are weak order preferences (WOP, Kießling 2005)), i.e., strict partial orders for which negative transitivity holds: \( \neg(x < y) \land \neg(y < z) \Rightarrow \neg(x < z) \) for a preference $P$. For a WOP $P = (A, <_P)$ the dominance test can be efficiently done by a numerical utility function $f_P$ which depends on the type of preference. Dominated tuples have higher function values.

\[ f : \text{dom}(A) \to \mathbb{R}^+_0 \quad x <_{P} y \iff f_P(x) > f_P(y) \]

For WOPs two domain values $x$ and $y$ having the same function value are considered as substitutable and are treated as one equivalence class (regular SV-semantics, cp. (Kießling 2005)).

The utility function has to be defined individually for every type of base preference. For numerical base preferences the utility function is interpreted as the numerical distance from a perfect value, e.g., $f_{\text{LOWEST}}(x) := x - \min$, where $\min$ is the minimum value of the domain of $A$ (Kießling 2005).

### 3.2 Semi-Pareto Preference

The Semi-Pareto preference is the key to our relational algebraic transformation laws for constrained skyline queries. Firstly introduced as Cutoff preference by Endres & Kießling (2008), our Semi-Pareto preference provides an optimization technique for preference queries in combination with the selection operator. Since this workshop paper was a first approach, and only covers the fundamentals of constrained skyline optimization, we now complete the theoretical background for our optimization techniques which we introduce in Section 4.

Comparing the definition of Semi-Pareto to Pareto, it is evident that Semi-Pareto is “the half of a Pareto preference”.

**Definition 2. Semi-Pareto Preference**

Let $P_1 = (A_1, <_{P_1})$, $P_2 = (A_2, <_{P_2})$ be preferences.

a) **Left-Semi-Pareto:** $P_1 \otimes_P P_2 = (A_1 \times A_2, <_{P_1 \otimes P_2})$

\[(x_1, x_2) <_{P_1 \otimes_P P_2} (y_1, y_2) \iff \begin{cases} x_1 <_{P_1} y_1 \land (x_2 <_{P_2} y_2 \lor x_2 = y_2) \lor \\ x_2 <_{P_2} y_2 \land (x_1 <_{P_1} y_1 \lor x_1 = y_1) \end{cases} \]

b) **Right-Semi-Pareto:** $P_1 \otimes_P P_2 = (A_1 \times A_2, <_{P_2 \otimes P_1})$

\[(x_1, x_2) <_{P_2 \otimes_P P_1} (y_1, y_2) \iff x_2 <_{P_2} y_2 \land (x_1 <_{P_1} y_1 \lor x_1 = y_1) \]

The proof that Semi-Pareto is a preference, i.e., it is irreversible and transitive, can be done straightforward, cp. the extended version of this paper (Endres & Kießling 2010). We now define Semi-Skylines.

**Definition 3. Semi-Skyline**

The Semi-Skyline of a Semi-Pareto preference $P := P_1 \otimes_P P_2$ or $P := P_1 \otimes_P P_2$ on an input relation $R$ are all tuples that are not dominated w.r.t. $P$.

**Example 1.** Consider two preferences on Table 1:

- $P_1 = \text{POS}(\text{B.Name}, ‘\text{Red Wine}’)$
- $P_2 = \text{HIGHEST}(\text{Ve})$

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Cal</th>
<th>Ve</th>
<th>Rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td>Red Wine</td>
<td>85</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>B2</td>
<td>Red Wine</td>
<td>181</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>B3</td>
<td>Coke</td>
<td>220</td>
<td>21</td>
<td>2</td>
</tr>
<tr>
<td>B4</td>
<td>Lemonade</td>
<td>281</td>
<td>17</td>
<td>2</td>
</tr>
<tr>
<td>B5</td>
<td>Red Wine</td>
<td>400</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 1: A sample data set for Beverages B.

Then we have the following semi-skylines:

- $\sigma[P_1 \otimes_P P_2](B) = \{B1, B2, B3, B4, B5\}$

No tuple is dominated due to $P_1 \otimes_P P_2$. All Red Wines are indifferent concerning $P_1$, and therefore cannot be worse than another tuple. $B3$ and $B4$ are worse than the others concerning $P_1$, but have a higher value for Ve, and therefore are not dominated.

- $\sigma[P_1 \otimes_P P_2](B) = \{B2, B3\}$

$B2$ dominates all other Red Wines because it has the highest amount of Ve, and concerning $P_1$ they are substitutable. $B4$ is dominated by $B3$ (substitutable in the kind of drink, but worse in the amount of Ve), but $B2$ does not dominate $B3$ (Red Wine is better than Coke, but Coke has a higher value for Ve).
We conclude the following lemma:

**Lemma 1.** Semi-Pareto is not a WOP.

To show that Semi-Pareto is not a WOP we give a counterexample.

**Example 2.** Consider the sample data set from Table 1 holding a relation on beverages.

Given two preferences $P_1 = \text{LOW \, EST (Cal)}$ and $P_2 = \text{LOW \, EST (Fat)}$ and $P = P_1 \otimes P_2$.

Then 'B1' is indifferent to 'B5', since none of them is better than the other concerning $P$. Furthermore, 'B5' is indifferent to 'B3'. It has more calories, but is better in the amount of fat. But 'B1' is better than 'B3', because 'B1' has less calories and less fat than 'B3'. Hence, negative transitivity does not hold.

Note that Semi-Pareto in contrast to Pareto is neither commutative nor associative. However, the following laws hold:

**Corollary 1.** Algebraic Laws for Semi-Pareto

a) $P_1 \otimes P_2 = P_2 \otimes P_1$

b) $(P_1 \otimes P_2) \otimes P_3 = P_1 \otimes (P_2 \otimes P_3)$

**Proof.** The proof is straightforward and can be done by applying Boolean algebraic transformations.

## 4 Constrained Skyline Optimization

In this section we study constrained Skyline queries and their algebraic optimization. However, we present a more general approach for arbitrary preferences.

### 4.1 Formal Background

Given a preference $P = (A, <_p)$ on a database schema $R$ with $A \subseteq \text{attr}(R)$, and a Boolean condition $H$, i.e., a hard constraint. Then, using preference selection from Section 3, we define a constrained preference (skyline) query as $\sigma [P](\sigma [H](R))$. Under some conditions, preference selection $\sigma [P]$ and hard selection $\sigma [H]$ are commutative, cp. Chomiczek (2003) and Hafnerrichter & Kießling (2005).

**Theorem 1.** Push Preference over Hard Selection

For a preference $P = (A, <_p)$ with $A \subseteq \text{attr}(R)$ and a hard constraint $H$ the following holds:

$$\sigma [P](\sigma [H](R)) = \sigma [H](\sigma [P](R)) \iff \forall w \in R : H(w) \land \exists v \in R : w <_p v \rightarrow H(v)$$

If a tuple $w$ is dominated by a tuple $v$, and $w$ fulfills the hard constraint, then $v$ has to fulfill the hard constraint, too. This guarantees the reduction of a tuple $w$ only if for each dominating tuple $v$ the condition $H(v)$ is fulfilled. Hence, a commutation due to Theorem 1 is possible.

**Example 3.** Consider $P := \text{HIGHEST}(Vc)$ and the sample data set of Beverages in Table 1.

We put a hard constraint $H := Vc \geq 15$ on the preference, i.e., $\sigma [P](\sigma [Vc \geq 15](B))$. A commutation of the preference and the hard selection is possible, because tuples which fulfill the hard constraint also fulfill the preference.

Thus, $\sigma [P](\sigma [Vc \geq 15](B)) = \sigma [Vc \geq 15](\sigma [P](B))$, both leading to the result 'B3'.

### 4.2 Algebraic Optimizations

In this section we look at a database relation $R = (A_1, \ldots, A_t, B_1, \ldots, B_k)$ where the $B_i$'s are numerical attributes. We consider a hard constraint $H$ on $R$ as follows:

$$H := h(b_1, \ldots, b_k) \Theta c, \quad \Theta \in \{\leq, <, >, \geq, =, \neq\}$$

where $h : \text{dom}(B_1) \times \ldots \times \text{dom}(B_k) \rightarrow \mathbb{R}$ is a monotone function, $b_i \in \text{dom}(B_i)$, and $c \in \mathbb{R}$ a constant. An instance of such a query is our running example, see Figure 1.

Applying Theorem 1 in the presence of hard constraints like $H$ is a non-trivial task. In the following we introduce a novel algebraic optimization idea that achieves to satisfy the complex conditions required for Theorem 1 to hold. The key idea is to syntactically derive an induced preference from the given query, which then can be used as a prefilter preference.

**Definition 4.** Prefilter Preference

A preference $Q = (A, <_Q)$ is a prefilter preference for a preference $P = (A, <_P)$ w.r.t. a relation $R$ iff

$$\sigma [P](R) = \sigma [P](\sigma [Q](R))$$

**Definition 5.** Induced Preference

Given a database relation $R$, and a hard constraint $H$ as above. Then we define a preference $\overline{H}_{B_i}$ induced by $B_i$, as:

$$\overline{H}_{B_i} := \{ \overline{\text{LOW \, EST}}(B_i) \quad \text{if} \quad \Theta \in \{\leq\} \}
\quad \overline{\text{HIGHEST}}(B_i) \quad \text{if} \quad \Theta \in \{>, \geq\}$$

For all induced preferences $\overline{H}_{B_i}$, $i = 1, \ldots, k$ we define the overall induced preference $\overline{H}$ as

$$\overline{H} := \overline{H}_{B_1} \otimes \ldots \otimes \overline{H}_{B_k}$$

**Example 4.** Consider the following query

$$\sigma [S, \text{Cal} + M, \text{Cal} + B, \text{Cal}] \leq 1100(S \times M \times B)$$

where $H$ is the sum of the calories. Since $\Theta$ is "\leq" our induced preferences are

- $\overline{H}_{S, \text{Cal}} := \overline{\text{LOW \, EST}}(S, \text{Cal})$
- $\overline{H}_{M, \text{Cal}} := \overline{\text{LOW \, EST}}(M, \text{Cal})$
- $\overline{H}_{B, \text{Cal}} := \overline{\text{LOW \, EST}}(B, \text{Cal})$

The overall induced preference is

$$\overline{H} = \overline{H}_{S, \text{Cal}} \otimes \overline{H}_{M, \text{Cal}} \otimes \overline{H}_{B, \text{Cal}}$$

Using these definitions, our Semi-Pareto preference can be pushed over the hard selection operator.

**Theorem 2.** Push Semi-Pareto over Hard Selection

Given an arbitrary preference $P$ and a hard constraint $H$ on a relation $R$. Then

$$\sigma [P \otimes \overline{H}](\sigma [H](R)) = \overline{H}(\sigma [P \otimes \overline{H}](R))$$

where $\overline{H}$ is an induced preference from Definition 5.

**Proof.** Consider a relation $R$ and tuples $t$ and $t'$. Assume $t <_p \overline{H} t' \land H(t) \iff t <_p \overline{H} t' \land (t <_p \overline{H} t' \lor t =_p \overline{H} t') \land H(t)$

\footnote{If two tuples have equal values concerning the attributes of a preference $P$ we write $t =_p t'$.}
Since \( t \) fulfills the hard constraint \( H \) and is worse or as good as \( t' \) concerning the induced preference, it follows that \( t' \) fulfills the hard constraint, too, i.e., \( H(t') \) is valid. Therefore we can apply Theorem 1 and push the preference over the hard constraint.

If \( P \) is an induced preference, i.e., LOWEST, HIGHEST or ANTICHAIN we can simplify Theorem 2 as follows:

**Corollary 2.** Consider a preference query where \( \overline{H} \) is an induced preference, then:

\[
\sigma[\overline{H}](\sigma_H(R)) = \sigma_H(\sigma[\overline{H}](\sigma_H(R)))
\]

Semi-Pareto can be used as a prefilter preference to eliminate tuples from the underlying relation which are definitely no candidates for the skyline. Particularly, this is a crucial step in queries involving joins as we will see later.

**Theorem 3.** Prefilter Preference and Hard Selection

Consider a preference query \( \sigma[P](\sigma_H(R)) \) with a hard constraint \( H \) on a relation \( R \). Then \( P \quad \overline{\sigma} \quad \overline{H} \) is a prefilter preference for \( P \), i.e.,

\[
\sigma[P](\sigma_H(R)) = \sigma[P](\sigma[P \quad \overline{\sigma} \quad \overline{H}](\sigma_H(R)))
\]

**Proof.** We prove “\( \subseteq \)” and “\( \supseteq \)”.

“\( \subseteq \)”:
Let \( t \in \sigma[P](\sigma[H](R)) \), i.e., \( \exists t' \in \sigma[H](R) : t <_P t' \).
Assume, \( t \notin \sigma[P](\sigma[P \quad \overline{\sigma} \quad \overline{H}](\sigma_H(R))) \). Then either

\[
\exists t' \in \sigma[P \quad \overline{\sigma} \quad \overline{H}](\sigma_H(R)) : t <_P t' \\
\text{or } \exists t' \in \sigma[H](R) : t <_P t'
\]

A contradiction to \( t \in \sigma[P](\sigma[H](R)) \).

“\( \supseteq \)”:
Let \( t \in \sigma[P](\sigma[P \quad \overline{\sigma} \quad \overline{H}](\sigma_H(R))) \).
Assume, \( t \notin \sigma[P](\sigma[H](R)) \), i.e.,

\[
\exists t' \in \sigma[H](R) : t <_P t'
\]

But then, \( t' \) must be dominated w.r.t. \( P \quad \overline{\sigma} \quad \overline{H} \) since \( t \in \sigma[P](\sigma[H](R)) \), i.e.,

\[
\exists t'' \in \sigma[H](R) : t' <_P \overline{\sigma} \overline{H} t'' \text{ i.e., } \exists t'' \in \sigma[H](R) : t' <_P t'' \land (t'' <_H t' \lor t'' =_H t'')
\]

In sum, we have \( t' <_P t'' \) and \( t <_P t' \). By transitivity it follows \( t <_P t'' \), hence a contradiction to \( t \notin \sigma[P](\sigma[H](R)) \).

Now we can push Semi-Pareto over the hard constraint.

**Corollary 3.** Insert and Push Semi-Pareto over \( H \)

\[
\sigma[P](\sigma_H(R)) = \sigma[P](\sigma[H](\sigma[P \quad \overline{\sigma} \quad \overline{H}](\sigma_H(R))))
\]

**Proof.** By Theorem 2 and Theorem 3.

Of course, our results are not restricted to a single input relation. They are also valid for cartesian products having a hard constraint concerning attributes of all participating relations.

**Corollary 4.** Push Semi-Pareto over Cart. Prod.

Let \( P_1 = (A_1, <_{P_1}) \), \( P_2 = (A_2, <_{P_2}) \) be two preferences, and \( H := h(b_1, b_2) \) a hard constraint, \( A_1, B_1 \subseteq \text{attr}(R) \) and \( A_2, B_2 \subseteq \text{attr}(S) \), \( b_1 \in \text{dom}(B_1) \), and \( b_2 \in \text{dom}(B_2) \). Then:

1. \( \sigma[P_1](\sigma[H](R \times S)) = \sigma[P_1](\sigma[H](\sigma[P_1 \quad \overline{\sigma} \quad \overline{H}](R \times S))) \)
2. \( \sigma[P_1 \land P_2](\sigma[H](\sigma[P_1 \quad \overline{\sigma} \quad \overline{H} B_1](R) \times \sigma[P_2 \quad \overline{\sigma} \quad \overline{H} B_2](S))) \)

**Proof.** This is a consequence of Corollary 3 and law L2 from Hafnerrichter & Kießling (2005), “Push Preference Over Cartesian Product”.

This theorem allows us to push the Semi-Pareto prefilter preference over a hard constraint as presented in the next example.

**Example 5.** Revisit the preference query from Figure 3 with preferences

- \( P_1 = \text{POS}(S.Name, \{\text{‘Chicken’}, \text{‘Noodle’}\}) \)
- \( P_2 = \text{POS}(M.Name,’\text{Beef’}) \)
- \( P_3 = \text{HIGHEST}(B.Vc) \)

combined to a Pareto preference \( P = P_1 \land P_2 \land P_3 \). Furthermore, consider the single hard constraint on the sum of calories (Cal) that must be less or equal to 1100 kcal, \( H = S.Cal + M.Cal + B.Cal \leq 1100 \). The induced preferences are the same as in Example 4. Using Corollary 4 we can insert the prefilter preferences and push them over the hard constraint down to the relations, cp. Figure 4.

![Figure 3: Push Semi-Pareto over Cart. Prod.](image-url)

The insertion of our Semi-Pareto prefilter preferences lead to an elimination of tuples from the relations before building the cartesian products and hence reduces memory and computation costs.

The case of a hard constraint in combination with Theta-joins (or Equi-joins) like \( R \quad \text{\overline{\quad \overline{\sigma} \quad \overline{H}}} \quad R.X=S.X \) for \( X \subseteq \text{attr}(R) \cap \text{attr}(S) \) we have to ensure that we do not eliminate join partners, i.e., for each tuple in the first relation there must exist a join partner in the second relation. To get rid of this problem we have to evaluate the Semi-Pareto preference as a grouped preference selection. Therefore, Semi-Pareto is only evaluated for tuples in the same equivalence class, i.e., grouped by \( X \).

**Corollary 5.** Push Semi-Pareto over Join

Let \( P = (A, <_{P}) \) be a preference and \( H := h(b_1, b_2) \) a hard constraint on two relations \( R \) and \( S \). Furthermore, \( A_1, B_1 \subseteq \text{attr}(R) \) and \( B_2 \subseteq \text{attr}(S) \), as well as \( X \subseteq \text{attr}(R) \cap \text{attr}(S) \), \( b_1 \in \text{dom}(B_1) \), \( b_2 \in \text{dom}(B_2) \), and \( \Theta = \{<,\leq,\geq,\neq\} \). Then

\[
\sigma[P](\sigma[H](\sigma[R \quad \text{\overline{\quad \overline{\sigma} \quad \overline{H}}} \quad \text{\overline{\overline{\sigma} \overline{\overline{H}}} \quad R.X=S.X \quad (S))}))
\]
Proof. This is a consequence from Corollary 3 and Hafenrichter and Kießling (2005), law L6, which discusses “Push Preference Over Join”.

Until now, we have only considered one hard constraint \( H \) in combination with preference constraints. However, in database queries multiple hard constraints composed by the SQL keywords AND (conjunction) or OR (disjunction) are often involved in the selection conditions. The next theorem will introduce a prefilter preference for such queries. For disjunctive constraints like \( H := H_1 \Theta_1 c_1 \Theta_2 H_2 \Theta_2 c_2 \), \( \Theta_i \in \{ \leq, \leq, <, >, =, \neq \} \), the condition \( \sigma_H \) could be regarded as the union of two operators \( \sigma_{H_1\Theta_1c_1}(R) \) and \( \sigma_{H_2\Theta_2c_2}(R) \). Therefore, we only consider conjunctive constraints.

**Corollary 6.** Consider a preference query \( \sigma[P](\sigma_H(R)) \) with \( k \) conjunctive hard constraints \( H := H_1 \wedge \ldots \wedge H_k \) on a relation \( R \). Then:

\[
\sigma[P](\sigma_{H_1\wedge\ldots\wedge H_k}(R)) = \sigma[P](\sigma_{H_1}(\ldots(\sigma_{H_k}(\sigma_H(R)))\ldots))
\]

Proof. Relational algebra leads to

\[
\sigma[P](\sigma_{H_1\wedge\ldots\wedge H_k}(R)) = \sigma[P](\sigma_{H_1}(\sigma[P \bowtie \tilde{H}_1](\sigma_{H_2}(\ldots(\sigma_{H_k}(\sigma_H(R)))\ldots)))
\]

for the single hard constraint \( H_1 \). Now by further applying Corollary 3 on \( \sigma[P \bowtie \tilde{H}_1](\sigma_{H_2}(\ldots(\sigma_{H_k}(\sigma_{H(R)})))) \) we get

\[
\sigma[P](\sigma_{H_1\wedge\ldots\wedge H_k}(R)) = \sigma[P](\sigma_{H_1}(\sigma_{H_2}(\ldots(\sigma_{H_k}(\sigma_{H(R)})))) = \sigma[P \bowtie \tilde{H}_1 \bowtie \tilde{H}_2 \bowtie \ldots \bowtie \tilde{H}_k](\sigma_{H_1}(\ldots(\sigma_{H_k}(\sigma_H(R)))\ldots))) = \sigma[P \bowtie \tilde{H}_1 \bowtie \tilde{H}_2 \bowtie \ldots \bowtie \tilde{H}_k] \text{ by Corollary 1}
\]

Note that the intermediate preferences \( \sigma[P \bowtie \tilde{H}_i] \) can be removed because they are only prefilter preferences and need not to be evaluated.

**Example 6.** Consider the skyline query from Figure 1, and Example 5 with our preference \( P := P_1 \bowtie P_2 \bowtie P_3 \). Now, for all three hard sum constraints on ‘Cal’, ‘VC’ and ‘Fat’ we have \( H := H_1 \wedge H_2 \wedge H_3 \) for

- \( H_1 := S.Cal + M.Cal + B.Cal \leq 1100 \)
- \( H_2 := S.VC + M.VC + B.VC \geq 38 \)
- \( H_3 := S.Fat + M.Fat + B.Fat \leq 9 \)

The overall induced preferences are

\[
\tilde{H}_S := \text{LOWEST}(S.Cal) \bowtie \text{HIGHEST}(S.VC) \bowtie \text{LOWEST}(S.Fat)
\]

\[
\tilde{H}_M := \text{LOWEST}(M.Cal) \bowtie \text{HIGHEST}(M.VC) \bowtie \text{LOWEST}(M.Fat)
\]

\[
\tilde{H}_B := \text{LOWEST}(B.Cal) \bowtie \text{HIGHEST}(B.VC) \bowtie \text{LOWEST}(B.Fat)
\]

Applying Corollary 6 leads to:

\[
\sigma[P](\sigma_H(S \times M \times B)) = \sigma[P](\sigma(H)
\]

\[
\sigma(P_1 \bowtie P_2 \bowtie P_3 \bowtie \tilde{H}_S \bowtie \tilde{H}_M \bowtie \tilde{H}_B)(S \times M \times B)
\]

This is equal to

\[
\sigma[P](\sigma_H(S \times M \times B)) = \sigma[P](\sigma(H)(P_1 \bowtie \tilde{H}_S))(S \times \sigma(P_2 \bowtie \tilde{H}_M)(M) \times \sigma(P_3 \bowtie \tilde{H}_B)(B)
\]

5 Efficient Evaluation of Semi-Skylines

Many algorithms have been developed for skyline evaluation, cp. Section 2. Since the concept of semi-skylines is totally new, no specialized algorithm for its computation exists. Therefore, only BNL with a worst-case complexity of \( O(n^2) \) remains to evaluate semi-skylines. Thus, the question for an efficient algorithm arises. We introduce the Staircase algorithm for the evaluation of semi-skylines with guaranteed worst-case complexity of \( O(n \log n) \). Staircase is a variant of the BNL algorithm, but the candidate window will be a Skiplist (Pugh 1990).

5.1 The Staircase Algorithm

For simplicity we restrict the attention to weak order preferences \( P_1 \) and \( P_2 \) as input for Semi-Pareto. Then we can define Left-Semi-Pareto from Definition 2 (analogously Right-Semi-Pareto) by utility functions:

\[
(x_1, x_2) \prec_{P_1 \bowtie P_2} (y_1, y_2) \iff f_{P_1}(x_1) > f_{P_1}(y_1) \wedge f_{P_2}(x_2) \geq f_{P_2}(y_2)
\]

A tuple \( x \) := \( (x_1, x_2) \) is worse than a tuple \( y \) := \( (y_1, y_2) \), iff the utility function value is worse in the first component, i.e., \( f_{P_1}(x_1) > f_{P_1}(y_1) \) and worse or equal in the second one, i.e., \( f_{P_2}(x_2) \geq f_{P_2}(y_2) \). Since we map tuples \( (x_1, x_2) \) with the same function values to equivalence classes represented by \( (f_{P_1}(x_1), f_{P_2}(x_2)) \), we can state directly dominance using these equivalence classes. For a graphical interpretation have a look at Figure 4a. All equivalence classes in the pruning region \( PR \), i.e., below and right of the equivalence class \( [y] = (2, 2) \) are worse than \( y \), because their \( P_1 \) value is greater to 2 and worse or equal or 2 in the second preference \( P_2 \). Note that the equivalence classes on the dashed line are not dominated. Therefore, a tuple belonging to the equivalence class \( (2, 2) \) dominates all tuples belonging to an equivalence class lying in \( PR \). This dominance test is only possible if the underlying preferences are WOPs. Comparing a new tuple \( x := (x_1, x_2) \) with equivalence class \( [x] = (f_{P_1}(x_1), f_{P_2}(x_2)) \) leads to the following possibilities:

a) If \( [x] \) falls into the pruning region \( PR \) (Figure 4a) we directly can state dominance using the equivalence classes. For example consider \( [x] = (3, 3) \). Since \( 3 > 2 \) and \( 3 \geq 2 \) the equivalence class \( [x] \) is worse then \( [y] \), thus the tuple \( x \) is dominated. If an equivalence class falls directly on the dashed line it is not dominated, since tuples in such a class are not worse concerning the first preference.

b) If an equivalence class is left below of \( [y] \) it is not dominated, but extends our staircase, cp. Figure 4b. Equally if an equivalence class is right above \( [y] \) (or on the dashed line) our staircase will be extended. For example, \( [y'] = (1, 3) \) and \( [y''] = (3, 0) \) extend our staircase and therefore the pruning region \( PR \). Inserting these equivalence classes all tuples lying in an equivalence class of the gray area are dominated, i.e., dominance can now be decided by the dichotomy of the staircase.

c) Only an equivalence class \( [z] \) left above \( [y] \) is better than \( [y] \) and therefore dominates it, cp. Figure 4c. In this case, we have to update our staircase, since it is possible that \( [z] \) dominates other equivalence classes (and their containing tuples), too. But updating is an easy step, because \( [z] \) only can dominate equivalence classes right below of itself. All equivalence classes right below of \( [z] \) (without the dashed line) in the order of our staircase
This leads to the following possibilities:

- The points on the staircase build a total order concerning the Manhattan distance and change the staircase to its new form consisting of \([z] = (0.5, 1), [y'] = (3, 0)\). In the worst-case of BNL all tuples in the candidate window have to be compared to the new tuple to decide dominance. In contradiction, using the staircase we only have to decide if a tuple is left of the staircase, i.e., it dominates, or a tuple is right of the staircase, i.e., it is dominated. This leads to the question how to represent the staircase data structure?

### 5.2 The Staircase Data Structure

We observe that the dominance decision based on the staircase is only applicable if the equivalence classes on the staircase are comparable and ordered, yielding a total order using the Manhattan distance, cf. (Krause 1987).

**Definition 6.** Manhattan distance, \(L_1\) norm

The Manhattan distance \(d_1\) for an equivalence class \([x] = (x_1, x_2)\) in our staircase space is the distance from the point \((0, \max_P)\) to the point \([z] = (x_1, x_2)\), where \(\max_P := \max(\max(f_{P_1}), \max(f_{P_2}))\). \(\max(f_P)\) is the maximum function value for \(P\).

\[
d_1([x]) = x_1 - x_2 + \max_P
\]

**Lemma 2.** Total Order of Staircase Points

The points on the staircase build a total order concerning the Manhattan distance from Definition 6.

**Proof.** We give the proof using a Left-Semi-Pareto preference \(P := P_1 \cup P_2\) with weak order preferences \(P_1 = (A_1, <_{P_1})\) and \(P_1 = (A_2, <_{P_2})\). The proof for Right-Semi-Pareto can be done analogously.

Consider an equivalence class \([y] = (y_1, y_2)\) with Manhattan distance \(d_1([y]) = y_1 - y_2 + \max_P\) already on the staircase. We want to insert a new tuple \(x\) with equivalence class \([x] = (x_1, x_2)\) and the same distance \(d_1([x])\). Since \(\max_P\) is fixed, they only have the same Manhattan distance if \(y_1 - y_2 = x_1 - x_2\).

This leads to the following possibilities:

- if \(x_i = y_i, i \in \{1, 2\}\), both fall in the same equivalence class, i.e., tuple \(x\) will be added to \([y]\).
- if \(x_1 > y_1\) and \(x_2 > y_2\), then \([x]\) is dominated by \([y]\). Therefore tuple \(x\) is dominated, too.
- if \(x_1 < y_1\) and \(x_2 < y_2\), then \([y]\) is dominated and replaced by \([x]\).

Since these are all possibilities for \(d_1([y]) = d_1([x])\) the staircase points build a total order concerning the Manhattan distance.

As an example consider the equivalence classes \([z] = (0.5, 1)\) and \([y'] = (3, 0)\) from Figure 4c. We get a distance \(d_1([z]) = -0.5 + \max_P\) and \(d_1([y']) = 3 + \max_P\).

Using the Manhattan distance the dominance decision in the staircase is easy. Search \(d_1([x])\) of a tuple \(x\). If \(d_1([x])\) exists, just compare the utility function values with the existing equivalence class and find out dominance (or add the tuple to the equivalence class if it has the same function values). If \(d_1([x])\) does not exists, compare the equivalence class of \([x]\) to the one with next lower distance. If \([x]\) is not dominated insert it into the staircase and update the staircase, i.e., delete all equivalence classes dominated by \([x]\).

Our first idea to use balanced search trees to store the staircase failed on finding the dominated equivalence classes for an update action, i.e., in a binary tree it is not easy to find all points which are worse than the newly inserted point. Skiplists are a collection of sorted linked lists, each at a given “level”, that mimic the behavior of a binary search tree.

**Example 7.** Figure 5 shows a Skiplist with equivalence classes as keys ordered by the Manhattan distance (number below the classes, \(\max_P = 10\)). Among other points it contains the classes from Figure 4b.

![Figure 4a: Comparing a dominated equivalence class.](image)

![Figure 4b: Extending the Staircase.](image)

![Figure 4c: Updating the Staircase.](image)

![Figure 5: A Skiplist with maximum level 4.](image)

The number of nodes in each list decreases with the level, implying that we can find a key quickly by searching first at higher levels, skipping over large numbers of shorter nodes, and progressively working downwards until a node with the desired key is found, or the bottom level is reached. Thus, the time complexity of a Skiplist operation (insert, delete, search) is logarithmic in the length of the list, \(O(\log n)\), cp. (Pugh 1990). Skiplists also provide easy access to all equivalence classes for the update action. Begin at the inserted point and run through the list until dominance fails.

**Theorem 4.** Staircase Time Complexity

Let \(P\) be a Semi-Pareto preference. Given \(n\) input tuples, then the semi-skyline of \(P\) can be evaluated in

- worst-case runtime: \(O(n \log n)\)
- best-case runtime: \(O(n)\)
Proof. For each input tuple we either have to insert or remove it from the staircase. These operations can be done in logarithmic time. Therefore, we get a worst-case complexity of \( n \cdot \mathcal{O}(\log n) \).

Since our Staircase algorithm is a specialized BNL algorithm, we can guarantee a best-case runtime of \( \mathcal{O}(n) \), cp. for instance (Godfrey et al. 2005).

Note that for simplicity we have only presented the Staircase algorithm for WOPs. Staircase can be generalized to arbitrary preferences, but due to length restrictions this can not be presented here.

6 Performance Benchmarks

We now present results from an experimental study designed to show the benefit of our techniques.

All compared algorithms are implemented in our PreferenceSQL system (Hafenrichter & Kießling 2005), a Java SE 6 framework for preference queries. All experiments are performed on a 2.53GHz Core 2 Duo machine running Mac OS X with 4 GB RAM for the JVM. Moreover a buffer pool large enough for all operations to fit into the main memory for all tests was used. The input sets and the skyline points are kept in main memory, too. Performing all operations in main memory is the best case for all used algorithms since no external operation is necessary. It has to be noted that this main memory requirement restricts the general applicability of our Staircase algorithm. However, as can be seen from the subsequent benchmarks, the Staircase algorithm can nevertheless be used for very large data sets.

6.1 BNL vs. Staircase

From Section 5 we know that BNL is the only algorithm to evaluate arbitrary preference queries. Therefore we compare BNL and our Staircase algorithm for semi-skyline evaluation. Both algorithms have been integrated in our PreferenceSQL framework to produce performance benchmarks.

We use synthetic data sets, since this is commonly used for skyline evaluation and allow us to carefully explore the effect of various data characteristics. For this, we generate data sets with correlated (COR), independent (IND) and anti-correlated (ANTI) distributions using an implementation of the popular data set generator of Bőrzsönyi et al. (2001), and vary the data cardinality \( n \), and the number of distinct domain values \( c \). In all synthetic experiments, the tuple size is 100 bytes (also by Godfrey et al. (2005) in their experiments). A tuple has attributes of type Integer and bulk attributes with “garbage” characters to ensure that each tuple is 100 byte long. For simplicity we generated a Semi-Pareto preference consisting of LOWEST preferences (MIN in skylines queries).

Figure 6 shows runtimes for BNL and Staircase on an anti-correlated data set containing up to \( n = 500K \) tuples. We fixed \( c = 100K \), i.e., the domain contains 100,000 different values. Figure 7 contains the comparison of BNL and Staircase for an independent distributed data set. Due to the limited space we only present these benchmarks. Further tests with different cardinalities and number of distinct domain values can be found in the extended version of this paper, cp. (Endres & Kießling 2010). In all benchmarks it turns out that Staircase for semi-skyline evaluation performs much better than BNL.

6.2 Constrained Skyline Optimization

To evaluate the Pareto prefilter preference, we performed several experiments. For this we integrated our optimization rules into the preference query optimizer of PreferenceSQL. This allows the evaluation of constrained preference queries. For the evaluation of the semi-skylines we used our Staircase algorithm from Section 5. The evaluation of the remaining preference was done by Hexagon (Preisinger & Kießling 2007). We evaluated the efficiency of our Semi-Pareto prefilter preference (abbr. pref-prefilter) by comparing the response times of several constrained preference queries to an evaluation of the queries with standard optimization (e.g., building joins, cp. Hafenrichter & Kießling (2005)), without prefilter preferences. We abbreviate this with no-pref-prefilter.

Remark: The reason for choosing the comparison between an optimization with prefilter preferences and without prefilter preferences is as follows: We developed generic optimization rules which can be easily integrated into a preference query optimizer. We do not rely on any index structures. Furthermore, the mentioned skyline join algorithms from Section 2 (Related Work) check the hard constraints 'inside' their algorithms, i.e., these works present modified join operators for constrained skyline evaluation. All these algorithms have the disadvantage that in the case of a cartesian product, or a join, still all tuple combinations have to be considered for the skyline join. Therefore, a comparison to these skyline join algorithms is not a good representation of the benefit of our optimization laws. However, these algorithms can be part of the operator repertoire to choose from.
For the evaluation we used the food database mentioned in the introduction. From this database we created several relations, e.g., *Soup*, *Meat*, and *Beverage* containing information about their eponymous types of food. The sizes of these relations are as follows: There are max. 500 soups, 680 meats, and 350 beverages available, i.e., there are about 120 Mio. possible combinations. We also evaluated the influence of various data distributions to our prefilter preference. In all tests it turns out, that our optimization laws yield significant benefits for constrained skyline computation. Due to limited space we only present tests on the USDA data set. For further benchmarks we refer to Endres and Kießling (2010).

**Test 1:** The first test is based on the query in Figure 1 and contains three constraints. For representation we varied the amount of calories (*Cal*), which must be less than or equal to a value called *max_cal*. The amount of vitamin C (*Vc*) and the fat value (*fat*) are fixed values. Notice, modifying the parameter *max_cal* changes the selectivity of the query, while varying the size of the relations changes the size of the problem to be solved. We varied the *max_cal* value in a range from 500 to 1600 calories, see Figure 8. Since the prefilter preference only depends on the preferences and not on the hard constraints, the response time for the preference query with different *max_cal* is nearly constant for each approach. In contrast, the approach without prefilter preference takes much more time to evaluate the whole query, since it must build the full cartesian product to evaluate the join conditions and the Pareto preference.

![Figure 8: Different amount of calories.](image1)

**Test 2:** In our second test we run the query from Figure 1 with different relation sizes, i.e., a different number of possible tuple combinations (but fixed constraints) on our USDA database and demonstrate the results in Figure 9. Again, the prefilter preference eliminates tuples before building the cartesian product and therefore speeds up the evaluation of the join and the overall Pareto preference, respectively.

![Figure 9: Different relations sizes.](image2)

**Test 3:** We regard a preference query without a cartesian product, i.e., a query containing only one relation, namely the *Meat* table, but with a hard constraint on it (*M.Cal ≤ 300*). Since the hard constraint is a strong filter for the data set, the assumption arises that inserting the prefilter preference and pushing it over the hard selection will not speed up the computation. This is due to the prefilter preference is not such a strong criteria as the hard selection. This result is verified by the benchmarks in Figure 10, where we run the query on the *Meat* table with a modified tuple count.

![Figure 10: One relation, different tuple count.](image3)

The runtimes without the prefilter preference are much better, because using the prefilter implies the evaluation of itself, the evaluation of the hard selection, and the computation of the original preference on the top of the query. In contrast, without prefilter preference only the hard selection and the preference on top of the query has to be evaluated.

### 7 Summary and Outlook

In this paper we have proposed the novel concept of semi-skylines. For semi-skylines we have provided the *Staircase* algorithm, employing Skiplists for on-the-fly dominance testing with a worst-case complexity of $O(n \log n)$. We do not rely on any pre-computed index structure. Hence, our methods are generally applicable. As an application of semi-skylines we presented algebraic preference query optimization techniques for constrained skyline queries. Our algebraic laws can be easily integrated into a database system by extending the query optimizer. After such an operator tree optimization a cost-based optimization phase can then choose the best available skyline evaluation algorithm. In particular, known skyline joins can be part of the operator repertoire to choose from. Thus our presented algebraic transformation approach using semi-skylines is not intended as a substitution for skyline joins, but may well complement each other. Our experimental results show the benefit of our algebraic optimizations. Although we reduced the query evaluation time from round about 10min to 1min (for 120-10⁶ tuples), further work must be done for a very fast retrieval of all skyline points, e.g. in an online diet planning service.
References


Schema-less XML in Columns

Zuzana Částková\(^1\) \hspace{1cm} Jaroslav Pokorný\(^2\)

\(^1\) e-Fractal s.r.o.
Vinořadská 174, Praha 3, Czech Republic
castkova.zuzka@gmail.com

\(^2\) Faculty of Mathematics and Physics
Charles University in Prague
Maloušké nám. 25, Praha 1, Czech Republic
pokorny@ksi.mff.cuni.cz

Abstract

C-store environment uses a relational database for storing table tuples on the disk by columns. Can it be effectively used as XML database? This paper considers XML data without a schema. A two-level model of C-store based on XML-enabled relational databases is proposed. A measure of the model suitability is the possibility of evaluating effectively XPath queries. The XPath fragment considered allows the node-test not referring to attribute values and text values. Child, descendant, parent, ancestor, siblings, and following (preceeding) are just the XPath axes used here. Low level memory system enabling the estimation of the number of two abstract operations providing an interface to an external memory is characteristic for algorithms for each axis. We will show that our algorithms are mostly of logarithmic complexity in \(n\), where \(n\) is the number of nodes of XML tree associated with a XML document.

Keywords: XML, XPath axes, relational database, column stores, C-store

1 Introduction

Column-oriented databases (column stores) are an attractive area both for practice and research in the past few years. The values for each single column (or attribute) are stored contiguously, compressed and densely packed in this architecture. The DBMS C-store introduced in (Stonebraker, et al 2007) is a particular approach to column stores.

Among original motivations for use of column stores belong read-intensive analytical processing workloads, such as those encountered in data warehouses and OLAP. As mentioned by Abadi, Madden, and Hachem (2008) even show that various attempts with row-store converge to performance that is significantly slower on a recently proposed data warehouse benchmark. A more advanced application of C-store is its variant SW-store for storing RDF data as describe authors of (Abadi, et al, 2009). Abadi in (Abadi 2007) explores other more general applications of C-store like so called tables with wide schemas and tables with sparse attributes, i.e. tables with many NULL values. Considering XML-enabled DBMSs, approaches that use rather universal table approach belong also to this category. For a detailed discussion of column-oriented database systems see, e.g., tutorial by Abadi, Bonecz, and Harizopoulos (2009). In Částková (2009) the author examines a use of column stores for storing and processing XML data.

Our goal is not only to store XML data in column store but also to evaluate effectively an appropriate set of XML queries. Choosing the XPath language means to consider evaluation of XPath axes, i.e. relationship types in which a current node is associated to other nodes in the XML tree. We consider child, descendant, parent, ancestor, siblings, and following (preceeding) axes in the paper. We will show that our algorithms are mostly of logarithmic complexity in \(n\), where \(n\) is the number of nodes of XML tree associated with a XML document.

This improves results published by Bojanczyk and Parys (2008) providing \(O(2^{\frac{n}{|\phi|}} n)\) complexity, where \(|\phi|\) is the size of query \(\phi\). Their queries deal with attribute values and only checking attribute values for equality.

As most known column store architecture we have chosen the C-store for our research.

In the paper we will consider XML data without schema. In Section 2 we recall some basics of C-store. Section 3 describes a low-level memory system forming an abstract level to real disk memory. In Section 4 we propose a two-level model of C-store based on XML-enabled relational databases. We use a simple structure-centred mapping for storing XML data into relational database and a combination of two well-known numbering schemes: foreign key and depth first. Section 5 is devoted to design and analysis of algorithms for processing XPath axes. Section 6 concludes the paper.

2 C-store: overview

According to work (Stonebraker, et al 2007) we can imagine the C-store approach in two logical levels: user-oriented logical tables and so called projections. Projections are collections of columns each sorted on some attribute(s). We denote this fact in the relational schema by a delimiter and a list of sort attributes. For example, \(R(A, B, C \mid B)\) means that projection \(R\) is sorted on \(B\).
One column can occur in more projections. Consequently, a redundancy occurs in C-store database which means that any compression seems to be appropriate for columns. At a logical level C-store supports standard tables equipped by the primary key and, possibly, a set of foreign keys. A C-store projection is anchored on a given logical table and retains all duplicate rows, i.e. it has the same number of rows as its anchor table.

In fact, C-store physically stores projections, i.e. C-store considers a logical table as a set of materialized views. The columns of projections are stored column-wise, using separate storage for each column. Because of the presence of updates. Fortunately, there is a lot of rather static XML data which narrows the use of C-store.

3 Low-level Memory System

We will consider a linear tape consisting from particular positions as model of classical disk memory. A position is addressable and can contain a data item. Besides positions we will also consider segments of fixed size. Each segment can contain more data items. Each data item can share more segments (see Figure 1). We will present a mutual correspondence between a position on the disk and a segment on the tape. Let $C$ be the number of disk cylinders, $S$ be the number of sectors on one disk track and $H$ be the number of disk heads. Further let $c \in \{0,\ldots, C-1\}$, $s \in \{0,\ldots, S-1\}$ and $h \in \{0,\ldots, H-1\}$. Then the triple $<c, s, h>$ uniquely determines a position on the disk. The associated segment is determined as follows:

$$segment_{c,s,h} = c*S^2 + s*H + h.$$ 

We have introduced segments as intermediate stage between physical disk operations and operation over the tape with positions.

Imagine now that the read head is moving over the tape. It can read the content from its current position. Let $i$ be arbitrary position on the tape and let $j, k$ be positions different from $i$ such that $|j-i| > 1$ and $|k-i| = 1$. If $j$ is the current position on the tape, we call the reading the position $j$ a jump, whereas reading the position $k$ we consider as reading a neighbour. Let $i$ be a current position on the tape. For comparison of time complexity of particular algorithms we will use the following operations:

- $jump(j)$ – reading the random position $j$ from the tape,
- $next()$ – reading the next (or the previous) position $(i \pm 1)$ on the tape.

$next()$ operation has time demands dependent on a current position. In fact, a position is non-addressable on disk. Each position holds a concrete value of one attribute of one table row. Implementation of mapping positions on segments influences the execution time of $next()$ operation and, mainly, of $jump()$ operation.

We will consider $next()$ and $jump()$ as basic operations for estimating time complexity of particular algorithms. Now we introduce a set of basic operations that we will be used in all these algorithms:

- $SetProjection(P)$ – setting up a projection (or a mapping table) $P$ as current (up to the next setting all operations go on over data in this projection).
- $SetCol(C)$ – setting up the $C$ column (in current projection) as current.
- $Search(X)$ – finding the first occurrence of $X$ item in current projection and column. At the same time, the record, where the item is found, is set up as current. The function requires several executions of $jump()$ or $next()$ operation dependent on the current projection is primarily ordered current column or not.
- $Read(I)$ – reading $I$th record in current projection and column. At the same time, $I$th record is set up as current. This function corresponds exactly to one execution of $jump()$ operation.
- $Next()$ – reading the next record in current column. At the same time the next record is set up as current. This operation agrees with execution of operation $next()$.
- $Previous()$ – reading the previous record in current attribute. At the same time, the previous record is set up as current. Complexity of this operation is the same as of execution of $next()$ operation.
- $ReadCol(C)$ – reading value in $C$ column (in the same projection) of the current record.

Two mapping algorithms used in the next sections will be evaluated based on how the associated data models are optimal for evaluation of an axis query. We will restrict only on retrieval of all nodes belonging to the chosen axis of a context node. Algorithms searching nodes of particular axes will be completed by a calculation of their time complexity on the level of abstract disk operations $next()$ and $jump()$.

![Figure 1: Segments and positions of items](image)
4 XML Data in C-store

First, we chose a method how to represent XML data in a relational database. It is clear that two aspects of the design have to be considered:

- a logical database schema enabling the reconstruction of the XML data structure
- a physical database schema used for storing XML data in C-store

We will use only one logical table. The physical tables are projections of the logical table and anchored on it.

There are a lot of methods concerning the first issue as it is shown by Mlýnkova and Pokorný (2005). For example, in the work (Boncz et al, 2006) we can find a schema based on coding preorder and postorder traversal in one logical table. The basic logical table contains attributes containing the number of nodes in the subtree below the node, and \( level \) storing the distance from the tree’s root. In combination with so called \( \text{staircase join} \) this approach can significantly support XPath processing, particularly in R-store environment. Here we use the approach described in work (Kuckelberg and Krieger 2003), where a mapping of XML document into a relational database by other structure-centred mapping is described. For this method, of the main importance is the knowledge of the list of child nodes for each node from XML tree. This is in accordance with an intuition that a support for path reconstruction should be at disposal. Particularly in C-store, we will ensure that representations of these nodes will be close to each other a column. There are several approaches, how to implement this list.

We use combination of two methods for storing data into C-store:

- **FK (Foreign Key Method)** – This method uses a unique identifier for each node of XML tree and a foreign key reference to its parent (see Figure 2).

- **DF (Depth First Method)** – This method uses traversing XML tree in a depth first manner. It stores to each node of XML tree a couple of values (min, max), that are assigned by traversing tree in a depth first manner. Suppose a counter which is increased each time another node is visited. After entering a node \( U \) in first time its min value is set to the current counter value, after last leaving the \( U \) node the current counter value sets the max value of \( U \). Counter is increased each step by one (see Figure 3). If \( U \) is an arbitrary node, we denote its min and max values as \( \text{min}(U) \) and \( \text{max}(U) \), respectively.

\[
\begin{align*}
\text{DF(ID, Type, Name, Value, Parent, Min, Max, Min_of_Parent, FK1, FK2)}
\end{align*}
\]

\[\text{FK1 (ID, Type, Name, Value, Parent, Min, Max, Min_of_Parent, FK2)}\]

\[\text{DF (ID, Type, Name, Value, Parent, Min, Max, Min_of_Parent, FK1)}\]

\[\text{FK2 (ID, Parent, FK2)}\]

\[\text{Observe that FK1 is sorted by Parent and secondary by Min. Further, we use mapping tables FK2>>DF and FK2>>FK1.}\]

5 XML Axes Querying

In this section we describe algorithms for computing axes for an XML tree mapped into relations FK1, DF, and FK2. For each algorithm we provide an analysis of its time complexity. Let \( n \) be the number all nodes of XML tree and \( id \) be the identifier of the current (context) node \( N \). We will denote the cardinality of the result set, i.e. the cardinality of axis of \( N \). We will also work with value sequences. So, we introduce one important notion for situations when some values lie in the sequence consecutively. Let \( A = a_1, a_2, \ldots, a_n \) be a sequence. Let \( M \) be a subsequence of \( A \). Then \( M \) is a contiguous subsequence of \( A \), iff there is \( i \leq n \) such that \( \{a_i, \ldots, a_{i+|M|-1}\} = M \).
5.1 Child

To retrieve all children of N we use sequential reading of one attribute that is trivial for C-store. We want to retrieve all nodes, whose parent is N. We use projection FK1, which is sorted on attribute Parent. Due to ordering we easily find the first occurrence (row number) of searched child and all other occurrences are in contiguous subsequence behind it. By reading the corresponding sequence of ID values we obtain the query result.

Algorithm 5.1 (Child axis)
Input: id of the input node N
Output: the set D containing all children of N

1. D = {}
2. SetProjection(FK1); //Finding corresponding records
3. SetCol(Parent);
4. r = id; start = Search(id);
5. while r != id:
6. (r,stop) = Next();
7. stop = stop - 1;
8. SetCol(ID); //Relational projection on ID attribute
9. D.add(Read(start));
10. while start < stop:
11. (d,start) = Next();
12. D.add(d);

Algorithm complexity. Finding the first child requires one Search() in column Parent with complexity O(log n). Finding the rest of children takes O(m) Next() operations.

5.2 Descendant

For finding the set of all descendants of N, FK approach would lead to recursive calling of relational operation join. Therefore we use DF approach and its mathematical properties. We are searching for the set of all nodes, which have been visited during navigation XML tree in a depth first manner values min and max of particular nodes have been assigned in this order:

min value of N, successively, all min and max values of all its descendants, min value of N, min and max values of nodes in the following axis for N.

Nodes in array A are arranged by the min value of DF method, i.e. in the order:

N, all N descendants, nodes in the following axis for N.

The first node P fulfilling the conditions Statement 1 is obviously the node in the following axis of N (it fulfills (1) and does not fulfill (2)). Hence each node PI appearing in array A behind node P is also a node in the following axis and, consequently, can not be its descendant.

A consequence of Statement 1 is, that although the tuples in projection DF are not ordered by the Max attribute, we need not to test the Max values of all nodes with minimum greater than Min of N. Namely, once we retrieve the first node in the following axis, we can stop the searching, since we most certainly will find no node on the child axis. Finding the first node, which is not a descendant of N, depends on the moment, when (2) stops to hold (even in by maximum non-ordered array).

Algorithm 5.2 (Descendant axis)
Input: id of the input node N
Output: the set P containing all descendants of N

0. P = {}
1. SetProjection(FK2); //Finding N node
2. SetCol(ID);
3. start = Search(id);
4. SetProjection(FK2>>DF); //Finding its position in DF through mapping table
5. SetCol(position);
6. start = Read(start);
7. SetProjection(DF); //Finding all descendants as sequence start...stop
8. SetCol(Max);
9. Max = Read(start); m = Max;
10. while m <= Max:
11. (m,stop) = Next();
12. stop = stop -1;
13. SetCol(ID); //Relational projection on ID attribute
14. Read(start); //No storing - start is position of N
15. while start < stop:
16. (p,start) = Next();
17. P.add(p);

Algorithm complexity. Finding the N node requires one Search() execution with complexity O(log n), since it is performed on ID in FK2 projection, hence on an ordered sequence. Thus, the time complexity of this function is O(log n). After that N was found in FK2 projection and consecutively through a mapping table in DF projection, retrieval of all its descendants requires O(m) Next() operations.
5.3 Parent

Algorithm for finding the parent of the node N is trivial. It uses the only FK2 projection. There, with logarithmical complexity, the position of the context node is found and the value of Parent attribute on the associated position. However, this approach uses no property of column-wise storing; conversely a use of row store would entail smaller number of disk operations.

Algorithm 5.3 (Parent axis)
Input: id of the input node N
Output: id_r of the parent of N

0 SetProjection(FK2); //Finding N node
1 SetCol(ID);
2 Search(id);
3 id_r=ReadCol(Parent); //Reading value in Parent

Algorithm complexity. Time complexity of function Search() requires O(log n) jump() operations. The use ReadCol() operation instead of Read() enables a query optimization natively more suitable for row-stores.

5.4 Ancestor

Storing tables column-wise is appropriate for certain types of queries. The optimal way, how to approach data stored in such way is to read one column and based on its values to decide about query result. As was apparent from the previous algorithms, beneficial is when the result members appear in a contiguous sequence. In this case the time complexity of the algorithm is the same as the time complexity finding the start of this sequence and passing just the set of resulted nodes. During reading the sequence no additional steps are necessary.

Statement 2. There is no algorithm mapping XML tree nodes on such sequence P, that for arbitrary node the set of all its ancestors is a contiguous subsequence of P and, moreover, each node of XML tree appears just once in P.

Proof: Proof is trivial. Assume that there exists such algorithm. Then it should map the tree in Figure 2 on a node sequence P in such way, that \{1,2\}, \{1,3\}, \{1,4\} are contiguous subsequences of P and in the same time node 1 appears in P just once. Such sequence does not exist.

A consequence of Statement 2 is that during searching all ancestors of N in XML tree it is not possible to exploit advantage of sequential reading (not even using any other method implementing the list of following nodes than DF or FK).

Now, we describe two algorithms finding all ancestors the context node. The first one is based on the FK method, the second one on the DF method (Sections 5.4.1-2). Then we will discuss advantages and disadvantages of both algorithms (Section 5.4.3). Finally, we describe an algorithm, which combines both approaches and reaches better results (Section 5.4.4).

5.4.1 Algorithm FK

Remind that both FK and DF methods describe a way, how to implement for a tree structure the lists of childern. FK method is based on knowledge of the parent of each node. In the previous sections, every time when we use FK method, we used attributes ID (as the primary key) and Parent (as the foreign key). But FK method can be also viewed in other way. We use attribute Min as the primary key and Min_Parent as a foreign key. Then we use projection DF, instead of projection FK1 or FK2. Obviously, all principles of FK method will be preserved:

- Min attribute is really the unique node identifier,
- Min_Parent attribute contains values of Min attribute or NULL (if the node has no parent), so it is really a foreign key.

Moreover, DF projection is ordered by the Min attribute. Consequently, for finding all ancestors of N the use of FK method over projection FK2 is equivalent to the use of FK method over projection DF. One iteration of searching the parent of N, its grandparent, etc., means in both cases searching in sorted column (ID or Min, respectively) and reading the value, which searched in the next iteration (Parent or Min_Parent, respectively).

The algorithm uses the projection FK2 and the mapping table FK2>>DF for finding N in projection DF. Then it uses above described FK method over projection DF. The preference of projection DF over projection FK2 serves as the basis to for easy integration of the algorithm with Algorithm 5.5 into a combined algorithm (see Section 4.4.4).

Algorithm 5.4 (Ancestor axis – FK method)
Input: id of the input node N
Output: the set P containing all ancestors of N

0 P = {}
1 SetProjection(FK2); //Finding N node
2 SetCol(ID);
3 pos = Search(id);
4 SetProjection(FK2>>DF); //Jump to projection DF
5 SetCol(position);
6 pos = Read(pos);
7 SetProjection(DF);
8 SetCol(Min_Parent);
9 min_r = Read(pos);
10 while min_r is not NULL: //Finding parent, grandpater,...
11 SetCol(Min);
12 pos = Search(min_r);
13 min_r = ReadCol(Min_Parent);
14 if min_r is not NULL:
15 P.add(ReadCol(ID));

Algorithm complexity. First, the algorithm retrieves the node N in projection FK2 by function Search(). First two jumps are performed (for approaching Min_Parent column of DF projection). The algorithm is then recursive. In each recursion level, we first find out the given node by Search() operation in sorted Min column of DF projection. Then we read in Min_Parent column of the same projection the minimum of parent of this node. This minimum becomes the _current_ in the next level of recursion. Finding the nearest ancestor takes O(log n) executions of operation jump(). This process repeats m-times.

5.4.2 Algorithm DF

This algorithm uses properties of sequential reading. Sequential reading of current column can be done by two
ways – with the next() or the previous(). For our purposes we consider both approaches as identical. In practice, previous() operation can be easily transformed on next() operation by creating the projection ordered in the reverse order. To be consistent with Algorithm 5.6 we use the previous() operation in this algorithm. But the algorithm uses projection FK2 as well as the mapping table FK2 >> DF for finding N node and projection DF retrieval of its all ancestors.

Algorithm 5.5 (Ancestor axis – DF method)
Input: id of the input node N
Output: the set P containing all ancestors of N

0 P = {}
1 SetProjection(FK2); //Finding N node
2 SetCol(ID);
3 stop = Search(id);
4 SetProjection(FK2>>DF); //Finding its position in DF through mapping table
5 SetCol(position);
6 stop = Read(stop);
7 SetProjection(DF); // Reading maximum of N node
8 SetCol(Max);
9 Max = Read(stop);
10 i = stop; // Testing all nodes with Min value less than Min of N node on Max condition
11 while i > 1:
12 (m,i) = Previous();
13 if m > Max:
14 P.add(id);
15 SetCol(ID); //now P_pos contains a list of positions, we need list of IDs
16 i = stop; id = Read(i);
17 while i > 1:
18 (id,i) = Previous();
19 if i in P_pos:
20 P.add(id);

Algorithm complexity. Finding N, i.e. executing Search() function in attribute ID of projection FK2, has complexity $O(\log n)$ jumps on disk. After finding N we have first to execute O(1) jumps on disk to reach the Max column of DF projection and to read Max value of N. Finally, the algorithm reads at most n members in a contiguous sequence, which means $O(n)$ executions of operation next().

5.4.3 Comparison of FK and DF algorithms

We can see that both algorithms for finding node N have the same time complexity. Thus, we will compare complexity of finding the set of ancestors of N.

Algorithm 5.4 consists of $O(m*\log n)$ jumps on disk. An advantage is that the algorithm reads no superfluous data. A disadvantage is that items read are not in a contiguous sequence, i.e. jumps are needed. On the other hand, Algorithm 5.5 reads more items (including that ones not belonging to the result set), but its advantage is, that these items are stored in a contiguous sequence. The time complexity of such reading is $O(n)$.

Suitability of particular algorithms can be judged by two factors – „density“ of sequence read by DF method and technical parameters of disk. Sequence density determines how many data in sequence is relevant for the query result. The higher the sequence density is, the more usable DF method is. Decreasing the density means that DF method is less and less effective up to certain moment (dependent on technical parameters of disk) when m jumps will take less time than processing $n$ next() operations.

Also of importance is that density of sequence read by DF method can be different in different parts of the sequence. Knowledge of the densities where nodes have minimum less than N leads to idea to combine approaches FK and DF for finding all ancestors of N. In sequence parts with low density Algorithm 5.4 is used, for parts with high density we use Algorithm 5.5. Of course, the knowledge of densities has to be obtainable without additional reading the sequence. In such case, the algorithm effectiveness could be worse than that one of Algorithm 5.5 itself. In the next section we describe the combined algorithm in detail including implementation of decision, which of two approaches should be used.

5.4.4 Combined algorithm – „Jump and go“

The name of algorithm reflects the movement that the algorithm reminds – jumping alternated with walking step by step.

Algorithm is based on recursive findings the parent of given item, grandparent, etc. It decides dynamically in each iteration, whether it will search out the parent of current item by binary search (Search() operation) over sorted Min column (jumping), or via reading sequence of several items appearing in projection DF immediately before the current node (walking).

Now we introduce a metric for measuring the density of a given sequence part. We can observe that the sequence part on which we decide contains nodes appearing in projection DF between the parent of current node (including) and the current node (out of it). Thus, this sequence contains several (≥2) nodes not relevant for the query and the only one node (parent current node) relevant for the query. The number of non-relevant nodes is essential for the sequence part density. It corresponds exactly to the number of nodes appearing in projection DF between the current node and its parent.

Statement 3. Let all nodes of XML tree be stored in an array A sorted by the min value of DF method. Let U be a node and R its parent. Denote by $p$ the number of nodes appearing in array A between nodes R and U. Then

$$p = \frac{\min(U) - \min(R) - 1}{2} \quad (3)$$

Proof: During traversing XML tree in a depth first manner min and max values of particular nodes were assigned in this order:

- min value of R, successively all min and max values of all nodes appearing between nodes R and U, min value of U.

Simultaneously, the counter of DF method was increased by $2*p$ between assigning $\min(R)$ and $\min(U)$. Thus,

$$\min(R) + 2*p + 1 = \min(U) \quad (4)$$

Then (4) immediately implies the equality (3).
A consequence of Statement 3 is, that for setting the
density of the nearest actual sequence we need to know
only Min value the current node and Min value of its
parent. Both these values are stored in projection DF, so
we need not to read the sequence and know its density in
unit time.

To be able to chose either jumping or walking for a
given sequence part, we have to compare their time
demands. Jumping works similarly as Algorithm 5.4.
Assuming, that we just read the values of Min and
Min_Parent attributes for current node, we use only its
corresponding part – finding position, on which its parent
appears. Then a new iteration follows, i.e. the decision
process, which approach will be applied.

Walking is based on Algorithm 5.5. First, we read the
value of Max attribute of the current node and then values
of Max attribute for all nodes, appearing between the
current node and its parent (these nodes have the Max
value less than the current node). The parent is
recognized, when the value of Max attribute is greater
than Max value of the current node. In this moment we
stop the searching. Again a new iteration the algorithm
follows.

Assume the current node U, its minimum min(U) and
minimum for its parent min_r(U). Finding the parent
node U by jumping requires

\[ \log(n) \times t_u \]

where, \( t_u \) is average time needed by a jump on disk.
In practice, we can the jumping method even improve in
such way, that we search out only in those part of DF
projection before the current item (see Section 5.4.2). The
number \( n \) would be lower in each step. Finding the parent
node U method need (see Statement 3)

\[ \min(U) - \min(R) - \frac{1}{2} + 1 \times t_u \]

where \( t_u \) is average time operation next (). The numbers
\( t_u \) and \( t_s \) are constant for the disk used, \( n \) is constant pro
each XML file. Comparing values (5) and (6) we can
determine which approach is for sequence actual more
appropriate:

- if value (5) is less, then we apply the jumping method,
- if value (6) is less, then we apply the walking method.

Finally, we introduce a formal description of the
complete algorithm „Jump and go“.

**Algorithm 5.6 (Jump and go)**

Input: \( id \) of the input node \( N \)

Output: the set \( P \) containing all ancestors of \( N \)

\[
\begin{align*}
&0 \quad P = \{} \\
&1 \quad \text{SetProjection}(FK2); //Finding N node \\
&2 \quad \text{SetCol}(ID); \\
&3 \quad \text{pos} = \text{Search}(id); \\
&4 \quad \text{SetProjection}(FK2 >> DF); //jump to 
&\quad \text{projection DF} \\
&5 \quad \text{SetCol}(position); \\
&6 \quad \text{pos} = \text{Read}(pos); \\
&7 \quad \text{SetProjection}(DF); \\
&8 \quad \text{SetCol}(Min\_Parent); \\
&9 \quad \text{min}_r = \text{Read}(pos); \\
&10 \quad \text{while min}_r \text{ is not NULL: } //Finding 
&\quad \text{parent, grandparent,}... \\
&\quad \text{+Decision what to chose} \\
&11 \quad \text{min} = \text{ReadCol}(Min); \\
&12 \quad \text{if } \log(\text{konst.n}) \times \text{konst.ts} <((\text{min} - \\
&\quad \text{min}_r - 1) / 2 + 1) \text{konst.ts}: \\
&\quad \text{SetCol}(Min); //Jumping \\
&13 \quad \text{pos} = \text{Search}(min_r); \\
&15 \quad \text{P.add(ReadCol(ID));} \\
&16 \quad \text{else:} \\
&17 \quad \text{m} = \text{max} = \text{ReadCol(Max);} //Walking \\
&18 \quad \text{SetCol(Max);} \\
&19 \quad \text{while pos > 1 and m <= max:} \\
&20 \quad (m,pos) = \text{Previous();} \\
&21 \quad \text{if m > max:} \\
&22 \quad \text{P.add(ReadCol(ID));} \\
&23 \quad \text{SetCol(Min\_Parent);} \\
&24 \quad \text{min}_r = \text{Read}(pos); \\
&25 \quad \text{...stop}
\end{align*}
\]

5.5 Sibling

For finding all siblings of the node \( N \) we again use
sequential reading of one attribute. We use projection
FK2, in which we approach the value of Parent
attribute for \( N \). Siblings are nodes, which have this value
the same. Using mapping table FK2>>FK1 we find the
occurrence of \( N \) in FK1. Due to the secondary ordering
the records by attribute Min, we know that the sequence
of items with the same value of Parent attribute
appearing before the given node in projection FK1
corresponds to its younger siblings and the sequence
behind the item corresponds to its older siblings.

**Algorithm 5.7 (Sibling axis)**

Input: \( id \) of the input node \( N \)

Output: the sets \( M \) and \( S \) containing all younger siblings
and older siblings of \( N \), respectively

\[
\begin{align*}
&0 \quad M = \{} \\
&1 \quad \text{SetProjection}(FK2); //Finding N node \\
&2 \quad \text{SetCol}(ID); \\
&3 \quad \text{start\_stop} = \text{Search}(id); \\
&4 \quad \text{SetProjection}(FK2 >> FK1); //through 
&\quad \text{join-index} \\
&5 \quad \text{SetCol(position);} \\
&6 \quad \text{start\_stop} = \text{Read(start\_stop);} \\
&7 \quad \text{SetProjection(FK1);} // FK1 \\
&8 \quad \text{SetCol(Parent);} \\
&9 \quad \text{parent} = \text{Read(start\_stop);} \\
&10 \quad r = \text{parent}; //Finding set M = \\
&\quad [\text{start, ...start\_stop}] \\
&11 \quad \text{while r == parent:} \\
&12 \quad (r,start) = \text{Previous();} \\
&13 \quad \text{start} = \text{start} + 1; \\
&14 \quad r = \text{parent}; //Finding set S = [\text{start, ...stop} \\
&\quad \text{stop} = \text{Read(start\_stop);} \\
&\quad \text{Finding set S = [start, ...stop] \\
&16 \quad \text{while r == parent:} \\
&16 \quad (r,start) = \text{Next();} \\
&17 \quad \text{stop} = \text{stop} - 1; \\
&18 \quad \text{SetCol(ID);} //Relational projection 
&\quad \text{on ID attribute} \\
&19 \quad i = \text{Read(start);} \\
&20 \quad \text{while start < stop:} \\
&21 \quad \text{if start < start\_stop:} \\
&22 \quad M.add(i); \\
&23 \quad \text{if start > start\_stop:} \\
&24 \quad S.add(i); \\
&25 \quad (i,start) = \text{Next();}
\end{align*}
\]

**Algorithm complexity.** Operation Search() has
complexity \( O(\log n) \). After finding \( N \) in projection FK1
the time complexity of selection of all siblings (operations Next() / Previous()) is \(O(m)\).

**Remark:** Algorithm 5.7 has found all siblings of \(N\) as a union of all younger and older siblings. A trivial modification of the algorithm enables to restrict its result set only to younger or older siblings without changing the time complexity of the algorithm.

### 5.6 Following (Preceding)

Let \(U\) be a node of XML tree. While for arbitrary node \(V\)

\[
\min(V) < \min(U),
\]

holds, then \(V\) is either ancestor or preceding node of \(U\). The difference between ancestor and preceding node is in value \(max\). Whereas the preceding nodes have \(max\) value less than the node \(U\), the ancestors have \(max\) value greater than node \(U\). To be preceding the next condition (8) has to hold together with the condition (7) for the node \(V\):

\[
max(V) < max(U).
\]

Otherwise, if for arbitrary node \(V\)

\[
\min(V) > \min(U)
\]

and

\[
max(V) > max(U),
\]

then node \(V\) follows node \(U\).

Notice that finding the result set \(M\) leads to calculation of a set difference. Let \(Min_inv\) be a set of nodes that fulfil the minimum invariant, i.e. the condition (7) for preceding axis and the condition (9) for the following axis. Further, let \(Max_not\) be the set of nodes that do not fulfil the maximum invariant (condition (8) and (10), respectively). Thus, \(Max_not\) is the set of ancestors (as concerns searching preceding) or of descendant (as concerns searching following). The query result is then the set

\[
M = Min_inv \setminus Max_not.
\]

Now we introduce a universal algorithm for searching both preceding and following nodes. The algorithm uses \(DF\) projection for finding the result set. It also uses projection \(FK2\) and mapping table \(FK2 >> DF\) for finding the node \(N\). The algorithm finds all preceding nodes or all following nodes, depending on which couple of operations \(Init\) and \(MaxInvariant\) is used.

<table>
<thead>
<tr>
<th>Preceding</th>
<th>Following</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Init(\text{position}))</td>
<td>(Init(\text{position}))</td>
</tr>
<tr>
<td>(return(1, \text{position}))</td>
<td>(return(\text{position} + 1, \text{N} + 1))</td>
</tr>
<tr>
<td>(MaxInvariant(\text{max}))</td>
<td>(MaxInvariant(\text{max}))</td>
</tr>
<tr>
<td>(\text{return max &lt; MAX})</td>
<td>(\text{return max &gt; MAX})</td>
</tr>
</tbody>
</table>

Operation \(Init()\) implements minimum invariant, i.e. it returns couple \((\text{start, stop})\) in projection \(FK1\), \(MAX\) denotes the \(Max\) value of \(N\).

**Algorithm 5.8 (Following, preceding axes)**

**Input:** id of the input node \(N\)

**Output:** the set \(M\) containing all preceding nodes (respectively following nodes) of \(N\)

```plaintext
0  SetProjection(FK2); //Finding N node
1  SetCol(ID);
2  position = Search(id);
3  SetProjection(FK2>>DF); //through
```

Algorithm complexity. Let \(p\) be the number of nodes in the \(Max_not\) set. The retrieval of \(N\) node needs \(O(\log n)\) jumps on disk. For obtaining the \(M\) set itself, the algorithm takes yet \(2^m (m+p+1)\) executions of \(Next()\) function.

### 6 Conclusions and future work

We can conclude with the following statement summarizing complexities of algorithms described in Section 5.

**Statement 3.** Let \(n\) be the number of nodes of XML tree, \(m\) be the number of nodes in an axis, and let \(p\) be the number of nodes not fulfilling the maximum invariant specified for following (preceding) axis. Axes queries over C-store modelled by relations specified in Section 4 have the complexities given in Table 1.

Analyzing these \(O\) expressions from proofs described in Section 4, we can observe that the functions behind the formal complexities have mostly the constants \(a\) and \(b\) in leading terms \(a^m\) or \(b^\log n\), respectively, equal to 1.

Of course, as it is mentioned by Častková (2009), the algorithms can be further optimized, particularly those ones for following (preceding) axes. Also it is possible to propose a different logical as well as physical model for relations implementing C-store. Another direction of future research is a use of indexes supporting queries over C-store. For example, authors of (Sidiouargos et al, 2008) show that with proper clustered indices the triple-store for RDF data performs better than the vertically-partitioned approach.

Analyzing these \(O\) expressions from proofs described in Section 4, we can observe that the functions behind the formal complexities have mostly the constants \(a\) and \(b\) in leading terms \(a^m\) or \(b^\log n\), respectively, equal to 1.

An important part missing in our paper concerns experiments with real XML data. Theoretical time complexities should be confirmed by real complexities gained with a help a column-oriented DBMS, e.g. C-store. Clearly, it should be done in accordance with our low-level memory system which seems to be sufficiently general and, consequently, usable for describing memory...
### Table 1: Complexity of algorithms for XPath axes

<table>
<thead>
<tr>
<th>Axis</th>
<th>#occurrences of <code>next()</code></th>
<th>#occurrences of <code>jump()</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>child</td>
<td>(O(m))</td>
<td>(O(\log n))</td>
</tr>
<tr>
<td>descendant</td>
<td>(O(m))</td>
<td>(O(\log n))</td>
</tr>
<tr>
<td>parent</td>
<td>0</td>
<td>(O(\log n))</td>
</tr>
<tr>
<td>ancestor - FK method</td>
<td>0</td>
<td>(O(m \log n))</td>
</tr>
<tr>
<td>ancestor - DF method</td>
<td>(O(n))</td>
<td>(O(\log n))</td>
</tr>
<tr>
<td>sibling</td>
<td>(O(m))</td>
<td>(O(\log n))</td>
</tr>
<tr>
<td>following</td>
<td>(O(m+p))</td>
<td>(O(\log n))</td>
</tr>
<tr>
<td>preceding</td>
<td>(O(m+p))</td>
<td>(O(\log n))</td>
</tr>
</tbody>
</table>

The system in C-store. On the other, real column-oriented DBMSs can have a different physical data model than the one used in our research. For example, MonetDB provides the model based on the technique described in (Boncz et al, 2006) and mentioned in Section 4.

Another direction comes from the situation when we have an XML schema for XML data. First observations discussed by Částková (2009) show that knowledge of such schema gives no significant contribution to the design of physical schema design for C-store.

**Acknowledgement** This research has been partially supported by the grant of GACR No. P202/10/0573.

**References**


Analyzing and Improving Table Space Allocation

Shanchan Wu1
Yefim Shuf2
Hong Min2
Hubertus Franke2
Frances H Villafuerte3
Julie Watts3

1 Department of Computer Science, University of Maryland, College Park, MD, USA
2 IBM T.J. Watson Research Center, Yorktown Heights, NY, USA
3 IBM Silicon Valley Lab, San Jose, CA, USA

Abstract
Space allocation is a fundamental operation performed by a database management system (DBMS) when it inserts a record into a table. A good space allocation algorithm quickly locates and reserves enough space for a record, places it closer to its related records, and utilizes the available space. Satisfying these conflicting requirements is challenging and trade-offs are carefully balanced by well-chosen heuristics. As a DBMS evolves over time, especially a commercial DBMS, its space allocation algorithm gets more sophisticated and complex and relies on many heuristics. Technological changes, new applications, and greater data volumes render many legacy heuristics ineffective. These factors hinder understanding of space allocation behavior under many workload conditions and make it difficult to enhance the algorithm without causing performance regressions for some of the workloads.

To facilitate research and study the performance of a table space allocation algorithm of a modern DBMS in real-world workload scenarios, we build an extensible simulation framework. We analyze algorithm behavior and make surprising observations. We use the findings to further improve the existing algorithm by proposing algorithm enhancements and showing their benefits with respect to key performance metrics. In conclusion, the proposed framework has been effective in research to understand the performance, improve the space allocation algorithms, and to guide the developers of a commercial DBMS.

1 Introduction
For a database management system (DBMS), the ability to insert a record quickly and efficiently is critical. To maximize disk space utilization while achieving high performance, database designers use various strategies for storing records. As improvements in processor speed continue to outpace improvements in disk access time, I/O is increasingly a major bottleneck in systems and especially in large DBMS (Hsu et al. 2001). An efficient space allocation strategy that minimizes I/O frequency is essential. One example is to maintain data clustering so that disk seeks can be reduced for sequential reads. However, a mix of inserts and deletes can easily cause disk space fragmentation.

Designing a table space allocation strategy that performs well for various workloads is challenging. The tradeoffs like “add more empty pages to a table space” or “fully utilize the existing space” are not easily explored via the back of an envelope analysis. Another requirement for a space allocation algorithm is to have fewer contensions in a highly concurrent transaction processing environment. We are not aware of any framework for space allocation algorithms research.

In this paper, we study a representative table space allocation algorithm of a commercial DBMS and its variations. To compare different space allocation strategies, we build a simulation tool. The tool is used to quantitatively answer “what if” questions that arise during a space allocation strategy design and identify the strengths and weaknesses of the strategy. The tool is also used to pinpoint when record insert performance degrades. We use the tool to analyze the algorithm under various conditions representing real workload scenarios and to evaluate our algorithm enhancements. It can be used to evaluate the implication of using Solid State Disks (SSDs) (Agrawal et al. 2008) whose performance characteristics are different than Hard Disks (HDs). Simulating various input patterns and their effect on the insertion algorithm is a complicated modeling task and the tool that helps accomplishing it has a considerable practical value.

Insert performance is one of the most challenging issues in real-life usage of DBMS. Ability to complete new data imports in a given time frame and to do so in a space usage efficient manner is a key characteristic of a well performing DBMS. Our work addresses key challenges in solving this important problem. This paper makes the following contributions:

- It presents an extensible framework for simulating space allocation algorithms and evaluating them with respect to various performance metrics on multi-threaded workloads. The framework is used as a testbed to explore ideas for improving space allocation algorithms and gain insights into how such algorithms behave in real-world workload scenarios.
- It evaluates a typical table space allocation algorithm of a commercial DBMS and identifies conditions under which the algorithm performs the best. As far as we know, it addresses issues not previously investigated in the literature: it studies the quality of the cluster ratio achieved and the scaling of the algorithm when facing a large number of concurrent threads.
- It presents observations with practical performance implications. We find that when record insertion is guided by a clustered index, a random record sequence can be inserted faster if it is pre-sorted in the clustered index key order. We show that pre-sorting records is a way to improve the cluster ratio. We also show that providing each thread with a distinct starting point when searching for space can reduce contentions.
- It proposes space allocation algorithm enhancements and quantifies benefits. We show one enhance-
The rest of the paper is organized as follows: in section 2, we present related work; in section 3, we describe the organization of a table space and a table space allocation algorithm of a modern DBMS; in section 4, we describe our simulation framework; we use the framework to exercise several workload scenarios to study the space allocation algorithm in section 5; in section 6, we propose algorithm enhancements and experimentally show their benefits; in section 7, we discuss how the performance metrics collected by our framework can be applied to analyze the actual cost of a record insert in a real system; we summarize in section 8.

2 Related Work

Research on table space allocation algorithms, while being important to the database community, has seldom been presented in the literature. Although there has been work on space management in 1996 (McAuliffe et al. 1996), in spite of technological changes, emergence of new applications, and demands for rapid loading of high volumes of data, the problem of space allocation has not been given much attention since (McAuliffe et al. 1996) and must be revisited.

McAuliffe et al. (McAuliffe et al. 1996) studied object placement algorithms, their storage utilization and allocation performance without regard to clustering. Their work focused on free space management in heap files. They noted that many object placement algorithms have serious performance deficiencies, including excessive CPU or memory overhead, I/O traffic, or poor disk utilization. Compared to (McAuliffe et al. 1996), we focus on multi-threaded workloads. We use a more comprehensive set of performance parameters for analyzing performance costs from two orthogonal aspects: one is dictated by the underlying hardware and DBMS, and one is algorithm driven.

Our goal is to improve DBMS performance by reducing the CPU cost of allocating space for a record and creating a better data layout reducing I/O delays. There has been work on improving I/O performance from different angles: disk access optimizations, prefetching, architectures for storing large volumes of data, and layout optimizations (Hsu et al. 2005). Disk drive models were studied (Ruemmler & Wilkes 1994) and I/O simulation tools were developed. Sorting of RIDs was used to reduce I/O for bulk deletes (Gärtner et al. 2001). To our knowledge, we are the first to build a research tool for studying the effects of input patterns on the insertion algorithm behavior.

Data prefetching was shown to reduce synchronous I/O operations. Hsu et al. (Hsu et al. 2002) examine the logical I/O reference behavior of the peak production database workloads from ten of the worlds largest corporations and analyzed factors that affect how these workloads respond to different techniques for caching, prefetching, and write buffering. Wilson et al. (Wilson et al. 1995) discussed the design of dynamic memory allocators. Due to the popularity of Flash memory as a data storage medium, some began designing Flash-based DBMS (Lee & Moon 2007). Our tool can be used to explore algorithms for systems with Flash-based storage.

There has been work on improving DBMS performance by reducing resource access contentions, mostly at the transaction level. To ensure data integrity, locking schemes were proposed (Felber & Reiter 2002, Silberschatz & Kedem 1980). Several concurrency control techniques were investigated (Felber & Reiter 2002, Reuter 1982, Gawlick & Kinkade 1985). The contentions addressed in the literature are contentions on accesses to records and indexes of databases, and not contentions during table space allocation, which is this papers focus. The methods to improve concurrency and space utilization by space reservation and tracking are described in (Mohan & Haderle 1994).

3 Table Space And Table Space Allocation Algorithm

Figure 1 shows table spaces and tables in a database. Since table spaces reside in database partition groups, the table space selected to hold a table defines how the data for the table is distributed across database partitions. A single table space can span several containers. Containers define physical storage for a table space. A container can be a file system directory, a file with a preset size, or a raw device such as an unformatted disk, a disk partition, or a logical volume. Multiple containers from one or more table spaces can be created on the same physical disk. For better performance, each container can reside on a different disk.

Figure 2 shows the structure of a table space. A table space contains multiple segments. Each segment contains a number of pages. The typical page types are: a header page, a space map page (SMAP), a compression dictionary page (if data is compressed), and a data page. The header page describes the table space. A space map page identifies data pages with enough free space for new records. Each space map page may contain several pages. A SMAP uses an indicator (several bits) for each data page to indicate the level of free space on that page. When a table space does not have enough available space to accommodate a new record, an extension operation may occur, in which case, new space is allocated at the end of a table space.

There is an index structure (an index tree) to keep track of the order and locations of records in a table. The index is updated once a record is inserted into a table. The number to the left of each page in Figure 2 indicates page...
number of that page in a table space. The location of a record is defined by page number of the data page where the record is located and the offset of the starting point of the record in the data page.

The table space allocation algorithm operates as follows:

1. For a new record to be inserted, look up the record’s key in a key index tree of the table to find a desired location (i.e., a candidate data page) for the record to be placed at. If the record’s key value does not exist in the index, the nearest key value in the index is used for identifying the candidate page.
2. If the placement in step 1 fails, find space within the same segment where the candidate data page is located.
3. If failed in step 2, search from the first segment that has free space forward to the last segment covered by the same space map page. Note: A reference to the first segment that has free space is updated when necessary.
4. If failed in step 3, go to the last segment of the table space. Search from the first page of the last segment to the last page of the last segment.
5. If failed in step 4 and if allocating new space will not cause an extension, then allocate a new page.
6. If failed in step 4 and if allocating new space will cause an extension, then do an exhaustive search from the first segment that has free space to the end of the table space. A reference to the first segment that has free space is updated when necessary.
7. If failed in step 6, allocate a new page with an extension.

An ideal table space allocation algorithm should be able to quickly find enough free space for a record, waste no space, and maintain data clustering which is measured by “cluster ratio” to indicate how closely the records’ physical placement ordering matches the logical ordering of certain keys. To maintain a high cluster ratio, the algorithm tries to place a record in the order of following vicinities if possible: index directed candidate page, or within the same segment of candidate page, or in a segment covered by the same space map page.

A good starting point to analyze performance of a space search algorithm is to identify major factors influencing its performance. The major factors for record insertions are I/O operations and contentions. The I/O operations include reading space map pages (SMAP) and data pages from a disk storage system. Since the size of a buffer pool is finite and is usually much smaller than the size of a corresponding table, a larger number of (random) page fetching operations usually results in more misses in a buffer pool and more I/O operations. To reduce I/O operations, each thread maintains in memory one recently used SMAP page.

The number of data pages and SMAP pages that are fetched before finding free space to insert a record.

The number of page latches or locks that are contended with during a space search.

Our simulation tool tracks these and other performance metrics.

4 The Simulation Framework

In this section, we describe the architecture of the simulation framework / tool. To reduce memory footprint, the tool does not store the content of records, but only keys and record sizes. The tool inputs record sequences and outputs statistical results. The input sequences are generated by an input workload generator (or by using the instrumentation features of a DBMS). It generates various input streams with configurable attributes.

Each thread in this multi-threaded tool operates on a separate input sequence. Many parameters such as time to insert a record, wait time for a latch, the properties of a table space, are configurable. An index is implemented as a B-tree. To find a candidate page from an index, we look for a record with the same key, or the nearest higher key, or the nearest lower key. The index is updated once a record is inserted. We implemented a Lock-Manager to simulate concurrent access issues and contentions.

Figure 3 describes the structure of the simulation framework.

The tool is used as follows. After the input generator generates desired sequences of input records, they are saved. Then the simulator is started to concurrently process input sequences with multiple threads and collect statistical data for performance metrics. At the end of a simulation, the state of a table space can be saved on a disk. The next time, if we want to continue from a particular state of a table space, we can configure the tool to load the saved state of a table space and process more input streams. This lets us to use identical initial states for different experiments.

5 Experimental Analysis Of The Algorithm

In this section, we analyze the characteristics and performance of the algorithm with different workloads. Intuitively, it is faster to find space for a new record when a table space has a substantial amount of free space than when a table space is almost filled. To study how the performance metrics of the algorithm change when the state of a table space changes, we simulate a real banking application example where a table space grows from being sparse to being full. We find a particular state when performance metrics deteriorate sharply and propose techniques to mitigate this performance problem. We also investigate
how the characteristics of input record sequences affect algorithm, performance. The study of this scenario will help us make decisions on whether or how to preprocess input sequences before inserting records. For this purpose, we compare and analyze performance metrics for the ordered record sequences and unordered record sequences.

5.1 A Banking Application Example

Let us consider a database design for a representative banking application workload. The data is organized by a clustering index on a data attribute such as account number. There is a fair amount of free space left on each page. When there is account activity, the banking application closes the old account record by updating its ending timestamp, and inserts a new record for the same account, preferably near the account record that has just been closed.

For this type of a database and application design, it is expected that the table space, which initially has a lot of free space, will grow full over time as more transactions are processed and more records are inserted. The application expects table space reorganizations and extensions to be performed periodically to space out records in the table space. However, when some accounts are more active than others, free space around these accounts becomes scarce and new records corresponding to these accounts are placed elsewhere. This prolongs a space search process for those records. The quality of data clustering degrades.

We simulate this workload and investigate performance issues when the account access pattern is skewed. The table space is pre-populated with records corresponding to all bank accounts, with every page having some free space. After that, concurrent streams periodically insert records into the table space. When there is sufficient free space, a record can be inserted quickly. When the available space decreases, it takes longer to place a new record.

5.1.1 Experimental Analysis

The initial state of a table space is created by inserting a sequence of records with non-duplicated sequential keys corresponding to all account numbers. Each page is left with 80% of free space (20% of space is occupied by account data). We use the 80/20 rule and designate 20% of accounts as very active accounts that generate 80% of account activities. The remaining 80% moderately active accounts generate 20% of activities. We create input sequences representing this skewed account activity pattern. Each sequence consists of sub-sequences representing daily bank activities. Each daily sub-sequence covers 2.5% of distinct accounts and 365 sub-sequences (365 days) make a long sequence of records. The records in each sub-sequence can fill 0.5% of space of the initial table space.

The performance metrics of the experiment are shown in Figure 4 through 8. In Figure 4 through 7, x-axis value corresponds to a particular sub-sequence (a particular day). For example, x=50 is the 50th subsequence or the 50th day. Figure 4 shows where the records are inserted into candidate pages, other pages in the same segment, corresponding to algorithm steps described in Section 3) over time, represented by a percentage of records in a sub-sequence. For example, when x=1 (the first day), almost 100% of records are placed on candidate pages. When x is around 160, the initial table space is almost full. Figure 5 shows the average number of pages checked in SMAP per record insertion. Figures 6 and 7 show the average number of fetching operations per record insertion. Figure 8 shows that the cluster ratio is worse in the final state compared to the initial state.

The experimental results show that during the transition of a table space from the almost full state to the full state, the performance metrics are significantly worse. A few sparsely located remaining free space slots in a table space cause a long exhaustive search. After the transition state, the few sparsely located remaining free space slots are filled and the indicator to the first segment that contains available space is shifted to the location near the end of a table space, which reduces the cost of an exhaustive search. A table space may also be transitioning from the full state to the almost full state because of deletion operations. The deletion operations can create sparsely located free slots and the algorithm will try to find those empty spots when new records are inserted. This can lead to a long exhaustive search.

5.1.2 Mitigating Performance Problems

To mitigate performance problems during the transition state, we evaluate ideas that use heuristics to avoid an exhaustive search. One is to avoid an exhaustive search by anticipating and detecting symptoms of the pre-transition state proactively. Another is to stop performing an exhaustive search if a table space is almost full.

Avoiding an Exhaustive Search by Anticipating and Detecting Symptoms of the Pre-Transition State Proactively:

For this banking application workload example, if it can be detected by inspecting performance metrics (using a performance reporting facility) that a transition state will occur soon, then we can reorganize a table space proactively before more records are inserted. Reorganizing involves sorting all data in a table space, repopulating the table space with added space, and leaving a reasonably high percentage of free space on each page. For example, suppose the detected time is day 150 (i.e., 10 days before
a table space is full). Once a table space is reorganized, we continue inserting the remaining data, i.e. data of day 151 through day 365. The results are shown in Figure 9 through Figure 13.

A comparison of Figure 4 to 8 with Figure 9 to 13 indicates that performance metrics improved significantly (after reorganizing a table space shortly before the transition state). The cluster ratio in the final state is better – it improved over the one without proactive reorganization. Of course, reorganization comes with its own cost in terms of time and space. So the cost/benefit of reorganization and its impact on data insert performance and query performance need to be considered and balanced.

**Avoiding an Exhaustive Search of an Almost Full Table Space:**

There is an alternative to the previously proposed idea of performing reorganization in the pre-transition state. During the insertion, after detecting that a table space is almost full and anticipating that allocating new space will cause an extension, the algorithm can be changed to skip an exhaustive search, and directly allocate a new page with an extension. When a table space is almost full, the probability that a new record will be inserted into an initial candidate page is low. This is one of the heuristics we can use to detect whether a table space is almost full. We set a threshold of 30% to perform our experiment, i.e., a table space is almost full if no greater than 30% of records in a particular time window (e.g. one day) are inserted into candidate pages. The results are in Figure 14 through Figure 18.

Comparing the sizes of the final table space (after all records were inserted) in the original algorithm and in the modified algorithm, we notice that the difference in table space sizes is very small, 0.999989 : 1. This means that applying this scheme will not cause much more space to be consumed. There are no pulses in Figures 15 through 17. In contrast, in the original algorithm, the pulses appear in Figures 5 through 7. These pulses indicate that substantially more work needs to be done to find free space during the corresponding time period. Other than the pulses, the values of other parts of the curves are similar. So the scheme improves performance, with respect to performance metrics, during the transition state of a table space while maintaining performance metrics during other states of a table space.

**5.1.3 Discussions**

The behavior anticipated by an application is to insert data into the pages dictated by an index, the “candidate pages”. When the percentage of candidate page placements becomes very small, extensions and re-organizations should be performed to add more disk space, re-cluster records, and space out records in a table space. Failing to anticipate this transition early enough, as shown, could lead to a big performance degradation which is followed by a steady state of sub-optimal record insert performance.

To mitigate the problem using approaches proposed in section 5.1.2, we can collect performance metrics during the insertion process to predict an upcoming transition state. Several statistical events can be used to indicate the approach of a transition state: (1) the percentage of records that can be inserted into candidate pages decreases quickly, (2) the percentage of records that are inserted into other pages in the same segment where the candidate page is located increases and then decreases, (3) the percentage of records inserted into other segments covered by the same SMAP increases and then decreases, and then (4) the percentage of records that are inserted during an exhaus-
tive search increases sharply. When an approaching transition state is predicted by these indicators, corresponding actions can be taken such as to advise a database administrator to perform a table space re-organization or start an automatic online table space re-organization.

The analysis of the algorithm suggests that in addition to the I/O cost of space search, the CPU cost can be a factor. We find there can be a noticeable CPU cost associated with scanning SMAP pages which are likely cached in a buffer pool. In the algorithm we examined, a performance bottleneck associated with a space search is largely related to the number of pages visited.

5.2 Ordered vs. Random Sequences

To reduce a time window to load data into databases, we investigate whether ”massaging” data prior to loading can reduce the load time. One way to pre-process data is to sort it. In this section, we investigate whether we should order records by key values before loading.

To answer this question, we first analyze the characteristics of indexing. When using an index, the index is consulted before an insertion and then updated after the insertion. At the beginning, when both a table space and an index are empty, the first record is inserted into the first page in a table space and the index is updated. When the page for a record pointed by the index is not available (due to insufficient space or held latches), a table space search algorithm is invoked. Over time, the table space grows gradually, with space near the beginning being slowly filled and leaving most of available space near the end of the table space.

Due to the characteristics of the cluster indexing, different insertion behaviors are observed with ordered and random input sequences. Since a record always gets the candidate page number of the nearest key in the index, when inserting a record of ordered sequences, the algorithm will likely first try a page (a candidate page) near the end of a table space where the pages are likely to have free space. When inserting a record of random sequences, a candidate page pointed by an index can potentially be anywhere in a table space. So we hypothesize that the algorithm finds free space faster for ordered sequences. We validate our hypothesis.

5.2.1 Experimental Analysis

We generate input sequences consisting of random permutations of distinct keys. The number of concurrently processed input sequences (N) is varied in our experiments. Each sequence in an experiment is of the same length. Each thread processes a different input sequence. We compared the results of two different types of inputs: (1) all input sequences are ordered before they are inserted; (2) all input sequences are in a random key order. The simulation results for N = 10, 25, 40 and 55 are as follows.

Figures 19 and 20 show performance metrics on where the records are inserted in a table space. It can be seen that when input records are ordered (vs. random), they are more likely to be placed on candidate pages and the algorithm is less likely to search for space in the last segment in a table space. Overall, more records in ordered sequences are placed successfully during the first three steps of a space allocation flow (described in Section 3) than in random sequence. This is an indication that the algorithm performs better on ordered input records.

Figures 21 through 24 show the comparison of the average number of page latch hits per record insertion in ordered sequences and in random sequences are similar when N=10 and 20. When N increases to 40 and 55, the average numbers of page latch hits per record insertion in ordered sequences are less than those in random sequences. Figure 22 through 24 show that the average number of pages checked in the SMAP, the average number of fetching data page operations, and the average number of fetching SMAP operations per record insertion with ordered sequences are all significantly smaller (i.e., better) than those with random sequences. Altogether, the data in Figures 21 through 24 further suggests that, to min-
imize the time to insert records into a table space, it is advisable to order the records before inserting. Furthermore, Figure 25 shows that the cluster ratio of a table space is better when input sequences are pre-sorted.

6 Algorithm Enhancements

In the previous section, we studied the algorithm using inputs with different characteristics. In this section, we address issues of heavy contents (often present in multi-threaded environments) and I/O frequency. Our goal is reducing contentions and I/O operations while maintaining or improving a cluster ratio. In this section, we propose three techniques that improve the space search algorithm. We show benefits with respect to performance metrics. The three enhancements can be combined together, but for the purpose of an analysis we evaluate them separately.

6.1 Reducing Contentions

6.1.1 The Observed Problem

When multiple threads try to insert records into a table space, contentions on accesses to resources can have a significant impact on performance. A thread waiting on a resource protected by a lock will have to wait for the lock to be released before it can proceed. Modern DBMS use fine grain locking and latching to reduce contentions during record insertions and updates.

After investigating the table space allocation algorithm, we found that contentions can be frequent during the search through the last segment of a table space. The following explains the reason for heavy contentions during the search through the last segment. (a) If a table space is empty or almost empty (i.e., each data page has plenty of free space), when a page is selected by an index as a candidate page, there is a high probability that a record can be inserted into that page. Even if a record cannot be inserted into that candidate page (perhaps because other threads filled it), it is still likely that the record can be inserted into a page in the same segment or a page covered by the same SMAP. In this situation, the performance is not a significant concern. (b) However, when a table space is almost full (i.e., only a few pages have enough free space for a new record), a thread will have to search through many pages before successfully inserting a record. Eventually, if no space is found, the algorithm allocates one or more new pages at the end of a table space. Hence, the pages in the last segment are more likely to have free space than pages in other segments. Therefore, when a table space is almost full, a new record is more likely to be inserted into a page in the last segment. In the original algorithm, when searching in the last segment, all threads start from the same page and are likely to find the same page with free space at the same time. The first thread that gets the page will locks, performs space chaining, and inserts into it. The other threads checking the same page will contend and wait until the page is unlocked.

6.1.2 A Proposed Enhancement

We propose the following technique to reduce contentions in the last segment. Instead of letting all threads traverse through the same sequence of pages from the same starting page in the last segment, select a random page within the last segment as a start searching page for each thread. Consequently, the first page found to be available in the last segment by different threads will tend to be different. Hence the chance of all threads contending on the same page in the last segment will be reduced.

Besides reducing contentions, we also consider reducing the number of page fetching operations by assigning a random offset number for each thread when a thread starts.

6.2 Using “Recent History Lookup List”

6.2.1 A Proposed Enhancement

To minimize the waste of space, a table space allocation algorithm uses free space in a table space as much as possible before allocating more space at the end of the table space.

A SMAP (space map page) is a structure that tracks the level of available space in every data page, with each data page represented by several bits in a SMAP. To find a page...
with enough free space, each thread has to scan through all SMAP bits, including those representing full pages.

A key approach to speed up searching for free space is to track only pages that have free space. However, if we build a separate global structure to remember all data pages which are not full (having free space to hold the shortest record), there might not be enough memory to hold the structure and it is preferable not to store it on a disk due to the cost of I/O operations. An alternative is to keep track of a small subset of pages that have enough free space; these few pages can be stored in main memory. A page in the small subset is re-used for free space until it is full and replaced by another page. The reused page in the subset also has better locality than a page identified by scanning SMAP. Consequently, we propose a data structure to hold a few available pages and corresponding strategies to access the structure when looking for space and updating the structure. We use RHL_LIST (Recent History Lookup List) to denote this structure as pages in the list are recently found available and used for insertion.

We design the RHL_LIST structure as follows. The RHL_LIST structure is a fixed size array. Each item in RHL_LIST contains a pointer (reference) to a data page, a pointer (reference) to a SMAP page that is relevant to this data page, and a status flag indicating the status of the page in RHL_LIST. There are three possible status states for each item: AVAILABLE, TRASH, BUSY. The AVAILABLE status of an item indicates that the page in this item is not currently occupied by any thread and there is enough free space on this page. The TRASH status indicates that the page does not have enough free space and can be replaced by another available page. The BUSY status of a page indicates that the page is currently occupied by a thread.

There are three main operations on the RHL_LIST structure: (i) to update the status of an item, (ii) to get an available page through the items in the RHL_LIST, and (iii) to insert a new item into RHL_LIST with the pointer to a new page which has enough free space and a pointer to a relevant SMAP.

When a page is obtained from the RHL_LIST, the status of the item where the page is located in the RHL_LIST is set to BUSY. After the page is processed by a thread, the status is set to be AVAILABLE or TRASH depending on the available space on that page. If the available space is greater than the maximum size of the records of the table, then it will be set to AVAILABLE, otherwise, it will be set to TRASH. The status of an item may also be changed from AVAILABLE to TRASH during the get_one_available_page() operation by a thread. When a thread traverses through the RHL_LIST, even if it finds the status of an item to be AVAILABLE, it has to check the space on the page to see whether it is indeed AVAILABLE (because the available space of that page may be changed by some threads without accessing the RHL_LIST, i.e., via other parts of searching). If the available space is smaller than necessary to hold the record, the status of that item is changed to TRASH. When a record is successfully inserted into a page and after that the available space of that page is still greater than the maximum size of the records, we try to insert a new item having a pointer to that page into RHL_LIST. The insertion will not be successful if there are no TRASH pages in RHL_LIST. This operation is inexpensive because it is in-memory and no extra objects are created. As the latch time on each item in RHL_LIST is very short, the contention on RHL_LIST items is not an obvious performance concern.

The data pages (and the corresponding SMAP pages) that are referenced via RHL_LIST are more likely to be in memory. When inserting records into those pages, we do not have to fetch them from a disk and it reduces I/O operations.

![Figure 31: The average numbers of pages checked in SMAP.](image)

![Figure 32: The average numbers of page latch hits.](image)

![Figure 33: The average numbers of fetching page operations.](image)

![Figure 34: The average numbers of fetching SMAP operations.](image)

![Figure 35: Cluster ratios.](image)

### 6.2.2 Experimental Results

We generate sequences of distinct key values; each sequence having the same number of records. We conduct experiments using 10, 25, 40 and 55 sequences (concurrent threads). We compare performance metrics of the original and the enhanced algorithm using the RHL_LIST. The experimental results are shown in Figures 31 through 35. Although cluster ratios do not change, other performance metrics (the number of pages checked and the number of pages fetched during a space search) improve significantly. In Figures 31 through 34, the curves for the algorithm using RHL_LIST are almost flat, while the curves for the original algorithm are growing quickly. This demonstrates that the enhanced algorithm using RHL_LIST has better scalability.

### 6.2.3 Discussion

Earlier in this section, we proposed an enhancement to the allocation algorithm and showed that referring to the recent history lookup list (RHL_LIST) improved record insert efficiency. The RHL_LIST is a structure to keep a small set of recently visited data pages with free space. Attempting to insert records directly into these pages reduces time searching for free space.

As an alternative to tracking data pages with free space, we can keep (in a data structure) a set of SMAP pages each of which indicates that at least some of their data pages have free space. The modified space search algorithm first checks the candidate page, then tries to search in some SMAP pages in that structure, then tries to search in the last segment, and then proceeds as in the original algorithm. The structure to keep a set of SMAP pages needs to be kept up to date. When and how to update the structure as well as a performance analysis are left for the future work.

### 6.3 Reducing Search for Available Space

#### 6.3.1 A Proposed Enhancement

In the original algorithm described in section 3, when inserting a new record, before making a decision to allocate
more space at the end of a table space for a new record, a potentially long search has to be made to better utilize existing space. In a workload with variable record sizes, as a result of frequent inserts and updates, the free space is largely fragmented into small empty slots (where larger records cannot be placed). For this scenario, our intuition is that (a) for large records, we should find a way to shorten a search path and (b) for small records, we can still try to use the original search path to maximize space utilization.

We propose Conditional Append Algorithm (CAA) enhancement. (a) For a large record, we first try a candidate page; if we cannot insert the record in the candidate page, we skip both the search for pages in the same segment and the search for pages covered by the same space map page. We attempt to place the record into one of the last M pages at the end of the table space. If we cannot successfully insert the record into one of the last M pages, we allocate a new page for this record. If the new allocation needs an extension, then the exhaustive search before the extension is also bypassed. (b) For a small record, we follow the original search path, except when searching in the last segment, we use the last M pages to replace the last segment. Whether a record is large or small, when searching in the last M pages, we apply the same technique described in section 6.1 and select a random start page for a search in the last M pages.

For our experiments, we define large size records as those whose sizes are greater than the average record size. We find that CAA has better performance characteristics than the original algorithm. When properly selecting the parameter M (which indicates the number of pages at the end of the table space where large records are placed), CAA and the original algorithm consume similar amounts of table space. CAA improves the performance metrics for the following reason: (1) It directly reduces a search path for large records; (2) The availability of small slots unused by large records in the last M pages shortens the time to find space for small records; (3) Randomizing a starting lookup page for each thread helps reduce contentions.

We find experimentally that in high contention environments, when the parameter M is too small (compared to the number of threads), CAA might underutilize space. The reason is that when many threads with large records fail to find space in the last M pages at about the same time, there is a chance these threads start to allocate new pages concurrently. When the number of these new pages is significantly larger than parameter M, some new pages allocated by threads will reside outside the range of last M pages from the end of the table space. As a result, they are not available for the search step in the last M pages dictated by CAA. Consequently, these pages might be under filled in the immediate future. Other than the above observation regarding very small M values, our experiments show that the value of M does not significantly affect performance characteristics.

### 6.3.2 Experimental Results

We generate a number of sequences of distinct key values; each sequence having the same number of records. The sizes of the records we use are distributed between 100 and 250 bytes. The average record size is 175 and a standard deviation is 35. We conduct experiments using 10, 25, 40 and 55 concurrent threads. Parameter M is set to (number of threads)/2. We compare performance metrics of the original algorithm and CAA. The experimental results, shown in Figures 36 through 40, demonstrate that CAA has better performance metrics. To illustrate how parameter M affects space utilization, we evaluate CAA by varying M and the number of threads. Figure 41 shows space utilization over various values of M and the number of threads. To understand how the space utilization is related to the ratio of (M : number of threads), we display data presented in Figure 41 in a different format shown in Figure 42. We observe that when M exceeds (number of threads / 2) the space utilization gets greater than 90%. In conclusion, we can leverage this observation: a DBMS can make observations on the number of concurrent threads and dynamically adjust parameter M to ensure it operates in the mode to achieve the highest space utilization.

### 7 Performance Factors Analysis

In the two previous sections, we evaluate the table space allocation algorithm and a few enhancements with respect to several performance metrics. In this section, we further analyze how the performance metrics relate to performance. The ultimate performance measures are average response times and throughput for record inserts (while maintaining good space utilization). Some previous work (McAuliffe et al. 1996) used “objects created per second” metric to analyze the performance. This is a throughput measure and by itself does not provide enough details to explain how the throughput is affected by the cost of each step of the space search process. In addition to providing a throughput measure, our framework allows us to get a breakdown on the cost of each step of the space search process and identify bottlenecks by plugging-in hardware and DBMS dependent parameters. We explain how it can be done using the performance model below:

\[
\text{avg \_cost \_to \_insert \_a \_record} = (\text{cost \_of \_index \_look \_up}) + (\text{cost \_of \_check \_a \_page \_in \_SMAP}) + (\text{avg \_num \_Pages \_Checked \_in \_SMAP}) + (\text{cost \_of \_per \_allocation \_operation})
\]

Figure 42: Space utilization with respect to #_of_last_pages/#_of_threads (M/N).
The parameter values in parentheses, which depend on DBMS and hardware, are system parameters. The remaining parameters are algorithm parameters. With fixed system parameters, reducing the values of algorithm parameters by algorithm changes, as shown in previous sections, avg\_cost\_to\_insert\_a\_record decreases and performance improves. The system parameters we use are: cost\_of\_index\_look\_up is the time cost of the primary index look up. We can make an assumption that non-leaf index pages are cached in a buffer pool. A percentage of leaf pages area accessed from disk units while the other pages are cached in a buffer pool. cost\_of\_check\_a\_page\_in\_SMAP is the cost of checking SMAP if a page has enough space (i.e. the cost of checking a few bytes). cost\_of\_per\_allocation\_operation is the cost to allocate a page (or several pages depending on the allocation scheme) at the end of the table space. It is an amortized cost between allocations that need extensions and those that do not. cost\_of\_waiting\_on\_a\_latch is the cost of waiting for a latch until the waiting thread can access the resource protected by the latch. This cost is related to the speed of an operation on the contented resource and the waiting scheme. If the waiting scheme is always wait until success then it is mainly related to the operation speed on the contented resource (a page). cost\_of\_fetcing\_a\_data\_page is the cost of fetching a page from a disk or memory. Typically, some data pages are fetched from disk units while other data pages are cached in a buffer pool. cost\_of\_fetcing\_a\_SMAP\_page is the cost of fetching a SMAP page. If we assume that all SMAP pages are kept in a buffer pool, then it is a memory operation. Otherwise, its cost is similar to cost\_of\_fetcing\_a\_data\_page, fetching a SMAP page with a percentage of pages coming from disk units and other pages cached in a buffer pool. cost\_of\_insertion\_into\_a\_page is the cost to insert a record into a memory resident page. It includes finding the free block via an offset table on the same page and then locating the space to insert. Three system parameters, cost\_of\_fetcing\_a\_data\_page, cost\_of\_index\_look\_up, cost\_of\_fetcing\_a\_SMAP\_page include the cost a page read from disk units. This cost includes time of transferring data on I/O channel, reading data from the cache in disk units, and potential disk seek time if the page is not found in the cache. In a cost analysis using the performance model above, we can make reasonable assumptions on the percentage of pages read from a cache based on the sizes of a cache and a database.

8 Summary

The problem of space allocation in a DBMS remains important. In this paper, we study the performance characteristics of a table space allocation algorithm of a modern DBMS, make observations about its behavior with practical performance implications, identify opportunities for optimization, propose and evaluate algorithm enhancements, and quantify their benefits with respect to performance metrics. To conduct this research, we build an extensible simulation framework. In our study, we look at the space allocation problem from a new angle and consider factors like contentions and cluster ratios. We show that pre-sorting data leads to better insertion performance which has implications for load utilities. We believe to be the first to build a research tool for studying the effect of input patterns on space allocation algorithm behavior in a DBMS. Our framework is flexible and can be used to explore SSD optimized algorithms. This simulation framework is used by both database researchers and developers for (i) design exploration and (ii) using real-world workload patterns to identify high-value optimizations and avoid performance regressions. It allows us enhance the algorithm in response to new requirements.

References


Potentiality of Power Management on Database Systems with Power Saving Function of Disk Drives

Norifumi Nishikawa\textsuperscript{1,2}, Miyuki Nakano\textsuperscript{1}, and Masaru Kitsuregawa\textsuperscript{1}
\textsuperscript{1}Institute of Industrial Science, the University of Tokyo
\textsuperscript{2}Systems Development Laboratory, Hitachi, Ltd.
292 Yoshida-cho, Totsuka-ku, Yokohama 244-0817, Japan
\{norifumi,miyuki,kitsure\}@tki.u-tokyo.ac.jp, norifumi.nishikawa.mn@hitachi.com

Abstract
Power consumption of modern datacenters is increasing rapidly. Databases, especially OLTPs, have become a major application at datacenters. Therefore, power-saving management for OLTP applications has become an important task for user budgets and datacenter operations. A recent report described that disk drives consume about 70\% of total power of IT equipment when such large OLTP applications are executed. As described in this paper, we specifically examine a novel power-saving management for multiple disk drives of OLTP applications. First, we constructed an experimental system with power meter, and measured basic power consumption and I/O behaviors of OLTP applications. Then we show that basic measurement results confirm there is still the potentiality of power-saving even though OLTP applications are running. We propose a new method that delays database writes based on this I/O behavior knowledge at run time. Experimental and simulation results obtained using our power-saving methods are explained. Our method provides dynamic power saving of disk drives running TPC–C applications, which issue many I/Os to the disk drives.

Keywords: Online transaction processing, OLTP, Disk drive, Power-saving

1 Introduction
Digital data produced by human beings are increasing every day. “How Much Information?”\textsuperscript{1}, a report published by the University of California, San Diego states that the amount of digital data will be measurable in Yotta-Bytes (YB) worldwide by 2025 (Bohn, Short 2009). This explosion of digital data must be managed and used by data-intensive applications such as customer management systems and online transaction processing systems (OLTP). The rapid growth of digital data necessitates that these applications be readily extensible and that they provide high performance in processing digital data.

Now, these large data-intensive applications run at datacenters. The total management cost of datacenters increases annually. Especially, the cost for power and cooling has increased drastically. An IDC report (Eastwood, Bozman, Pucciarelli, and Perry 2009) describes that power and cooling costs were only 10\% of hardware costs in 1997, but these costs will account for 75\% of hardware costs in 2011.

IT equipment of datacenters includes servers, storage, and networks. Although the power consumption of storage is only approximately 30\% of that of IT equipment (Rajecki 2008), our investigation specifically examines power saving for storage because main applications of datacenters are data-intensive applications which require many more disk drive spindles than other applications do. One report of the literature (Poess, and Nambiar 2008) describes that the power of disk drives for large online transaction processing systems (OLTPs, data-intensive applications) accounts for approximately 70\% of the total power of IT equipment. Consequently, disk drive power-saving is important for maintaining good low power consumption of datacenters.

Many storage and disk drive power consumption methods have been proposed to date. Typical methods control the timing of the transition of the power status of disks or control the rotation speed of disks (Douglas, Krishnan, and Bershad 1995, Helmbold, Long, Sconyers, and Sherrod 2000, Gurumurthi, Sivasubramaniam, Kandemir, and Franke 2003, Zhu, Chen, Tan, Zhou, Keeton, and Wilkes 2005, Gniady, Hu, and Lu 2004, Son, Kandem, and Choudhary 2005), extend the duration of the idle period to use a disk’s power-saving functions by controlling the I/O interval (Papathanasiou, and Scott 2004, Li, and Wang 2004, Yao, and Wang 2006, Heath, Pinheiro, Hom, Kremer, and Bianchini 2002), and extend the idle period by replacing data to disks (Colarelli and Grunwald 2002, Li and Wang 2004, Pinheiro and Bianchini 2004, Weddle, Oldham, Qian, and Wang 2007, Verma, Koller, Useite, Rangaswami 2010). However, these methods are implemented inside storage, or use only storage access statistics gathered at storage.

Recently, other power-saving approaches based on applications such as DBMS have been proposed along with hardware-based approaches for storage and servers. One report (Harizopoulos, Shah, Meza, and Ranganathan 2009) presented a discussion of DBMS and its system configurations, which enable reduced power consumption of IT equipment based on experiments using a TPC–H database. An earlier study (Poess, Nambiar 2010) described the relation between performance and power consumption when varying the system configuration of a TPC–H database. Results obtained using these approaches suggest that the utilization of application knowledge offers power-saving potential for use in storage systems.

The application workload, however, changes according to the number of users and the data size; an adaptive power...
consumption mechanism must therefore follow the dynamic variation of the application workload. A previous report (Poess and Nambia 2010) presented one answer to datacenter system design and construction problems, but no solution can be applied to a datacenter that is already in service. Another discussion (Harizopoulos, Shah, Meza, and Ranganathan 2009) of the power-saving approach of DBMS includes no quantitative evaluation of the effect of DBMS oriented power-saving approach and its side effects on DBMS performance.

As described in this paper, we investigate the potential of power saving of multiple disk drives used for OLTP application. Because OLTP applications constitute a major workload at datacenters, power management of OLTP application is expected to achieve great power savings at datacenters. However power management of OLTP is unlikely to be realized because OLTP is a high-I/O-load application of DBMS. It is quite difficult to realize power saving of OLTPs at run time. We have already reported preliminary experience results of power saving of TPC–C with two disk drives (Nishikawa, Nakano, and Kitsuregawa 2010).

The contribution of this paper is (i) to clarify the potential of run time power management of OLTP application based on their I/O behaviors, (ii) to propose a method that delays database writes based on this I/O behavior knowledge at run time, and (iii) to measure the actual power consumption with running TPC–C benchmark. Then we show that dynamic power saving of DBMS can be achieved without performance degradation.

Section 2 presents related works. Section 3 presents characteristics of disk drive power consumption. Section 4 presents a description of I/O behavior of TPC–C application on a small system configuration using one disk drive. Section 5 presents a proposed power-saving method using I/O behavior characteristics of TPC–C application. Section 6 explains the examination results; conclusions are presented in section 7.

2 Related Works

In this section, we describe works related to storage and the DBMS-based power-saving method. Power-saving methods for storage have been studied at storage fields. Therefore we first summarize storage-based related works. Then we discuss DBMS-based related works.

2.1 Storage-based Power Saving Method

Storage-based related works are categorized into those addressing disk drive rotation control, I/O interval control, and data placement control. Herein, these storage-based power-saving methods are discussed.

2.1.1 Disk Rotation Control

This approach controls the disk-drive rotation speed or power status. Proposed approaches of disk drive control are categorized into two groups: i) changing the length of the wait time to change the status of disk drive to standby or to sleep (Douglis, Krishman, and Bershad 1995, Helmbold, Long, Sconyers, and Sherrod 2000); and ii) rotating a disk drive at multiple speeds, which specifically requires that the power consumption of a disk drive be lower when its rotation speed is low (Gurumurthi, Sivasubramaniam, Kandemir, and Franke 2003, Zhu, Chen, Tan, Zhou, Keeton, and Wilkes 2005).

Furthermore, another report in the literature (Zhu, Chen, Tan, Zhou, Keeton, and Wilkes 2005) proposes an I/O timing prediction method using a program counter at the OS level. Another report of a study (Gniady, Hu, and Lu 2004) proposes that compilers examine a scientific application source code and embed a disk control code to reduce unnecessary idle time.

Using disk power-saving functions and saving power requires an idle length sufficient to supplement the power loss during the power-on process of the disk drive (approximately 30 s). A typical OLTP processes tens to hundreds of transactions per second; the idle duration of disk drives is usually much shorter than 1 s. Applying these methods, which do not change the I/O timing of OLTP (a typical application of DBMS or DSS), is therefore difficult.

2.1.2 I/O Interval Control

This approach controls the I/O timing of an application to increase the probability that a disk drive is in a power-saving mode. A feature of this approach is to increase the idle period using hierarchical memory architecture such as cache memory (Papathanasiou and Scott 2004, Li and Wang 2004, Yao and Wang 2006), with varying application of I/O issue timing (Heath, Pinheiro, Hom, Kremer, and Bianchini 2002).

These simple cache control methods have difficulty increasing the idle period for I/O intensive application such as OLTP because many I/Os are issued to disks even if the cache hit rate is high. Consequently, it is difficult to apply this approach directly to data-intensive applications.

2.1.3 Data Placement Control

This approach is intended to reduce the disk-drive power consumption by controlling the data placement on disk drives. The concept of this approach is to concentrate frequently accessed data into a few disk drives, then to move the status of other disk drives to standby or to sleep mode (Colarelli and Grunwald 2002, Pinheiro and Bianchini 2004, Weddle, Oldham, Qian, and Wang 2007, Son, Chen, and Kandemir 2005). These approaches determine data placement based on the access frequency of blocks of storage. Recently, the data placement control function is implemented in the storage virtualization layer (Verma, Koller, Useche, Ragaswami 2010). This approach produces replicas of active data chunks into active RAID storage and cuts power applied to other inactive RAID storage.

The information used for these methods is obtained inside storage. Our approach, however, uses application information to calculate the data placement on storage. Applications (or their management software) know what data is accessed and its access timing. Therefore, we believe this application knowledge provides good hints for an appropriate data placement to reduce power consumption.

2.2 DBMS-based Power Saving Method

As described in the Introduction, DBMS-based power-saving methods are quite new. These approaches
demonstrate that utilization of application knowledge can support storage power saving. A few articles discuss the power efficiency datacenter system design and construction solutions (Harizopoulos, Shah, Meza, and Ranganathan 2009, Poess and Nambiar 2010).

However, no such proposal for power-saving methods is useful after the datacenter is in service. These methods have difficulty accommodating the workload variance of datacenters because such power-saving methods depend on hardware design.

3 Characteristics of Disk Drive Power Consumption

For developing the power-saving method for TPC–C, we first analyze disk drive power consumption characteristics. This section explains characteristics of disk-drive power consumption based on actual measurement results.

3.1 Measurement Environment

Fig. 1 presents an outline of equipment used in our power consumption measurement environment. A load-generating PC provides power to a measured disk drive using 4-pin power cables. The voltage of the red cable is 5 V; that of the yellow cable is 12 V. We connected a digital power meter (WT1600; Yokogawa Electric Corp.) to measure the electric current. We also measured voltages between the red cable and the black cable, and the yellow cable and black cable. The disk drive power consumption is their sum.

The load-generating PC CPUs are two Athlon 64 FX-74 3 GHz, 1 MB cache, 4-core processors (AMD, Advanced Micro Devices, Inc.). Main memory sizes of the load generating PCs are 8 GB. Measured disk drives are Barracuda ES ST3750640NS (750 GB, 7200 rpm; Seagate Technology LLC*1). The disk drive write caches are turned off to protect the database reliability because the DBMS uses no write cache.

![Fig. 1: Measurement Environment of Drive Disk Power Consumption.](image1)

3.2 Disk Drive Power Status

A disk drive we used for analyses has a power-saving function that changes the disk drive power status according to the following.

1. Active: I/Os are issued to the disk drive. The disk drive power consumption is highest.
2. Idle: No I/O is issued to the disk drive, but the disk drive serves I/Os immediately.
3. Standby: The disk drive reduces power to rotating disks by parking the head and stopping the disk drive rotation.
4. Sleep: The disk drive parks the head and stops the disk drive rotation. In this status, the disk drive also stops the power supply to the cache. Power consumption is the same as that in standby status, but a disk drive reset is necessary to change the status to active or idle.

We use the disk drive’s active, idle, and standby statuses because the power consumption of sleep status is the same, but it requires an additional operation to change the disk drive status to active or idle states.

3.3 Power Consumption at Active/Idle States

Fig. 2 depicts a relation between the disk drive power consumption and I/Os per second (IOPS). The I/O size is 16 KB, which indicates that the power consumption of random I/O increases in accordance with an increase of IOPS, but it saturates the increase of power where IOPS is larger than 70–80 IOPS.

The power consumption of a sequential write is much less than that of random I/O because the disk drive head movements are far fewer than those of random I/O.

![Fig. 2: IOPS and Power Consumption.](image2)

In Fig. 2, maximum values of IOPS of random I/O are shown to differ by the ratio of read and write percentages of I/Os: maximum random I/O of 100% read I/O is about 90 IOPS; 60% read and 40% write I/O is about 80 IOPS; 20% read and 80% write I/O is about 70 IOPS; and 100% write I/O is about 60 IOPS. The maximum IOPS of the sequential write I/O is about 120 IOPS. Fig. 2 shows only 16 KB I/O size, but the trends of IOPS and power consumption of other I/O size (4 KB, 8 KB, 32 KB, 64 KB, and 128 KB) resemble the trend shown for 16 KB I/O.

3.4 Power Consumption of Standby Status and Break-even Time

Fig. 3 portrays the power consumption of a standby disk drive, along with the transition from active/idle status to standby status, and migration from standby status to active/idle status. Fig. 3 also shows the disk drive break-even time. The break-even time is the interval during which the reduced energy is equal to the added energy of spinning up disk drives.

Figure 3 also shows that the power consumption of a standby state is 1.5 VA/s, which is less than one-fifth of...
the power consumption of the idle status. The transition status from standby to an active/idle, however, requires 8 s and more than an average of 23.3 VA/s. The transition status from an active/idle to a standby requires 3.5 VA/s with 0.2 s. The break-even time calculated from these values is 23.9 s. Therefore, the idle time of more than 24 s is necessary to use this disk drive power-saving function.

For measurements, we created a database, loaded data into the database, and restarted the DBMS to clarify the DBMS buffer. Then we started tpcc-mysql and measured the I/O behavior for 10 min after the throughput of DBMS was stable.

4.2 I/O Behavior Characteristics of TPC–C

Figure 4 shows the number of reads and writes per second and the average I/O intervals of each datum. As presented there, the characteristics of TPC–C I/Os were the following: i) write I/Os to Log data were dominant, ii) I/Os to tables and indexes were fewer than or equal to two I/Os per second, and iii) more than half these I/Os were writes. No I/O was measured to District, Item, or Warehouse data. The I/O intervals of these data (Log, tables and indexes) were shorter than the break-even time (23.9 s), except for NewOrder data.

Figure 5 portrays the intervals of I/Os of data. This figure reveals that the access intervals of OrderLine, Orders, and Stock data were skewed; access intervals were longer than the break-even time. Therefore, a power-saving method using I/O behavior characteristics enables stoppage of disk drives for a long time.

5 Power-Saving Method using I/O Behavior Characteristics of OLTP Application

This section presents a description of our power-saving method. The feature of our approach is to use...
5.2 Power-Saving Method using Delayed Write I/Os

We propose a delayed write I/O method to use long I/O intervals for power-saving of disk drives. This method is based on the database behavior of writing operations.

A DBMS writes database data in the following cases: i) a checkpoint write writes-back dirty DBMS buffer pages to disk drives, and ii) a dirty page flush is issued to acquire a free page into the DBMS buffer. The dirty page flush does not occur when the DBMS buffer has sufficient free space. A checkpoint write is asynchronous to query the execution of DBMS; DBMS enables timing control. We use an asynchronous checkpoint write to hold sufficient time to stop the disk-drive spindle.

Figure 6 presents our proposed method. Our approach produces a long idle period by delaying write I/O database data until the DBMS reads database data on the same disk drive. As shown there, our approach causes no delay of query processing threads. We can spin down the disk drive at this long idle period without degradation of OLTP throughput. The read I/Os are issued to disk drives synchronously because a delay of the read I/O causes a delay of query execution.

The I/O delay of the proposed method is the following:

\[ T = C x (1 - i/f) \]

where \( T \) signifies interval length, \( C \) is a flush C guarantees interval length between two check points. This step also ensures that the number of write I/Os does not exceed 60 I/Os per second. The value of 60 I/Os per second is a maximum number of random write I/Os of disk drives (see section 3.3).
3. Step 3. If write I/Os are delayed more than the interval of DBMS checkpoint, then they must be written to a disk drive. The maximum throughput is 60 I/Os per second, as in step 2.

Our proposed method applies write delay to the DBMS buffer. The delay length is determined dynamically according to the number of dirty pages in the DBMS buffer. The write is issued asynchronously with transaction processing. It is assumed that little impact is given to the transaction response because i) this delayed write method is applied only to inactive disk drives, and ii) there are few data read operations to the inactive disk drives. Consequently, the data flush to these inactive disk drives has little impact to the data read operations issued by transactions. Therefore, these methods are useful for power saving of datacenters after the datacenters are run.

6 Evaluation

We evaluated our power-saving method. First, we present results of our method obtained using a small database on two disk drives. Then we present results from a larger database on five disk drives.

6.1 Evaluation of Two Disk Drives

We evaluated our data placement method based on access frequency using two disk drives. First, we show the actual measurement with a spin down function, which is provided by the Seagate disk drive. Then we evaluated our write I/O delay method using the data placement method during a simulation.

6.1.1 Evaluation of Data Placement Method Based on Access Frequency

6.1.1.1 System Configuration

Figure 7 presents the system configuration used for this evaluation; it is based on the configuration described in sections 3 and 4. We add a SATA disk drive to the configuration (Barracuda ES ST3750640NS; Seagate Technology LLC) and put database data and Log data into two SATA disk drives. The added disk drive is the same model as those described in sections 3 and 4 (Seagate). We connected a digital power meter to the added drive and measured the disk drive power consumption.

The DBMS and DB configurations are as described in section 4. Disk drives are configured to transition to the standby status when the idle period (x) is longer than 5 s, and move to active state when an I/O arrives. We use the minimum standby timeout length (5 s) that is selectable at the operating system. For evaluation, we started tpc-mysql; then we measured the disk-drive power consumption and transaction throughput for 10 min after the transaction throughput was stable.

In Fig. 5, we can observe that almost all idle lengths of Customer, History, and Orders data are longer than the break-even time. Therefore, once the data are accessed, the probability of the next access to the data within a short interval length is low. Consequently, the disk drive should be stopped immediately after the disk drive is accessed. Therefore, we use the minimum standby timeout value (5 s) of the operating system.

6.1.1.2 Data Placement Variation

We assessed four data placements of two disk drives shown in Table 1 to verify data placement method effects. As described section 5, we use some patterns for data placement. The disk drives are of two types: active and non-active. Using this approach, it is likely that non-active disk drives will be put into standby status when no I/O is issued. It is also likely that the standby status will be maintained for a long time. Table 1 shows that disk drive #1 is active and disk drive #2 is not active. We put active data into disk drive #1 and non-active data into disk drive #2. Numbers of I/Os to the disk drive #2 are reduced in accordance with the increase of the case number.

<table>
<thead>
<tr>
<th>Case</th>
<th>Data on Disk Drive #1</th>
<th>Data on Disk Drive #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case #1</td>
<td>Log</td>
<td>Customer, District, History, Item, NewOrders, Orders, OrderLine, Stock, Warehouse</td>
</tr>
<tr>
<td>Case #2</td>
<td>Log, OrderLine</td>
<td>Customer, District, History, Item, NewOrders, Orders, Stock, Warehouse</td>
</tr>
<tr>
<td>Case #3</td>
<td>Log, NewOrders, OrderLine, Orders, Stock</td>
<td>Customer, District, History, Item, Warehouse</td>
</tr>
<tr>
<td>Case #4</td>
<td>Log, Customer, NewOrders, OrderLine, Orders, Stock</td>
<td>District, History, Item, Warehouse</td>
</tr>
</tbody>
</table>

Table 1: Data Placement

6.1.1.3 Evaluation Results of Data Placement Method

Figure 8 presents the actual power consumption of the two disk drives measured using the data placement, i.e., data are distributed into two disk drives in Table 1. Here, we call the results obtained using a spin down function of disk drive as “with power-saving”; the results which the spin down function are turned off are those “without power-saving”. The spin down function is applied only to the non-active disk drive (drive #2 in Table 1).

As shown there, the power consumption without power-saving is nearly equal to 18 VA for all cases. In contrast, the power consumption with the power-saving method differs greatly among the four cases. In case #1, the value is equal to the value without the power-saving method. Therefore, in case #1, the data placement method
does not reduce the disk-drive power consumption. In case #2, the power consumption is increased to 20.2 VA (+10.2% compared with the power consumption without power-saving method). In cases #3 and #4, however, the disk-drive power consumption values (16.6 VA and 14.0 VA respectively) are smaller than those without the power-saving method. In the most efficient case, case #4, the disk drive power consumption is about 23.1% less than that without the power-saving method.

In case #4, the power consumption per transaction is 0.102 VA/transaction, which is more than 40% lower than the value without power-saving. In case #3, the power consumption with power-saving is 0.140 VA/transaction, which is nearly equal to the transaction throughput without power-saving. In case #2, the transaction throughput with power-saving is 144.6 transactions per second, which is more than 22% lower than case #2 without power-saving. However, in cases #3 and #4 with power-saving, reduction of the transaction throughput maintains approximately 10% degradation.

From Figs. 8, 9, and 10, it is apparent that the considerable data placement can achieve large power saving. In case #1, the power-saving function does not reduce power consumption because all of the idle lengths of the disk drives #1 and #2 are less than the standby timeout (5 s). In case #2, the power consumption is increased and transaction throughput is decreased because the idle period length of disk drive #2 is greater than the standby timeout, but shorter than the break-even time (23.9 s). This difference causes an energy loss to spin up disk drive #2 and a transaction delay until the disk drive is spun up. Furthermore, this transaction delay halts I/Os to disk drive #1, causing another energy loss.

In cases #3 and #4, the disk drive power consumption is reduced when using our proposed method because the idle periods are longer than the break-even time (23.9 s). The degradation of transaction throughput results from the wait that is necessary for disk drive #2 to spin up.

### 6.1.2 Evaluation of Delayed I/O

#### 6.1.2.1 Simulation setup

We evaluated our proposed method with a delayed write I/O function. No implementation of a delayed write I/O function exists on commercial DBMS. Therefore, we generate the I/O trace of the write I/O function from I/O traces obtained during experiments of data placement method (Fig. 6). We reorder these I/O traces along with the delayed I/O function policy. We then apply our power-saving function to the generated I/O traces during a certain period described in subsection 5.2. In this simulation, we use i of 60 IOPS and C of 600 s. The value of i is 1.7 IOPS for case #1, 0.5 IOPS for case #2, 0.2 IOPS for case #3, and 0.1 IOPS for case #4.

![Fig. 11: Simulation Method for Write Delay.](image-url)

We measured disk-drive power consumption by replaying the I/O trace on these two disk drives using the `btrecord` and `btreplay` programs (Brunelle 2007). Figure 11 portrays the power simulation method. The evaluation environment is identical to the measurement environment described in subsection 6.1.1.1 and Fig. 7.
This simulation setup measures only the power consumption, I/O response time, and throughput of the disk drives. Therefore we calculate transaction throughput according to the duration from when the transaction is interrupted until the status of disk drives changes from standby to active/idle state using I/O trace and transaction throughput measured in the previous subsection (6.1.1).

6.1.2.2 Simulation Results
Figure 12 portrays results of disk-drive power consumption for each case. Results show the measured power consumption without power saving and with power saving (case #1 – case #4). Here, the power-saving method includes both the data placement method and the delayed write I/O method. Figure 12 shows that the disk drive power consumption is decreased, except in case #1. The maximum reduction of power consumption was 36% for cases #3 and #4; their power consumption levels were 11.4 VA and 11.3 VA, respectively.

Figure 13 presents the power consumption per transaction with and without power-saving methods. As Fig. 13 shows, power consumption is higher in case #1, but the power consumption is lower in other cases (#2–#4). Especially, the power consumption is decreased more than 30% in cases #3 and #4.

Figure 14 presents results of transaction throughput for each case. Transaction throughput was reduced by 6% in case #1, 10.0% in case #2, 8.0% in case #3, and 7.0% in case #4. Results show that our power-saving method only slightly affects the transaction throughput (less than 10%).

The power consumption was increased and transaction throughput was decreased in case #1 for reasons described in the preceding subsection. These results demonstrate that our power-saving methods—the data placement method based on TPC-C I/O behavior characteristics, and the delayed write I/O—radically reduce disk drive power consumption in a small DBMS configuration.

6.1.3 Impacts to Transaction Throughput
The read delay attributable to our power-saving method might affect the transaction response time. Therefore we investigate the read response time of disk drives with application of our proposed power-saving method.

We analyzed the relation between read response time and the number of reads (used trace files gathered at 6.1.1.3) to verify the impact of disk status changes. Figure 15 shows read response times and cumulative distribution function of the number of reads to disk drives when our power-saving functions is applied. As Fig. 15 shows, the number of reads for which the response time is longer than 1 s (which might strongly impact the transaction response time) is only 1.8% of all read I/Os. Moreover, the numbers of reads per transaction to tables and indexes on the disk drives with the applied power-saving function are less than 0.001 (calculated from Table 1 and Fig. 4). Therefore, the number of read I/Os per transaction affected by changing disk drive power status is less than 1/50,000: our proposed method has little impact on the transaction throughput.

6.2 Evaluation of Five Disk Drives
The data allocation variations are limited in the two-disk-drive case. Therefore, we increased the number of disk drives from two to five. We simulated the power consumption of a five-disk-drive DBMS environment to confirm the effect of our method under a larger database configuration.
6.2.1 Simulation Condition

We placed Log data on one disk drive, and Database data on four disk drives. The data size differences among these data are minimized.

Table 2 shows the data placement and the percentage of the data size. Log data were sent to disk drive #1. Customer data were sent to disk drive #2. OrderLines data were sent to disk drive #3. Stock data were sent to disk drive #4. Other data were sent to disk drive #5. Disk drive power consumption characteristics were as described in section 3. The I/O trace data used for the simulation were collected in the environment described in section 4.

<table>
<thead>
<tr>
<th>Disk Drive</th>
<th>Data</th>
<th>Capacity [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disk #1</td>
<td>Log</td>
<td>20</td>
</tr>
<tr>
<td>Disk #2</td>
<td>Customer</td>
<td>17</td>
</tr>
<tr>
<td>Disk #3</td>
<td>OrderLine</td>
<td>25</td>
</tr>
<tr>
<td>Disk #4</td>
<td>Stock</td>
<td>30</td>
</tr>
<tr>
<td>Disk #5</td>
<td>District, History, Item, NewOrders, Orders, Warehouse</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 2: Data Placement and Number of Disk Drives (Five Disks)

6.2.2 Simulation Setup

In this simulation, we calculate the disk-drive power consumption using disk-drive power consumption characteristics presented in section 3, and the I/O trace data collected in the environment presented in section 4.

We first generate I/O traces for each disk drive with and without power-saving methods. The I/O trace is gathered by each data (Log and Tables). For I/O traces without power-saving methods, we merge these traces using each disk drive. We generate the I/O trace with power-saving methods as described in subsection 6.1.2.

For calculating the power consumption of disk drives, we approximate the measured power consumption using a quadratic function. The equation of power consumption and number of I/Os to disk drive per second is
\[
 y = -0.0005x^2 + 0.1016x + 8.8025, \quad \text{where } x \text{ is the number of I/Os per second and } y \text{ is the disk-drive power consumption (VA).}
\]

The value of 8.8025 is the power consumption when the disk drive is in an idle state. We used the power consumption and I/O delay of the standby status of the disk drive. We generated the I/O trace with power-saving methods as described in subsection 6.1.2.

6.2.3 Simulation Results

Figure 16 presents simulation results of the disk-drive power consumption per second. The data are normalized using the results without the power-saving method. As Fig. 16 shows, our data placement method based on TPC–C I/O behavior characteristics and delayed write I/O reduces the disk-drive power consumption by 41.2%.

Figure 17 shows the power consumption per transaction. These data are normalized using the results without the power-saving method. The power consumption per transaction without the power-saving method with data placement method is nearly equal to the power consumption per transaction without the power-saving method. The power consumption using all the proposed power-saving methods is decreased 40%.

Figure 18 depicts transaction-throughput simulation results. The data are normalized using the results without power-saving method. As Figure 18 shows, the transaction throughput using our proposed method is almost equal to the throughput without the power-saving method. The transaction throughput is reduced only with data placement method because a transaction waits for a disk drive spin up when the disk drive is in standby status. The transaction throughput with delayed write I/O is higher than that without it because the delayed write I/O gathers I/O into a short period of time and the frequency of a waiting disk drive’s spin up is reduced.

These evaluation results demonstrate that our power-saving method saves more than 40% of disk-drive power consumption with little transaction throughput degradation despite very hard conditions of running TPC–C applications with very high transaction traffic. This result implies that our method can save more disk-drive power consumption when the I/O traffic becomes lower. Consequently, the proposed method is expected to follow dynamic workload changes of various applications at datacenters easily.
7 Conclusion and Future Works

Business applications using DBMS are the major applications that are served at datacenters. Traditional power saving approaches for these business applications is static because such power-saving methods depend on the hardware design. As described in this paper, we proposed a novel power-saving method combining application knowledge and a disk power-saving function. We then ran a TPC-C application and measured the disk-drive power consumption and the transaction throughput. Results show the potential of the dynamic power saving approach for business applications.

Our power-saving method enables dynamic reduction of disk-drive power consumption for TPC-C applications running with high throughput. The salient feature of our approach is the DBMS control of its I/O and power status of disk drives to reduce the disk drive power consumption. Our proposed method extends idle periods of disk drives using i) a data placement method based on I/O behavior of Log and Tables of database and ii) delayed writing I/O of the DBMS buffer flush.

We also demonstrated that our method achieves an approximately 40% power reduction in a five-disk-drive case. Although we focus on power saving of OLTP applications in this paper, our method is likely to be applicable to other DBMS applications such as DSS or Web Services. The I/O traffic of these applications is not so heavy compared with the I/O traffic of TPC-C. Therefore, the proposed method offers great potential to save power consumption of datacenters.

We plan to extend the proposed power-saving method to a large RAID storage system and other workloads such as TPC–H and TPC–W. We then intend to develop a storage power saving system that offers very little degradation of transaction throughput.

8 References


Optimizing Queries for Web Generated Sensor Data

Gerard Marks, Mark Roantree, Dominick Smyth
Interoperable Systems Group, School of Computer Science, Dublin City University, Glasnevin, Dublin 9, Ireland
Email: gmarks, mark, dsmyth @computing.dcu.ie

Abstract

The Web continues to offer new services and in doing so, generates large volumes of new data. The concept of the Sensor Web has led to web generated sensor data becoming available not only to the companies that own the sensors but also to end users and knowledge workers. Applications vary from climate, traffic and sports event monitoring. In one application that provides an availability service for bicycle rentals in major cities, large volumes of data are collected for data mining purposes. All transactions are recorded and stored in XML format leading to performance issues as volumes increase. In this paper, we apply a new optimization technique to boosting the performance of XML queries on Sensor Web data.

Keywords: Sensor Web, XML Queries, XML Optimization.

1 Introduction

The use of sensors in the physical-world is constantly increasing and could now be regarded as widespread. The number of applications built on top of this sensor data is also increasing. Examples are: urban traffic watch; weather monitoring; tracking of goods, etc.

Recently the city of Dublin, in keeping with many other European cities, deployed a bike sharing scheme where people can rent (and return) a bike from stations located throughout the city centre. Stations are equipped with sensors that monitor bike availability and publish such data to the DublinBikes website (www.dublinbikes.ie). Consumers can connect to the website (either through PC or mobile application) to check where stations are, how many bikes are available for rent, how many spaces are available to return bikes, and what type of payment methods are available. The data is of great interest to both consumers and providers of the service. Consumers can check where to rent or return a bike while providers can understand at which station it is better to pick up or return bikes for maintenance in order to minimize service disruption. In effect, the web service offers an efficient mechanism for determining the current status of bike or space availability.

Motivation. There are many requirements where it is necessary to access historical data or look for trends and patterns over time. For example, city planners or the companies offering the bicycle rental service must determine the optimum location for new sites; determine those sites that require expansion, or reduce or close sites that are unpopular. Furthermore, this analysis must take place over large periods of time to avoid any bias that could result from poor weather patterns or holiday season. The problem with large volumes of XML data is that queries often perform badly. What is needed is an efficient mechanism for harvesting the sensed data; updating the data warehouse, and optimizing the user queries.

Contribution. The contribution of this paper is twofold. Firstly, we provide a framework for harvesting sensor web data and automatically preparing the data for insertion into the data warehouse. To be fully automated, data must be harvested online, enriched or transformed automatically and presented for mining operations using standard query languages such as XPath and XQuery. Secondly, we must provide a superior XML processor to those currently available to address the performance issues of current approaches. Our BranchIndex was first introduced in (Marks and Roantree 2010) where we used standard benchmarking datasets to compare it against other approaches. Here, we use a real world dataset for the first time and introduce new optimizations to the index structure. We provide details of our systems performance through a series of experiments where the repository exceeds 2GB in size, with continuous harvesting from a number of bike rental sites every 60 seconds.

Paper Structure. The paper is structured as follows: in §2, we provide a more detailed description of the problem involved in extracting information from sensor repositories; in §3, we present a discussion of similar research in this area; in §4, a description of data management processors for Web Sensor data is provided; in §5, we describe how XML queries are optimized; in §6, we use an end-user query set to provide details as to how our system performs against other XML storage solutions; and finally in §7, we provide some conclusions.

2 Problem Description

The bicycle rental application records information on bike availability in cities and towns across the world. The data is collected from each location at regular intervals (see Table 1) and the dataset at the time of our experiments was 2.06 GB in size. Later in the description of our architecture, we show a typical XML document for a single station in the city of Lyon) in Fig. 2.

The following queries were supplied to us to carry out an evaluation of the bicycle rental dataset.

Funded by Enterprise Ireland Grants No. CFTD/07/201 and IR/2009/001

Copyright ©2011, Australian Computer Society, Inc. This paper appeared at the 22nd Australasian Database Conference (ADC2011), Perth, Australia. Conferences in Research and Practice in Information Technology (CRPIT), Vol. 115, Heng Tao Shen and Yanchun Zhang, Ed. Reproduction for academic, not-for profit purposes permitted provided this text is included.
Table 1: Bicycle Rental Data Collection

<table>
<thead>
<tr>
<th>City</th>
<th>Country</th>
<th>Stations</th>
<th>Data Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aix-en-Provence</td>
<td>France</td>
<td>15</td>
<td>8 KB</td>
</tr>
<tr>
<td>Amiens</td>
<td>France</td>
<td>25</td>
<td>8 KB</td>
</tr>
<tr>
<td>Besancon</td>
<td>France</td>
<td>30</td>
<td>8 KB</td>
</tr>
<tr>
<td>Lyon</td>
<td>France</td>
<td>340</td>
<td>80 KB</td>
</tr>
<tr>
<td>Mulhouse</td>
<td>France</td>
<td>35</td>
<td>12 KB</td>
</tr>
<tr>
<td>Nancy</td>
<td>France</td>
<td>25</td>
<td>8 KB</td>
</tr>
<tr>
<td>Nantes</td>
<td>France</td>
<td>89</td>
<td>24 KB</td>
</tr>
<tr>
<td>Plaine-Commune</td>
<td>France</td>
<td>44</td>
<td>12 KB</td>
</tr>
<tr>
<td>Rennes</td>
<td>France</td>
<td>18</td>
<td>8 KB</td>
</tr>
<tr>
<td>Dublin</td>
<td>Ireland</td>
<td>40</td>
<td>12 KB</td>
</tr>
<tr>
<td>Toyama</td>
<td>Japan</td>
<td>16</td>
<td>8 KB</td>
</tr>
<tr>
<td>Luxembourg</td>
<td>Luxembourg</td>
<td>46</td>
<td>12 KB</td>
</tr>
<tr>
<td>Santander</td>
<td>Spain</td>
<td>13</td>
<td>4 KB</td>
</tr>
</tbody>
</table>

Q1. Return all information recorded for each station in the city of Nantes.
/bikes//city/Nantes/stations//station

Q2. Return the number of bikes that are free across all stations in Dublin.
/bikes//city/Dublin/stations//station/free

Q3. Return all stations, from all cities, that have bicycle availability.
//city//stations[.//station//available = ‘0’]

Q4. Return all stations, from all cities, that have information on wind direction and speed; and the time it was recorded.
//city//stations[.//weather//time
[.//weather//wind//direction
[.//weather//wind//speed

Q5. Return all information regarding stations in Luxembourg in which there were no free bikes.
/Luxembourg//stations[.//station//available = ‘0’]

Q6. Return the id (identification) of each station that had no bicycle availability.
//stations//station[.//available = ‘0’]/id

Q7. Return those cities with a wind speed greater than 6 miles per hour.
/bicycles//city//stations[.//wind//speed > ‘6’]/parent::*

Q8. Return the stations that had a wind direction of 40.
//direction[. = ‘40’]/ancestor::*//station

Q9. Return all entries for Lyon for a specific date: 01/06/2010.
//Lyon//@day = ‘01’[.//@month = ‘06’[.//@year = ‘2010’]

Q10. Return the wind chill in Lyon for a specific date: 01/06/2010.
/Lyon//@day = ‘01’[.//@month = ‘06’[.//@year = ‘2010’]/chill

There is a considerable gap between the user requirements outlined by the query set and the information generated by service providers. Unlike most sensor networks, data is provided in XML format which provides high degree of interoperability and some level of semantics. However, none of the current query set can be processed without a number of processing layers. The best solution should see the provision of generic services with domain-specific or application-specific processors reduced to a minimum.

3 Related Research

As we combine the topics of harvesting sensor data and optimization of XML repositories, we separate our discussion on state of the art into two categories.

3.1 Extraction of Web Based Data

There has been considerable research into information extraction from the web. Examples include Stalker (Knoblock et al. 2003) and TextRunner (Banko et al. 2007). A comparative survey of a number of methods was conducted by Chang (Chang et al. 2006). All of these efforts focus on wrapper induction and generation for extracting information from semi-structured data. However, they do not address enrichment of data from heterogeneous sources, a necessary process for querying and data mining.

Wang (Wang et al. 2006) explores the issue of data transformation in the context of RFID data streams. While they limit their attention to semantic enrichment of data by exploiting contextual information and on complex ECA rules, we are also interested in storing historic data while harvesting data from web sources. Other researchers such as (Liu et al. 2006) and (Erwig 2003), consider the general transformation of XML data in the interest of interoperability. While this research provides generic constructs and operations to transform XML data, they do not focus on either data enrichment and mining capabilities.

In approaches to storing web sensor data, Balazinska (Balazinska et al. 2007) provides extensive information on storing sensor data from web sources, previous trend analysis and emerging trend recognition.

Efforts focusing on mining for association rules in XML documents can be found in ((Liu, Zeleznikow et al. 2006), (Wan and Dobbie 2004) and (Rusu et al. 2007)). Nayak et al. (Nayak et al. 2002) survey the data mining techniques that can be applied to the structure or content of XML documents. While these approaches all focus on discovering relationships between data in XML format, they do not deal with the task of collecting, enriching and structuring the data in preparation for high level queries.

In summary, all of the above provide differing solutions to harvesting of web data. However, none provide any mechanism for optimization. In the remainder of this section, we focus on those research efforts that offer XML optimization in a manner similar to us.

3.2 Optimization Using Partition Indexes

XML repositories differ from their relational counterparts in that they have a tree type structure and employ a set of axes to navigate across the tree structure to compute query results. Current query optimization efforts can be classified into two broad categories: index based approaches that build indexes on XML documents to provide efficient access to nodes, as found in XPath Accelerator (Grust 2002) and Xeek (Luoma 2007b); and algorithmic based approaches that are focused on designing new join algorithms, e.g. TJFast (Lu et al. 2005), StaircaseJoin (Grust et al. 2003). The first approach can use standard relational databases to deploy the index structure and thus, benefit from mature relational technology. The second depends on a modification to the underlying RDBMS kernel (Boncz et al. 2006), or a native XML database that is newly created. The XPath Accelerator (Grust 2002) demonstrated that an optimized XPath index stored inside a relational database can be used to evaluate all 13 XPath axes. However, the XPath Accelerator, and similar approaches (Grust et al. 2007), suffer from scalability issues, as this type of node evaluation (even across relatively small XML documents) is inefficient (Luoma 2007b).

A more recent solution is to partition nodes in an XML tree into disjoint subsets which can be identified more efficiently as there will always be less partitions than there are nodes. After the relevant
partitions are identified, only the nodes that comprise these partitions are evaluated using the inefficient node comparison step. Based on pre/post encoding, (Luoma 2007a) is an index based approach that requires a user defined partitioning factor to divide the pre/post plane into disjoint sub-partitions. At present, this approach lacks flexibility in determining the size of partitions without a time-consuming preprocessing stage to identify suitable partitioning factors.

Optimization of XML repositories using node partitioning will segment documents into disjoint subsets. In effect, one creates an index of partitions rather than an index of nodes. As there are fewer partitions than nodes in an XML dataset, highly efficient join operations can be performed between partitions, which reduces the workload for the more inefficient task of node comparisons.

In (Luoma 2007a), the pre/post plane is partitioned based on a user defined partitioning factor of 4. For each node, the pre/post identifier of its part is the lower bound of its x and y values respectively. The ancestors of node x can only exist in the parts that have a lower bound x value \( \leq 4 \) and a lower bound y \( \geq 4 \). They can also compute similar queries for the other major XPath axes, i.e. descendant, following and preceding. The problem with this approach is that an ideal partitioning factor is not known in advance and requires rigorous experimentation to identify. For example, in reported experiments each XML document was evaluated for the partitioning factors 1, 2, 4, ..., 256 (Luoma 2007a). In general, this type of experimentation is not feasible even for relatively small XML documents. Additionally, as XML data is irregular by nature, a single partitioning factor per dataset is less than ideal. Finally, although they suggest that the partitioning approach may be tailored to other encoding schemes such as: order/size, it relies heavily on the lower bound of each x and y value in the partitioned pre/post (or order/size) plane. Therefore, this approach does not lend itself naturally to prefix based encoding schemes such as ORDPATH (O’Neil et al. 2004), which have become very popular in recent years for reasons of updatability.

Our approach overcomes these issues by automatically partitioning nodes based on their individual layout and structural properties within each XML dataset. We do not rely on user defined partitioning factors, therefore avoiding the time-consuming processes that are required to identify suitable partitioning factors. Also, our approach is not dependent on the specific properties of XML node labels, and thus can be used in conjunction with any XML encoding scheme.

Finally, an algorithm based partitioning approach (Wong et al. 2008) uses partitioning in the pre/post plane to optimize the performance of native XML structural join operations. This approach is similar to (Luoma 2007a) as it depends on the properties of pre/post labels to avoid unnecessary node comparisons. Additionally, as (Wong et al. 2008) adopts an algorithm based approach, the partitions are dynamically calculated i.e. they are not indexed. Thus, in contrast to our approach, the calculation of these partitions leads to a performance overhead during the query process.

### 4 Data Acquisition and Management

In the next two sections, we present our architecture used to acquire and manage data. In this section, we cover: station sensors; the harvesting, enrichment, transformation processors; and the query and mining component. We then dedicate the following section to a discussion on how we optimize XML queries.

#### 4.1 Hardware Component: Sensors

Irrespective of the city or service provider, each station consists of a number of parking stands for bicycles, the actual bicycles, and a sensor based system to determine the status of the parking stands (empty or occupied). For each station this information is made available through the service provider’s web site which provides:

- station ID
- total number of bike stands
- number of bikes available
- number of free bike stands available
- number of tickets

![Figure 1: System Components and Data Management](image-url)
Timestamp information is not available through the online service so we generate our own timestamp when harvesting data.

4.2 Data Management Component

The Data Management component collects and organizes data. It is composed of four processors: Data Harvester, Enrichment and Integration and Storage.

The role of the Harvest Processor is to pull data from specified online services and as such, is a website wrapper that retrieves data every 60 seconds. Input to this component is the list of URLs where service data is located, while the output is a number of XML sensor files, one for each station. At present, the process takes between 12 and 22 seconds per site, with 13 websites in the system. A failure to retrieve data, for example when the site is unavailable, is managed using a timeout policy. The harvesting process was repeated at one minute intervals throughout the day.

The Enrichment processor is used to provide context for current station sensor data. There has been different approaches to context provision for sensor data in the past (Whitehouse et al. 2006, Devlic et al. 2008, Marks and Roantree 2009) and here, we adopt a simple system of enrichment of harvested data using available information such as station location, time and date, and weather (see XMLschema in figure 3). The time at which the information is retrieved is added as a timestamp to the station data. The time required to retrieve data from each station is also added. Finally, station data is integrated with the most recent weather conditions using various weather sites. Within the database, a schema was defined to group files into MonetDB collections based on the frequency of change of common attributes, i.e. date. The schema is displayed in Fig. 3. For a single city (in this case Dublin), this schema results in a parent collection, “Dublin”, at level one; a new collection each year at level 2; a new collection each month at level 3 and finally a new collection each day at level 4.

The Integration processor has the task of combining inputs from distributed sources. At present, stations are synchronized by time and by city.

The Storage processor places the final XML document into its appropriate place in the schema. As existing XML technology is not robust enough to manage continuous updates, we cache all data for and execute a single transaction to the MonetDB database once every 30 minutes. Part of ongoing research is to optimize our updates and reduce the time between updates. However, as new sites are added to the system, the size of each transaction is growing and the time necessary to commit updates increases accordingly.

4.3 Transformation Component

Processors in the Data Management component are generic and applicable across domains. The data transformations required by end users or specific query expressions are based upon the needs of end users and thus, the architecture uses a separate component to perform transformations. There are currently two transformations required. The first reorganizes data from the minute-by-minute view output of the collection to a station-by-station view; the second provides an aggregation of all station data on a minute-by-minute basis. The transformation process takes place once a day in order to eliminate latency within the system.

Every 24 hours, the system harvests 1,440 sensor outputs per city, each representing the state of all stations concerning the number of bicycles or parking slots at the given time stamp. For live queries, this information is sufficient to determine bike or free slot availability. However, not all of the data mining queries from our motivation section can be easily facilitated with the data in this format.

4.3.1 Transform 1 - Station Files

For each city, the system harvests from each station and viewing each dataset in chronological order, the XQuery code listed in Algorithm 1 was used to produce the station files. In Table 1, the Stations column shows the number of files generated for each city. Each station file contains a minute by minute status of one station for an entire day. While harvested data provides detailed information on the number of bikes and free spaces available at any given moment, station files only report on the status of a station, which is limited to one of 5 options. Station Status:

- Full - all bike stands occupied by a bike.
- Empty - all bike stands empty.
- Error - an error occurred during data collection from the web site.
- Bad Sum - the sum of free spaces and available bikes did not equal the total stands.
- Normal - data collected and processed as expected (default value).

In addition, the average scrape time for collection of data from the given station over the course of the day was also recorded. An example of this transformation has been represented in Algorithm 1. The result of these transformations can be seen in figure 3 and are used in a number of the queries during the evaluation phase.
Algorithm 1 Station Transform

1: for i in 1 to numStation do
2:     for $s$ in /station/station[id=i] do
3:         if $s/error)=1 then
4:             $result = 'error'
5:         else if $s/available + $s/free ≠ $s/total then
6:             $result = 'bad sum'
7:         else if $s/free = 0 then
8:             $result = 'full'
9:         else if $s/available = 0 then
10:            $result = 'empty'
11:        else
12:            $result = 'normal'
13:        end if
14:     order by time
15:     return $result
16: end for
17: end for

Figure 3: Extended Schema

4.3.2 Transform 2 - System File

Our second transformation process again takes the 1,440 sensor outputs for each city, for the 24-hour period. Operating independently of transform 1, this transform generates one output file per city that provides, on a minute by minute basis: the total number of bikes available, the number of available stands and a count of stations that reported an error across the entire system.

Figure 4: Small Segment from the Bicycle Rental Repository

5 Query Processing Component

In this section, we describe how to optimize queries through indexing XML documents. To do this, we present the key constructs used in our optimization model and then provide a step-by-step description of how we create a partition index that facilitates fast generation of result sets.

Definition 1 A branch is a set of connected node identifiers within an XML document.

A branch is the abstract data type used to describe a partition of nodes. In our work, we will deal with the local-branch and path-branch sub-types of a branch.

Definition 2 A local-branch is a branch, such that its members represent a single branching node and the nodes in its subtree. A local-branch cannot contain a member that represents a descendant of another branching node.

The local-branch uses the branching node to form each partition. Our process uses the rule that each local-branch must not contain nodes that are descendants of another branching node to create primary partitions.

Definition 3 A path-branch is a branch with a single path.

The path-branch is an abstract type with no branching node. Each member is a child member of the preceding node. Its three sub-types (orphan-path, branchlink-path and leaf-path) are used to partition the document.

Definition 4 An orphan-path is a path-branch such that its members cannot belong to a local-branch.

The orphan-path definition implies that members of the orphan-path cannot have an ancestor that is a branching node. The motivation is to ensure that each node in the XML document is now a member of some partition.

Definition 5 A branchlink-path is a path-branch that contains a link to a single descendant partition of its local-branch.

In any local-branch, there is always a single branching node and a set of non-branching nodes. With the non-branching nodes, we must identify those that share descendant relationships with other partitions. These are referred to as branchlink-path partitions and each member occupies the path linking two branching nodes (i.e. two partitions).

Definition 6 A leaf-path is a path-branch that contains a leaf node inside its local-branch.

A leaf-path differs from a branchlink-path in that it does not contain a link to descendants partitions. In other words, it contains a single leaf node and its ancestors.

Definition 7 A branch class describes the structure of a branch, from the document node to its leaf node, and includes both elements and attributes.
5.1 Indexing the Node Set

The algorithms for encoding an XML document using a pre/post encoding scheme were provided in (Grust 2002). In brief, each time a starting tag is encountered a new element object is created, which is assigned to the attributes: name, type, level, and preorder. Subsequently, the new element is pushed onto an element stack. Each time an end tag is encountered an element is popped from the element stack and is assigned a postorder identifier. Once an element has been popped from the stack, we call it the current node, and the waiting list is a set in which elements reside temporarily prior to being indexed.

When creating the first set of partitions, the goal is to ensure that all nodes are included in local-branch or path-branch partitions. The first step in the process is to determine if the current node is a branching node by checking if it has more than one child node. The next steps are as follows:

1. If the current node is non-branching and does not reside at level 1, it is placed on the waiting list.

2. If the current node is branching, it is assigned to the next local-branch in sequence. Therefore, current nodes in the waiting list become its descendants and will occupy the same local-branch as the current node.

3. If the current node is non-branching, but a node at level 1 is encountered, the current node does not have a branching node ancestor. Therefore, the current node is assigned to an orphan-path (Definition 4). For the same reason, any node currently on the waiting list is assigned to the same orphan-path.

At the end of this process, only the document node is unassigned. Fig 5 illustrates the set of local-branches LB-1 to LB-8 and orphan-paths OP-9 and OP-10.

5.2 Reducing Large Partitions

Although local-branches are rooted subtrees, they may contain nodes that do not have an ancestor or descendant relationship. A separation of nodes that do not have a hierarchical association leads to an optimized pruning effort.

Each local-branch instance has a single branching node root which may have many (non-branching node) descendants. It is the non-branching descendants of the root that are examined to determine if they share a hierarchical association. For this reason, we partition the non-branching nodes (in each local-branch) into disjoint path-branches (Definition 3). As orphan-paths and local-branches are disjoint, each of these path-branch instances will be a branchlink-path.
(Definition 5) or a leaf-path (Definition 6).

The initial partitioning method was introduced to describe how partitions are formed. In reality, we employ a refined version of this algorithm, RefinePartitions (algorithm 2), for creating BranchIndex partitions. The new branch partitions are created by processing two local-branches simultaneously. Now, current nodes up to and including the first branching node, are placed in the first waiting list (wList1) where they wait to be indexed. Subsequently, the next set of current nodes, up to and including the second branching node, are placed on the second waiting list (wList2). At this point, wList1 and wList2 contain the nodes that comprise the first and second local-branches respectively.

Algorithm 2 RefinePartitions

1. if node at level 1 encountered then
2. move nodes that comprise wList2 to orphan-path;
3. end if
4. move non-branching nodes from wList1 to leaf-path;
5. for each node n in wList2 do
6. if n = ancestor of wList1.ROOT ∧ n ≠ branching node then
7. move n to branchlink-path;
8. else if n ≠ ancestor wList1.ROOT then
9. move n to leaf-path;
10. end if
11. end for
12. move local-branch from wList1 to local-branch;
13. move local-branch from wList2 to wList1;

If a node at level 1 is encountered, the nodes that comprise wList2 are an orphan-path (line 2). If a branchlink-path (Definition 5) exists, RefinePartitions identifies it as the non-branching nodes in wList2 that are ancestors of the root node in wList1 (lines 6-7). If one or more leaf-paths (Definition 6) exist, they will be the nodes in wList2 that are not ancestors of root node in wList1 (lines 8-9). The remaining nodes that comprise the first local-branch (wList1) are then moved to the index (line 12). This will be the single branching node root of the first local-branch only.

At this point, the only node that remains in wList2 is the root node of the second local-branch. This local-branch is then moved to wList1 (line 13) and wList2 is now empty. The next local-branch is placed in wList2 and the process is repeated until no branches remain. Using this algorithm, a lot more partitions are created and this should lead to increased pruning during query processing. The refinement is illustrated in Fig 6.

The process will also track the ancestor-descendant relationships between branch partitions. This is achieved by maintaining the parent-child mappings between branches. Given two branches B1 and B2: B2 is a child of B1 if and only if the parent node of a node that comprises B2 belongs to B1. When the RefinePartitions process is complete, the ancestor-descendant relationships between branches are determined using a recursive function across these parent-child relationships.

5.3 Partition Classes

The indexing process results in a large number of branch partitions. This benefits the optimization process as it facilitates a highly aggressive pruning process and thus, reduces the inefficient stage of node comparisons. However, this type of pruning requires smaller partitions with the effect of a larger index.

The final phase in constructing the partition index is to reduce its size while maintaining the same degree of pruning. To achieve this, we use a classification process for all branches based on root to leaf structure of the partition.

Every branch instance can belong to a single branch class. A process of classifying each branch will use the structure of the branch instance and its relationship to other branch instances as the matching criteria. Additionally, in order to belong to the same class, each branch instance must have an identical set of descendant branches. The latter is required to ensure that there is no overlap between branch classes.

To explain this concept, we use figure 7 which contains three branch instances, B1-B3. Table 5 shows the extended DataGuides associated with branch classes: C1 and C2. Note that the order of the Class type associated with each branch class is important. After classification, as B1 and B3 have an identical set of descendant branch instances, they will be instances of the C1 class, while branch B2 is an instance of the C2 class.
Finally, the process that maintains parent-child relationships between branch instances (discussed earlier), must be replaced with one that maintains parent-child relationships between branch classes. The ancestor-descendant relationships are then generated for branch classes with the same approach as that used for branch instances.

5.4 Dataset Statistics

To add context to our experimental evaluation, we provide the statistics for our index after processing the 2.06GB bicycle rental repository.

- Size = 2.06GB
- Class Count = 1,067
- Node Count = 85,965,102
- NCLT Entries = 1,224
- Class Count = 5,032

The number of tuples in the CLASS relation is greater than the number of branch classes identified for that dataset: the bicycle rental dataset has 1,067 branch classes and the number of tuples in the class index is 5,032. The deliberate decision to duplicate selected branch classes allows the ancestor or descendant branch classes associated with a set of context nodes to be identified using equijoins, which have been shown to be more efficient than non-equljoins (Luoma 2007b).

The NODE relation shows the number of nodes in the dataset to be in excess of 85 million which provides a challenge for any XML query processor. The NCLT relation contains a tuple for each distinct name, class, level and type identified in the NODE relation. As the NCLT relation is used instead of the NODE relation (where possible), the number of tuples in the NCLT relation is significantly smaller than the number of tuples in the NODE relation which boosts query performance.

6 Experiments and Evaluation

In previous work (Marks and Roantree 2010), we demonstrated that the approach most similar to ours (Luoma 2007a) and a traditional node based approach to XPath (Grust et al. 2007) were not capable of scaling to the volumes of data required in this target application. Thus, in this current paper, we compare our BranchIndex to the approach described by Georgiadis at al in (Georgiadis and Vassalos 2006) (a recent DataGuide approach) - we refer to this approach as the PathIndex. In addition, we evaluate the performance of two vendor systems: MonetDB/XQuery (Boncz et al. 2006) and SQL Server 2008.

MonetDB/XQuery is a leading open-source XML database; we evaluate this approach because it was evaluated by Georgiadis at al in (Georgiadis and Vassalos 2006) (i.e. the PathIndex). SQL Server was chosen as it uses the optimization techniques described in (Pal et al. 2004), which were subsequently discussed by Grust in (Grust et al. 2007).

All experiments were run on identical servers with a 2.66GHz Intel(R) Core(TM)2 Duo CPU and 4GB of RAM. The BranchIndex and PathIndex were deployed in an Oracle 11g relational database. Oracle 11g and MonetDB/XQuery version 4.34.4 were both deployed on Fedora 12 Linux (64bit) platforms; SQL Server was deployed on a Windows 7 (64bit) platform. A small sample of one of the XML documents is shown in figure 2.

Across the vendor systems, we call the (XQuery) count() function to ensure that any overhead associated with document reconstruction (Chobotko et al. 2007) is not included in the query response times - this approach was also used in (Georgiadis and Vassalos 2007) for evaluating the comparative query response times of vendor systems. Similarly - to provide a balanced evaluation - for the BranchIndex and the PathIndex, we call the (SQL) count() function on the PRE column of the NDE relation to count the result nodes.

Finally, each XPath query was executed eleven times. In each case, the first query execution was ignored (to ensure hot cache response times) and the remaining ten queries were averaged to provide the final result in milliseconds. A timeout of ten minutes was placed on each query to allow us to perform the evaluation in a reasonable amount of time - each query that took longer than ten minutes is marked as: >10mins.

6.1 Query Classification

Experimental analysis was based on the end user queries listed earlier in §2. We classified queries into five different types to help evaluate our approach to optimization.

- QC1: Non Text Node Queries (without predicates). The query does not evaluate a text node nor does it contain predicate filters.
- QC2: Non Text Node Queries (with predicates). This query does not evaluate a text node but contains one or more predicate.
- QC3: Low Cardinality Text Node Queries. The query evaluates text node(s) that have low selectivity.
- QC4: High Cardinality Text Node Queries. The query evaluates text node(s) that have high selectivity.
- QC5: Single Step Path Fragment Queries - this type of query does not contain a primary path fragment (PPF), as defined by Georgiadis in (Georgiadis and Vassalos 2006), that spans more than one step in the XPath expression.

If a column in a relation has low cardinality (e.g. gender, genre), the number values that have identical character content will be high. Thus, queries predicated on these columns usually return a large number of tuples, i.e. they have low selectivity (Marks and Roantree 2009). In contrast, if a column in a relation has high cardinality (e.g. name, title), the number of values that have identical character content will be low; thus, queries predicated on these columns usually return a small number of tuples, i.e. they have high selectivity (Marks and Roantree 2009). Columns that have high cardinality provide the best performance for B-tree indexes.

Text nodes, in the BranchIndex and the PathIndex are stored as the attribute: value in the NDE relation (Table 2). It will become clear throughout this section, that the selectivity of these text nodes is important when describing the best and worse case queries for our BranchIndex - as text nodes were not considered during the branch classification process to keep the size of the BranchIndex small, and therefore optimized.
6.2 Results and Evaluation

The results of these queries are shown in Table 6. BranchIndex, PathIndex, MonetDB/XQuery, and SQL Server 2008 respectively.

- **Query Type QC1.** For queries Q01 and Q02, the BranchIndex and PathIndex both perform well. For query Q01, the PathIndex's system will access the PATHS relation using a regular path expression to identify all path identifiers associated with the path: /bikes/city/Wanted/stations/station. The NODE relation is subsequently accessed to determine the nodes associated with these paths and thus, form the result set. Our BranchIndex processor accesses the NCLT relation for the first step (/bikes); the pair (NCLT, CLASS) for steps two, three and four. Finally, the NODE index is accessed for step five to generate the result set. Thus, for QC1 queries, neither PathIndex nor BranchIndex systems require inefficient node comparisons as they access the NODE relation only to locate the result nodes (the final step). However, the BranchIndex performs better than the PathIndex for queries in this category as the number of tuples in the NCLT and CLASS indexes is small (see Dataset Statistics in the previous section). MonetDB and SQL Server also perform well for queries in this category. We believe this is due to MonetDB/XQuery’s Staircase Join algorithm and SQL Server’s (secondary) PATH index respectively.

- **Query Type QC2.** For these queries, the PathIndex system must access the NODE index to perform a join between each primary path fragment (PPF) (Georgiadis and Vassalos 2007). Query Q03 has two primary path fragments: //city//stations and /station/available, whereas query Q04 has four PPF’s. Thus, there is one structural join (based on individual node comparisons) required in Q03 and four in Q04 and this leads to a significant performance overhead.

In contrast, the BranchIndex still accesses the NODE relation just once to retrieve the result set - again inefficient node comparisons are not required in this approach. Therefore, the BranchIndex outperforms PathIndex (and MonetDB/XQuery) by orders of magnitude for queries in category QC2, i.e see queries Q03 and Q04 (Table 6).

- **Query Type QC3.** These queries highlight the sole weakness in our approach. The reasons are: the request for text nodes, e.g. ‘0’, requires that our system accesses the NODE index; and the low selectivity of these text nodes requires a large number of nodes to be evaluated using inefficient pre/post node comparisons. In fact, Q06 took the BranchIndex longer than ten minutes to return the result. The PathIndex also performs poorly for this category of queries as it also requires a large number of node comparisons between PPFs. MonetDB/XQuery performs best overall for category QC3 queries, which is most likely due to the Staircase Join (Grust et al. 2003) algorithm. However, it is not particularly efficient, e.g. Q07 and Q08 took more than a minute each, and Q09 took more than ten minutes. SQL Server did not answer any QC3 query inside the threshold ten minutes.

- **Query Type QC4.** Here, the high selectivity of the text nodes ensures that even though it is necessary to access the NODE index (because of the text nodes), the text nodes have high selectivity leading to fewer pre/post node comparisons for the BranchIndex. For the same reason, PathIndex performs well in this category. We do not know why MonetDB/XQuery performs poorly for these queries (i.e. Q09 and Q10). We believe this could be bug-related as we anticipated a reasonable performance from MonetDB.

- **Query Type QC5.** The PathIndex’s primary path fragments cannot optimize queries in category QC5. For example, Q08 has three primary path fragments, each of which spans just one XPath step. The PATHS relation, used by PathIndex to process multiple XPath steps simultaneously, is redundant if each PPF spans just one step each; thus, no optimization is achieved. Only our BranchIndex and MonetDB managed to return results for this category.

7 Conclusions

In this paper, we presented an architecture for harvesting web sensor data. This data was enriched and then indexed to facilitate high level XPath queries used to determine station usage patterns. This architecture was used to bridge the considerable gap between end-user requirements and the data available from web service providers. Even with enrichment of sensor data, the performance using existing XML solutions was not adequate or even possible for query processing. We also presented our system of document partition indexing which is shown to outperform other approaches for four of the five query classifications.

Our current focus is on fine-tuning our index to improve results on the poorly performing QC3 queries. We are also building extensions to our XPath query interface to incorporate data mining primitives for more complex query types.

<table>
<thead>
<tr>
<th>Query</th>
<th>Type</th>
<th>BranchIndex</th>
<th>PathIndex</th>
<th>MonetDB</th>
<th>SQLS</th>
<th>Result Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q01</td>
<td>QC1</td>
<td>1.390ms</td>
<td>3.110ms</td>
<td>5.092ms</td>
<td>258ms</td>
<td>1,411,451</td>
</tr>
<tr>
<td>Q02</td>
<td>QC1</td>
<td>1.193ms</td>
<td>1.070ms</td>
<td>3.010ms</td>
<td>166ms</td>
<td>634,320</td>
</tr>
<tr>
<td>Q03</td>
<td>QC2</td>
<td>282ms</td>
<td>&gt;10mins</td>
<td>16,766ms</td>
<td>&gt;10mins</td>
<td>191,680</td>
</tr>
<tr>
<td>Q04</td>
<td>QC3</td>
<td>282ms</td>
<td>&gt;10mins</td>
<td>1,076ms</td>
<td>&gt;10mins</td>
<td>13,046</td>
</tr>
<tr>
<td>Q05</td>
<td>QC3</td>
<td>87,385ms</td>
<td>157,090ms</td>
<td>11,890ms</td>
<td>&gt;10mins</td>
<td>13,046</td>
</tr>
<tr>
<td>Q06</td>
<td>QC3</td>
<td>&gt;10mins</td>
<td>&gt;10mins</td>
<td>108,192ms</td>
<td>&gt;10mins</td>
<td>429,585</td>
</tr>
<tr>
<td>Q07</td>
<td>QC3</td>
<td>96,143ms</td>
<td>&gt;10mins</td>
<td>107,629ms</td>
<td>&gt;10mins</td>
<td>145</td>
</tr>
<tr>
<td>Q08</td>
<td>QC3</td>
<td>302ms</td>
<td>146ms</td>
<td>16,690ms</td>
<td>&gt;10mins</td>
<td>642</td>
</tr>
<tr>
<td>Q09</td>
<td>QC3</td>
<td>164ms</td>
<td>&gt;10mins</td>
<td>105,129ms</td>
<td>error</td>
<td>6,529,626</td>
</tr>
<tr>
<td>Q10</td>
<td>QC3</td>
<td>3,379ms</td>
<td>&gt;10mins</td>
<td>106,129ms</td>
<td>error</td>
<td>6,529,626</td>
</tr>
<tr>
<td>Q08</td>
<td>QC3</td>
<td>3,379ms</td>
<td>&gt;10mins</td>
<td>106,129ms</td>
<td>error</td>
<td>6,529,626</td>
</tr>
</tbody>
</table>

Table 6: Results for the Bicycle Rental Dataset
References


User Preference Representation Based on Psychometric Models

Biyun Hu\(^1\), Zhoujun Li\(^1\), Wenhan Chao\(^1\), Xia Hu\(^2\) and Jun Wang\(^1\)
\(^1\)School of Computer Science and Engineering, Beihang University of China
XueYuan Road No.37, HaiDian District, Beijing 100191, China
\(^2\)Ira A. Fulton School of Engineering, Arizona State University, USA
Tempe, AZ 85281 USA

byhu8210@yahoo.com.cn, liizj@buaa.edu.cn, cwh2k@163.com, huxia001@gmail.com, wangjun0706149@cse.buaa.edu.cn

Abstract

Neighbourhood-based collaborative filtering is one of the most popular recommendation techniques, and has been applied successfully in various fields. User ratings are often used by neighbourhood-based collaborative filtering to compute the similarity between two users or items, but, user ratings may not always be representatives of their true preferences, resulting in unreliable similarity information and poor recommendation. To solve these problems, this paper proposes to use latent preferences for neighbourhood-based collaborative filtering instead of user ratings. Latent preferences are based on user latent interest estimated from ratings through a psychometric model. Experimental results show that latent preferences can improve the recommendation accuracy and coverage while lessening the prediction time of neighbourhood-based collaborative filtering by finding out reliable and effective neighbours; and latent preferences are better than user ratings for representing user preferences.

Keywords: User Rating, True Preference, Rating Residual, Latent Preference, Psychometric Model, Collaborative Filtering.

1 Introduction

Collaborative Filtering (CF) is a popular technique used to help recommendation system users find out the most valuable information based on their past preferences. These preferences can be explicitly obtained by recording the ratings that users have awarded on items, such as albums, movies, and books. CF algorithms can be mainly divided into three categories: model-based, neighbourhood-based and hybrid ones (Adomavicius and Tuzhilin 2005). Model-based approaches first learn a model from history dataset. The model is then used for recommending. A lot of machine learning algorithms and statistical techniques have been used to learn the model, such as probabilistic latent semantic analysis (Hofmann 2003), latent dirichlet allocation (Marlin 2003), matrix factorization (Ma, Yang, Lyu, and King 2008, Salakhutdinov and Mnih 2008), and clustering (Xue, Lin, Yang, Xi, Zeng, Yu, and Chen 2005). Although many model-based algorithms have been proposed, it seems that in real applications, neighbourhood-based algorithms are more popular used (Koren 2008), such as Amazon (Linden, Smith and York 2003) and TiVo (Ali and Van Stam 2004). These algorithms look into the similarity between users or items, and then use these relationships to make recommendations (Konstan, Miller, Maltz, Herlocker, Gordon, and Riedl 1997, Sarwar, Karypis, Konstan, and Riedl 2001, Linden, Smith and York 2003). However, user ratings may be deviated from true preferences by reasons such as wrong usage of a rating scale or type errors, resulting in unreliable similarity information and further causing poor prediction.

To overcome these drawbacks, the paper proposes to substitute latent preferences for user ratings to make recommendations. Latent preferences are computed based on user latent interest, which is estimated from user ratings through a psychometric model.

The rest of the paper is organized as follows: Section 2 provides a brief review of neighbourhood-based collaborative filtering. Section 3 analyses how user ratings may cause poor recommendation. The proposed preference representation is presented in Section 4. Experimental results are reported in Section 5 and discussed is Section 6. Finally, we conclude the paper and give future work.

2 Neighbourhood-based Collaborative Filtering

Neighbourhood-based CF algorithms can be further divided into two categories: user-based CF and item-based CF. The two often contain the following three steps:

Similarity weighting: For user-based CF, the similarity between two users is often computed based on the items co-rated by the two (co-rated items), and Pearson correlation coefficient is widely used. For item-based CF, the similarity between two items is usually evaluated based on the users who have co-rated the two (co-rate items), and adjusted cosine similarity is found best to compute the similarity (compared with cosine similarity and Pearson correlation coefficient (Sarwar, Karypis, Konstan, and Riedl 2001)).

Neighbour selection: This step requires that a number of neighbours of the active user (for user-based CF) or the target item (for item-based CF) be selected (the active user is the user whom the recommendations are for, and the target item is the unrated item for which a rating need to be predicted). These selected neighbours have the highest similarity weights. Noteworthy, not all neighbours chosen

Copyright (c)2011, Australian Computer Society, Inc. This paper appeared at the 22nd Australasian Database Conference (ADC 2011), Perth, Australia. Conferences in Research and Practice in Information Technology (CRPIT), Vol. 115. H. T. Shen and Y. Zhang, Eds. Reproduction for academic, not-for-profit purposes permitted provided this text is included.
are effective neighbours (they are the neighbours actually used in the following prediction step). For example, in Formula (1), user neighbour \( u_n \) is effective only when his/her rating for target item \( i \) is not missing.

**Prediction:** Predictions are often given as the weighted combination of neighbour ratings. For example, for user-based CF, the prediction is usually computed as Formula (1), where \( p_{i,j} \) is the predicted rating for the active user \( a \) on the target item \( i \), \( r_a \) active user \( a \)'s average rating; \( r_{i,j} \) the rating awarded to item \( i \) by active user \( a \)'s neighbour \( u_n \); \( sim(a,u_n) \) the similarity between active user \( a \) and his/her neighbour \( u_n \); and \( k \) the number of neighbours.

\[
p_{i,j} = r_a + \frac{\sum_{a=1}^{k} \text{sim}(a,u_n)(r_{i,j} - r_a)}{\sum_{a=1}^{k} \text{sim}(a,u_n)} \tag{1}
\]

3 **User Rating and Rating Residual**

User ratings are often used by neighbourhood-based CF, but those ratings may not always be representatives of user true preferences. Users may award random ratings to the items they don’t care about, they may make type errors, and they may wrongly apply the rating scale used by a system. All these and other possible disturbances may deviate user ratings from user true preferences, causing rating residual (the difference between user ratings and their true preferences).

3.1 **Assumptions**

Ratings with residual can influence the neighbourhood-based CF. For ease of analysis, the following assumptions are first made. These assumptions are validated in the experiments in subsection 6.1.

1. Assumption 1. Two users who have co-rated more items tend to be more similar.
2. Assumption 2. It is likely that two items co-rated by more users are more similar.

Based on the assumptions, the following inferences can be drawn.

1. Inference 1. Less similar users are prone to co-rate fewer items.
2. Inference 2. Probably, less similar items are co-rated by fewer users.

3.2 **Effects of Ratings with Residual**

3.2.1 **Effects on Similarity Weighting**

Ratings with residual can have two following negative effects on similarity weighting:

1. Negative effect 1. Ratings with residual can make less similar users/items become more similar.
2. Negative effect 2. Ratings with residual can make more similar users/items become less similar.

For example, as Table 1 shows, for a rating scale of 1-5, user \( u_1 \) is an ideal scorer and her rating \( r_1 \) represents her true preference \( i_1 \), while \( u_2 \) is a more severe rater whose ratings are all chosen from the wrong rating scale 1-3. If \( u_3 \) has used the rating scale 1-5 correctly, then his true preferences may be the ratings given in the fourth row of Table 1. When using true preferences, the two users are more similar (the Pearson coefficient is 0.4), but when ratings are utilized, the two users are less similar (the Pearson coefficient is 0.2), that is, because of ratings with residual, more similar users \( u_1 \) and \( u_2 \) have become less similar. Similarly, user \( u_1 \)'s true preferences are given in the last row of Table 1, but when \( u_1 \) is rating, \( u_3 \) has a rating residual of 1 or -1, then \( u_3 \)'s observed ratings are presented in the fifth row of Table 1. When using true preferences, \( u_1 \) and \( u_3 \) are less similar (the Pearson coefficient is -0.1), but when observed ratings are used, the two become more similar (the coefficient is 0.1).

3.2.2 **Effects on Neighbour Selection**

In the neighbour selection step, the two negative effects on similarity weighting work together to promote neighbourhood-based CF algorithms to use unreliable neighbours (actually less similar users/items). Considering the Inference 1 or 2, chances that these unreliable neighbours are not effective. For example: as Table 2 shows, because of ratings with residual, less similar user \( u_2 \) is selected as a neighbour of user \( u_1 \), and now user-based CF algorithm needs to predict the rating that user \( u_1 \) will award to item \( i_1 \). Because \( u_1 \) and \( u_2 \) are less similar, according to Inference 1, it is likely that \( u_2 \) hasn’t rated \( i_1 \) too, that is, \( u_3 \) is an invalid neighbour.

<table>
<thead>
<tr>
<th>( r_i / t_i )</th>
<th>( i_1 )</th>
<th>( i_2 )</th>
<th>( i_3 )</th>
<th>( i_4 )</th>
<th>( i_5 )</th>
<th>( i_6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r_1 )</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>( t_2 )</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>( r_3 )</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>( t_6 )</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

**Table 1:** User \( u \)'s rating \((r)\) and true preference \((t)\) for item \( i \).

<table>
<thead>
<tr>
<th>( u )</th>
<th>( i_1 )</th>
<th>( i_2 )</th>
<th>( i_3 )</th>
<th>( i_4 )</th>
<th>( i_5 )</th>
<th>( i_6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( u_1 )</td>
<td>null</td>
<td>*</td>
<td>null</td>
<td>*</td>
<td>null</td>
<td>*</td>
</tr>
<tr>
<td>( u_2 )</td>
<td>*</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>*</td>
<td>null</td>
</tr>
</tbody>
</table>

**Table 2:** \( u_1 \) is an unreliable neighbour of \( u_1 \), chances that \( u_2 \) is also an invalid neighbour. * is a rating, and null denotes a missing rating.

3.2.3 **Effects on Prediction**

Unreliable neighbours used would result in poor recommendation accuracy. Ineffective neighbours used would cause low recommendation coverage, and force neighbourhood-based CF algorithms to choose more neighbours for predicting, leading to increased prediction time (the computation time of Equation (1)).

In summary, the analysis above is illustrated in Figure 1.
Psychometric Models and Collaborative Filtering

Psychologists have extended the Rasch model because it can only handle binary scores (e.g. right or wrong, usually coded 1 or 0). A typical extended model, rating scale model (Andrich 1978), can be expressed as Formula (4) or (5),

\[
\log(p(r_{ui} = x)/ p(r_{ui} = x-1)) = \theta_u - b_i - \tau_x \tag{4}
\]

\[
p(r_{ui} = x) = \frac{e^{\kappa_i + x(\theta_u - b_i)}}{1 + \sum_{k=1}^{m} e^{\kappa_i + k(\theta_u - b_i)}} \tag{5}
\]

where \(x\) is a rating taken from successive rating categories set \(\{0, 1, 2, ..., m\}\); \(p(r_{ui} = x)\) is the probability of observing rating \(x\) for person \(u\) encountering item \(i\); \(\tau_x\) the ordered thresholds denoting the difficulty of being observed in rating \(x\) relative to rating \(x-1\); and \(\kappa_i = 0, \kappa_i = -\sum_{l=1}^{x} \tau_x, x = 1, 2, ..., m-1\), \(\kappa_m = 0\) the category coefficients expressed in terms of the ordered thresholds \(\tau_1, \tau_2, ..., \tau_m\).

Since proposed, these psychometric models have been applied successfully in the analysis of educational tests, attitude surveys and other rated assessments.

4.2 Psychometric Models and Collaborative Filtering

In previous work (Hu, Li, and Wang 2010), psychometric models have been used successfully to solve the sparsity problem of traditional neighbourhood-based CF algorithms. This paper differentiates from the previous work in that it focuses on presenting a better representation of user preferences, and further discusses the benefits of the representation for existing CF algorithms. While better CF algorithms are needed, the quality of user preferences is also important and needs to be researched, because only when accurate user preference information can be obtained, can CF algorithms make precise recommendations.

In recommendation system application, the parameters in Rasch model have a different meaning and reading. According to the correspondence made by Battisti, Nicolini, and Salini (2005), who apply Rasch model to measure service quality, we have made a similar correspondence as shown in Figure 2. The factor related to the students that in educational test was the ability (\(\theta_u\)) becomes now the interest. The factor related to the questions that was the difficulty (\(b_i\)), in recommendation system application becomes the agreeability. Bigger \(\theta_u\) values identify more interested users. Noteworthy, larger \(b_i\) values identify less agreeable items. Take movie recommendation as an example, intuitively, it is probable that only people who are very interested in movie will show positive response for a film with little agreeability, on the contrary, it is unlikely that people who are not interested in movie will like an agreeable film.

![Figure 1: The negative effects of ratings with residual on neighbourhood-based collaborative filtering.](image-url)
4.3 Parameter Estimation

Basic techniques for estimating these psychometric models include joint maximum likelihood, conditional maximum likelihood, marginal maximum likelihood, and Bayesian estimation with Markov chain Monte Carlo (Johnson 2007). In this paper, we have used the Winsteps Rasch measurement computer program for parameter estimation (Linacre 2007). In Winsteps, initially, the user interest $\theta_i$, item agreeability $b_j$, and threshold $\tau_x$ are all estimated to be 0, and then the PROX (normal approximation) estimation algorithm is used for the first phase of estimation. This produces revised estimates derived by Linacre (1995):

$$\hat{\theta}_i = \mu_i + \sqrt{1 + \frac{\sigma_i^2}{2.9}} \log(R_i / (N_i - R_i))$$

(6)

where $\hat{\theta}_i$ is the revised interest estimate for user $i$; $\mu_i$, the mean agreeability of the items rated by user $i$; $\sigma_i$, the standard variance of those item agreeability; $R_i$ the total rating of user $i$ (sum across all items rated by user $i$); and $N_i$, the maximum possible total rating on those same items (the maximum rating category $m \times$ the number of items rated by user $i$). Similarly, for the item agreeability parameters,

$$\hat{b}_j = \mu_j - \sqrt{1 + \frac{\sigma_j^2}{2.9}} \log(R_j / (N_j - R_j))$$

(7)

where $\hat{b}_j$ is the revised agreeability estimate for item $j$; $\mu_j$, the mean interest of the users who have rated item $j$; $\sigma_j$, the standard variance of those user interest; $R_j$ the total rating of item $j$ (sum across all users who have rated item $j$); and $N_j$ the maximum possible total rating on those same users (the maximum rating category $m \times$ the number of users who have rated item $j$). Winsteps iterates on the user ratings and updates these PROX estimates until the increment of user interest or item agreeability is small or maximum PROX iterations are reached. Initial estimates of the threshold between rating category $x$ and $x-1$ are:

$$\tau_x = \log(N_{x-1} / N_x)$$

(8)

where $N_{x-1}$ is the number of rating $x-1$ in the data. Winsteps takes the PROX estimates and uses JMLE (Joint Maximum Likelihood Estimation) for the second phase of estimation. First, the expected total ratings for users and items are computed and compared with those observed raw total ratings, and then estimates are revised. For example, if a user’s expected total rating is less than that user’s observed raw total rating, then the ability estimate raised. Concrete estimation equations for JMLE are derived by Wright and Masters (1982).

4.4 Latent Preference

Just like students’ scores are decided by their ability (but may be distorted by reasons such as raters with different severity), user preferences are decided by the latent interest of users, therefore, obtaining the user latent interest is the key to a better representation of user preferences. In this paper, we first infer user latent interest through psychometric models, and then compute user $u$’s preference for item $i$ based on user $u$’s latent interest, we name the new preference information $l_{piu}$ as latent preference, and define it as Formula (9). Compared with user ratings $r_{ju}$, latent preferences are decided by latent interest and free from rating residual, thus, latent preferences may be better to represent user preferences.

$$l_{piu} = \sum_{x=1}^{m} x[p(r_{iu} = x)]$$

(9)

5 Experiments

5.1 MovieLens Dataset

The MovieLens dataset provided by the GroupLens Research Project$^1$ is used in the experiments. The dataset contains 100,000 ratings of approximately 1,682 movies made by 943 users. Ratings are discrete values from 1 to 5 (a rating scale of 1-5). Each user has at least 20 ratings. The sparsity level of the dataset is 0.9369. As the paper (Sarwar, Karypis, Konstan, and Riedl 2001) does, 80% of the dataset was randomly selected into a training set and the remaining into a test set.

5.2 Metrics

The following two recommendation quality metrics are reported in this paper.

Mean Absolute Error (MAE). It corresponds to the average absolute deviation of predictions to the actual ratings in the test set, as shown in Equation (10), where $p_{iu}$ is the predicted rating for user $u$ on item $i$; and $r_{iu}$ the tested rating. A smaller MAE value indicates a better accuracy. MAE is one of the most often used metrics, because most research has focused on improving the accuracy of recommendations (Herlocker, Konstan, Terveen, and Riedl 2004).

$$MAE = \text{avg} | p_{iu} - r_{iu} |$$

(10)

Coverage. Recommendation coverage is less investigated than accuracy; however, it is an important metric, because systems with lower coverage may be less valuable to users (Herlocker, Konstan, Terveen, and Riedl 2004). As Equation (11) shows, the coverage is the ratio of predicted ratings to all the ratings in the test set.

$$\text{coverage} = \text{the \_number\_of\_p_{iu} / N}$$

(11)

\footnote{1 \url{http://www.grouplens.org}}
5.3 Procedures

First, user latent interest was estimated from the training set through the rating scale model (ref. subsection 4.3). Next, for each rating in the training set, corresponding latent preference was computed (ref. subsection 4.4). Then, based on the new training set formed by latent preferences, the user-based CF and item-based CF algorithms were used respectively to make predictions for the test ratings. Finally, the prediction results were compared with that obtained using the original training set composed of user ratings.

5.4 Results

5.4.1 Recommendation Accuracy and Coverage

Using latent preferences and user ratings respectively, the recommendation accuracy and coverage of the user-based CF algorithm are reported in Figure 3.a and Figure 3.b, from which we can see that, latent preferences can improve the recommendation accuracy and coverage of the user-based CF algorithm. The accuracy ascends by 23.4% when k is set 5 (MAE decreases from 0.947 to 0.725); and the coverage increases by 209% when k set 10 (coverage increases from 0.25 to 0.772).

Using latent preferences and user ratings respectively, the recommendation accuracy and coverage of the item-based CF algorithm are reported in Figure 4.a and Figure 4.b, which show that, latent preferences can improve the recommendation accuracy and coverage of the item-based CF algorithm. The accuracy increases by 31% when k is set 20 (MAE decreases from 1.064 to 0.734); and the coverage ascends by 335% when k set 15 (coverage increases from 0.184 to 0.8).

5.4.2 Neighbour Number and Prediction Time

Neighbour number and accuracy: As Figures 3.a and 4.a show, compared with using user ratings, when latent preferences are employed, the two neighbourhood-based CF algorithms can get much better recommendation accuracy with only 5 neighbours.

Neighbour number and coverage: As can be seen from Figure 3.b, using latent preferences and 5 neighbours, the recommendation coverage of the user-based CF algorithm (0.646) is even higher than that obtained using 60 neighbours and user ratings (0.639). As Figure 4.b shows, using latent preferences and 5 neighbours, the item-based CF algorithm receives significantly better recommendation coverage (0.628) than that got using 60 neighbours and user ratings (0.38).

These results above show that latent preferences enable neighbourhood-based CF algorithms to get a better recommendation quality with fewer neighbours, so latent preferences can reduce the prediction time of these algorithms.

5.4.3 The Change Trend of Accuracy

As Figure 3.a and Figure 4.a show, when using latent preferences, the recommendation accuracy of the user-based and item-based CF algorithms drops slightly with the increasing of the neighbour number k; while using user ratings, the recommendation accuracy ascends
Figures 5.a and 5.b, more reliable similarity values are computed and reported in Figure 6.a and 6.b. Those neighbours with only 5 co-rate users for item-based CF (Figures 5.a and 5.b), these items become less and less similar when user ratings are used (Figure 5.a), these users tend to be less similar. For example, for user-based CF, when $c > 15$, it is likely that two items co-rated by more users are more similar. For example, for item-based CF, when $c > 15$, it is likely that two items co-rated by more users are more similar (Assumption 2).

**Latent preferences vs. user ratings:** When using latent preferences (Figures 6.a and 6.b), the average similarity values computed by user-based CF conform to Assumption 1 exactly, and those by item-based CF almost comply with Assumption 2. For item-based CF, when $c \leq 15$ (Figure 6.b), it is likely that those unreliable similarity values arise from the data sparsity problem, because for one reason, data sparsity can also cause unreliable similarity information (Bobadilla and Serradilla 2009), and for another, the data set used in the experiments is more sparse for item-based CF than for user-based CF. There are at least 20 ratings for each user, but there is no such a restriction for each item. When using user ratings, the problems are the followings.

**Problem 1:** Less similar users/items have become more similar. For example, for user-based CF, when $x$ decreases from 55 to 5, using latent preferences (Figure 6.a), those related users become less and less similar; but when user ratings are used (Figure 5.a), these users become more and more similar.

**Problem 2:** More similar users/items have become less similar. For example, for item-based CF, when $x$ increases from 15 to 35, using latent preferences (Figure 6.b), those related items become more and more similar; but when user ratings are used (Figure 5.b), these items become less and less similar. The two problems are exactly the negative effects of ratings with residual on similarity weighting analysed in subsection 3.2.1 (Figure 1: A->B). These problems will promote neighbourhood-based CF algorithms to choose unreliable neighbours (Figure 1: B->C). For example, for item-based CF confirm the Assumptions 1 and 2 respectively. The more items two users have co-rated, the more similar the two users tend to be (Assumption 1). It is likely that two items co-rated by more users are more similar (Assumption 2).

**Discussion of Experiments**

6.1 Neighbour Reliability and Recommendation Accuracy

Recommendation accuracy is mainly related to the reliability of neighbours (whether those neighbours used are really similar); therefore, we can speculate that latent preferences enable neighbourhood-based collaborative filtering to find out more reliable neighbours. To validate this, the average similarity values for neighbours based on different numbers of co-rated items for user-based CF or co-rate users for item-based CF ($c > x$ where $x$ was set 5, 10, 15, ..., 105 respectively) are computed and reported in Figures 5.a, 5.b, 6.a, and 6.b. Those neighbours with similarity values smaller than 0 are omitted, because usually these neighbours are not used (rank behind in a neighbour list). These figures show the followings:

**Assumptions verification:** Using user ratings (Figures 5.a and 5.b), more reliable similarity values based on $c > 55$ for user-based CF and $c > 35$ for item-based CF confirm the Assumptions 1 and 2 respectively. The more items two users have co-rated, the more similar the two users tend to be (Assumption 1). It is likely that two items co-rated by more users are more similar (Assumption 2).

**Table 3:** The benefits of using latent preferences for neighbourhood-based collaborative filtering, and the influence of the neighbour number $k$.

<table>
<thead>
<tr>
<th>($c &gt; x$)</th>
<th>Accuracy Increases</th>
<th>Coverage Increases</th>
<th>Time reduced</th>
<th>$k$ increases</th>
</tr>
</thead>
<tbody>
<tr>
<td>User-based</td>
<td>23.4%</td>
<td>209%</td>
<td>Better quality with only 5 neighbours</td>
<td>Accuracy slightly decreases</td>
</tr>
<tr>
<td>Item-based</td>
<td>31%</td>
<td>335%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In summary, the experimental results reported above are summarized in Table 3.
user-based CF, using user ratings (Figure 5.a), when $x = 105$, although those related neighbours are more reliable, their average similarity value is smaller than that of those less reliable neighbours with 15 or fewer co-rated items ($c \leq 15$), so these less reliable neighbours will first be chosen from by user-based CF, causing low recommendation accuracy (Figure 1: C->D). From the analysis above, we conclude that, compared with user ratings, latent preferences can make the relationships between two users/items become more clear and reliable.

Latent preferences enable more reliable neighbours. This can also be drawn from the change trend of the recommendation accuracy. When using latent preferences, the recommendation accuracy drops slightly with the increasing of the neighbour number (Figures 3.a and 4.a). This is an intuitive result. First, as the neighbour number increases, more less similar neighbours will be used, so the recommendation accuracy drops. Second, when reliable neighbours are used, the number of neighbours will not make much difference in the recommendation accuracy. For example, user $u_k$ has two neighbours $u_{k1}$ and $u_{k2}$, and the two neighbours all like item $i_k$. No matter one neighbour or two neighbours are used, the prediction result of user-based CF for user $u_k$ on item $i_k$ is the same, that is, user $u_k$ will like item $i_k$. When using user ratings, the recommendation accuracy increases or nearly increases as the neighbour number ascends (Figures 3.a and 4.a). This is counter-intuitive. More less similar neighbours are used, but the recommendation accuracy increases. We think that this phenomenon happens because that, as the neighbour number increases, more reliable neighbours will balance the negative effects of less reliable neighbours. For example, for item-based CF (Figure 5.b), first, those less reliable neighbours with average similarity value 0.274 ($x = 5$) will be used, as the neighbour number increases, more reliable neighbours with average similarity value 0.248 ($x = 105$) will be chosen. In this process, the negative effects of those less reliable neighbours may be balanced.

From the analysis above, we conclude that: latent preferences are better representatives of user preferences than user ratings, and they enable neighbourhood-based collaborative filtering algorithms to find out more reliable neighbours, thus can improve the recommendation accuracy of these algorithms.

### 6.2 Neighbour Effectiveness and Coverage and Prediction Time

In previous subsection, it is concluded that using latent preferences, more reliable neighbours can be found. Chances that these more reliable neighbours are more effective, thus they can improve the recommendation coverage and lessen the prediction time (it can be analysed similarly as done in subsections 3.2.2 and 3.2.3). For further validation, the average effective ratios of those neighbours for predicting all the test ratings are computed and reported in Table 4. For predicting a specific test rating, the effective ratio is computed as: \((\text{the number of effective neighbours} / k)\). The results change only a little when we vary the neighbour number $k$ from 5 to 30, so only the results with $k$ set 5 are reported. As can be seen from Table 4, the average effective ratios of more reliable neighbours found by using latent preferences are much higher than those of less reliable neighbours found by using user ratings. When using latent preferences, for the first 5 neighbours used, averagely, one neighbour is effective for user-based or item-based CF algorithm; but when user ratings are used, the first 5 neighbours are nearly invalid (this shows the negative effects ratings with residual on neighbour selection (Figure 1: C->E), and this will further cause low recommendation coverage and long prediction time (Figure 1: A->B->C->E->F and A->B->C->E->G->H)).

<table>
<thead>
<tr>
<th>Latent preferences</th>
<th>User ratings</th>
</tr>
</thead>
<tbody>
<tr>
<td>User-based CF</td>
<td>0.25</td>
</tr>
<tr>
<td>Item-based CF</td>
<td>0.24</td>
</tr>
<tr>
<td>User-based CF</td>
<td>0.04</td>
</tr>
<tr>
<td>Item-based CF</td>
<td>0.02</td>
</tr>
</tbody>
</table>

Table 4: The average effective ratios of neighbours found by using latent preferences and user ratings when the neighbour number $k$ is set 5.

From the analysis above, we conclude that: by finding out more effective neighbours, latent preferences can improve the recommendation coverage of neighbourhood-based collaborative filtering algorithms. This can further reduce the prediction time of these algorithms because fewer neighbours are needed.

### 7 Conclusions

The contributions of the paper include the followings. First, a theoretical analysis of the negative effects of using user ratings on the neighbourhood-based collaborative filtering is presented. Second, a new preference representation method, latent preference, is proposed. Third, experimental results have shown that latent preferences can improve the recommendation accuracy and coverage while lessening the prediction time of neighbourhood-based collaborative filtering algorithms by finding out reliable and effective neighbours. Fourth, experimental results have manifested the negative effects of using user ratings presented in the theoretical analysis. In conclusion, theoretical and experimental analysis has shown that latent preferences are better representatives of user preferences than user ratings.

In future work, we will further investigate whether latent preferences can improve the recommendation quality of model-based collaborative filtering algorithms.

### Acknowledgements

Authors acknowledge Shyong Lam and Jon Herlocker for cleaning up and generating the MovieLens dataset, and John Michael Linacre for providing Ministep student version of Winsteps.

### 8 References


Filtering Architecture. Proc. 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, WA, USA, 394–401, ACM Press.


A System for Managing Data Provenance in In Silico Experiments

Jarrod Trevathan, Ian M. Atkinson, Wayne W. Read, Nigel Sim and Chris Christensen

eResearch Centre, 
James Cook University, 
101 Angus Smith Drive, Douglas, Townsville, Queensland, Australia, 4811 
Email: eResearch@jcu.edu.au

Abstract

In silico experiments use computers or computer simulation to speed up the rate at which scientific discoveries are made. However, the voluminous amounts of data generated in such experiments is often recorded in an ad hoc manner without regard to workflow, and often lacks rigorous business rules. The absence of stringent auditing and reporting policies makes it difficult to repeat experiments and largely denies independent parties the ability to verify study results. This paper presents a data provenance management system based on the utility of the ICAT metadata storage service as a viable schema for representing in silico experiments. The system provides a portal interface to integrate ICAT with job execution. We have built on a data repository which can handle arbitrary data size, complexity and type. This can be practically used to compare, validate and aid in the repetition of historic experiments. Furthermore, data can be verified via external repositories/sources which will ultimately enhance the scientific merit of in silico experimentation. Our proposed system augments existing applications and therefore does not require users to modify their current experimentation platform. A test case for a pharmacological study is presented to illustrate the proposed system’s versatility for reporting and auditing of experiments and their results.

Keywords: Provenance, in silico, experiment management, workflow, version control

1 Introduction

Execution of computational, or in silico experiments follows the same methodical processes as any other scientific pursuit. Therefore, clear, accurate and detailed record keeping is equally important in a virtual environment as it would be in a physical laboratory. Since the information about an in silico experiment setup is already computerised, it is a much simpler task to capture this in a standardised way than in a comparable physical laboratory. Various disciplines specify standards and procedures for the collection and storage formats of scientific metadata. However, there are numerous inconsistencies between disciplines.

As Grid (Foster and Carl 1999, Fran et al. 2003) and Cloud (Armbrust et al. 2009, Buyya et al. 2008) computing systems become more widely adopted to perform in silico experiments, it is increasingly important to acknowledge that their heterogeneous nature poses a risk to experiment reliability and repeatability. Systems will vary in terms of both hardware and software. This makes it difficult to ascertain whether the outcome of an experiment will be the same when repeated on different platforms.

Extensive documentation regarding the main attributes of the system environment can help to overcome this by providing appropriate insight which can explain unusual trends in the data.

Collecting static machine information regarding hardware and software, sets up reference metrics which can be used to group and loosely compare execution runs on different machines. The same workflow run on a machine with identical memory and CPU should be very similar. Significant discrepancies would indicate there are issues relating to the hardware, software, or the application being executed. Then software dependencies could be investigated, ideally leading to the offending library, or methods being identified and rectified.

When analysing a dataset using computational techniques, it is easy and sometimes commodious to assume that the dataset simply came into existence. This is typically done without questioning the true origin, quality, or how/why it came to be in its intermediate and final representations. Simple provenance information is essential when trying to audit, review or reproduce findings (Buneman et al. 2000, Simmhan et al. 2005).

Existing data provenance management systems (DPMS) are built into some eScience tools. For example, experimentation environments such as Chimera (Foster et al. 2002), myGrid (Stevens et al. 2003, Yu and Buyya 2005) and Kepler (Altintas et al. 2006, Ludscher et al. 2005) provide various mechanisms for automatically capturing data provenance for experiments conducted using their frameworks. These environments use their own proprietary systems for storing the provenance information in either an XML, RDF or relational form. Such systems are satisfactory when the target audience:

1. Is largely consuming existing services; and
2. Is able to migrate to a new system; and
3. Does not desire to participate in the development of in silico simulations.

However, such an application-specific nature leads to a heterogeneous environment where each system does not interface conveniently with other external tools and systems. It is also undesirable to re-tool the practitioner with an entire new framework in an effort to improve their book keeping practices.

While the collection methods for provenance information will vary between implementations, it does...
not need to occur within the main application. We propose that a pre and post process can be used to collect this information without the need for modification to the running application. This data is written to the document store, and returned along with the results of the main application. Therefore, a provenance management system must enable the current users of in silico experimentation to capture their provenance information without amending their existing systems. The provenance data can then be stored in an arbitrary data repository where it can be verified by independent third parties and used to reproduce experiments if desired.

The physical science community utilises a number of Laboratory Information Management Systems (LIMS) which track physical experimentation. One such system from the crystallography community is the ICAT schema (Flannery et al. 2009). ICAT is a relational database implementation of the Council for the Central Laboratory of the Research Councils (CCLRC) Scientific Metadata Model. There are a number of Service-Oriented Architecture (SOA) implementations which present this schema to other applications. The generic and interoperable nature of ICAT makes it ideal as a repository for data storing applications which present this schema to other systems for Grid computing.

metadata in eScience. Yu and Buyya (Yu and Buyya 2007) discuss a number of models and uses of provenance management systems. The provenance data can then be stored in a general-purpose repository where it can be verified by independent third parties and used to reproduce experiments if desired.

This paper presents a DPMS based on the utility of the ICAT metadata storage service as a viable schema for representing in silico experiments. The system provides a portal interface to integrate ICAT with job execution. We have built on a data repository which can handle arbitrary data size, complexity and type. This can be practically used to compare, validate and aid in the repetition of historic experiments. Furthermore, data can be verified via external repositories/sources which will ultimately enhance the scientific merit of in silico experimentation. Our proposed system augments existing applications and therefore does not require users to modify their current experimentation platform. A test case for a pharmacological study is presented to illustrate the proposed system’s versatility for reporting and auditing of experiments and their results.

This paper is organised as follows: Section 2 discusses related work on grid and cloud computing data management systems. Section 3 describes the architectural framework within which the system operates. Section 4 presents the design methodology for constructing our system. Section 5 describes the implementation of the proposed system in a case study and discussion for the proposed system. Section 7 provides some concluding remarks and avenues for future work.

2 Related Work

The Computer Science and Informatics communities currently do not adopt a uniform experiment management regime. However, there exist opportunities and resources to facilitate data provenance management and other flow-on benefits to these research communities. This section presents related work that has been conducted on creating in silico workflow and data provenance management systems.

Bunemann et al (Buneman et al. 2000) outline some of the major issues with capturing data provenance, while Simmhan et al (Simmhan et al. 2005) discuss a number of models and uses of provenance metadata in eScience. Yu and Buyya (Yu and Buyya 2005) present a taxonomy of workflow management systems for Grid computing.

Chimera (Foster et al. 2002) is a virtual data system, which combines a virtual data catalogue for representing data derivation procedures and derived data, with a virtual data language interpreter. The interpreter translates users’ requests into data definition and query operations on the database. Chimera is coupled with distributed “Data Grid” services to enable on-demand execution of computation schedules constructed from database queries. The system has been tested on two problems. The first involved the reconstruction of simulated collision event data from a high-energy physics experiment. The second was the search of digital sky survey data for galactic clusters.

A novel use of provenance in Chimera is to plan and estimate the cost of regenerating datasets. When a dataset has been previously created and it needs to be regenerated (e.g., to create a new replica), its provenance guides the workflow planner in selecting an optimal plan for resource allocation.

myGrid (Stevens et al. 2003) provides middleware in support of in silico (computational laboratory) experiments in biology, modelled as workflows in a Grid environment. myGrid services include resource discovery, workflow enactment, and metadata and provenance management, which enable integration and present a semantically enhanced information model for bio-informatics. myGrid is service-oriented and executes workflows written in XSBuff language using the Taverna engine (Oinn et al. 2004). A provenance log of the workflow enactment contains the services invoked, their parameters, the start and end times, the data products used and derived, and ontology descriptions, and it is automatically recorded when the workflow executes. This process-oriented workflow derivation log is inverted to infer the provenance for the intermediate and final data products. Users need to annotate workflows and services with semantic descriptions to enable this inference and have the semantic metadata carried over to the data products.

Kepler (Ludscher et al. 2005) is an open-source scientific workflow system which enables scientists to design scientific workflows and execute them efficiently. Kepler provides emerging Grid-based approaches for access to distributed resources such as data and computational services, while hiding the underlying complexity of the Grid technologies. The Kepler system supports the automation of low-level data processing tasks so the focus can remain on the scientific questions of interest. The workflows that Kepler produces can be implemented in cross platform environments, provide documentation and visualisation of the processes and bring scientists to the desktop. Each workflow step is represented by actors, which are individual processing components that can be manipulated through a drag and drop method into a workflow, via Kepler’s visual interface. The actors are then connected and organised according to the data flow, and the dependencies among them, to form the workflow.

Altintas et al (Altintas et al. 2006) present a framework for data and process provenance in Kepler. They outline the requirements and issues related to data and workflow provenance in a multi-disciplinary workflow and show how generic provenance capture can be facilitated by Kepler’s actor-oriented workflow environment. They also describe the usage of the stored provenance information for efficient rerun of scientific workflows.

Other DPMSs include the CMCS project (Myers et al. 2003, Pancarella et al. 2003) for collaboration and metadata-based management for multi-scale science, the Earth System Science Workbench (ESSW) (Frew and Bose 2001) for metadata management and data storage for Earth Science applications, and Trio (Cui and Widom 2000, Widom 2005)
for tracing lineage information in data warehouses.

3 ARCHER

This section describes the ARCHER project and presents architecture within which the proposed DPMS operates.

The *Australian ResearCH Enabling environment* (ARCHER) project (Androulakis et al. 2009) is an Australian higher education initiative which has developed ‘production-ready’ software tools, operating in a secure environment, to assist researchers to:

- Collect, capture and retain large data sets from a range of different sources including scientific instruments;
- Deposit data files and data sets to eResearch storage repositories;
- Populate these eResearch data repositories with associated metadata;
- Permit data set annotation and discussion in a collaborative environment; and
- Support next-generation methods for research publication, dissemination and access.

3.1 Applications

Figure 1 illustrates the main software components and workflow of the ARCHER platform. Within this framework, ARCHER offers the following services:

* ARCHER Enhanced Plone - a collaborative workspace development tool for building websites where researchers can come together.
* HERMES - for managing research datasets from a desktop client application.

Figure 1: The ARCHER research data management Workflow

3.2 Data Services Layer

Figure 2 illustrates the logical layers within the ARCHER system. At the top is the Application Layer that supports the aforementioned applications. Below this is the Data Services Layer. The *ARCHER Data Services* (ADS) infrastructure supports other components in the ARCHER Toolset, and incorporates both third-party and ARCHER-developed software.

* HYDRANT - for managing workflow automations from a web application.
* DIMSIM - Distributed Integrated Multi-Sensor and Instrument Middleware for concurrent data capture and analysis.
* CIMA - Common Instrument Middleware Architecture.
* SAL - A Sensor Abstraction Layer to automate sensor network hardware configuration and simplify sensor instrument access and control (Trevathan et al. 2010).

Figure 2: Layers in ARCHER
Figure 3: The SRB middleware (figure taken from (Baru et al. 1998))

It provides data storage via the Storage Resource Broker (SRB) (Baru et al. 1998), metadata storage with ICAT, and authentication through MyProxy. ADS is made up of two layers:

- ADS Service Layer provides web service interfaces to the data and metadata storage.
- ADS Infrastructure Layer provides distributed storage, authentication, and an optional certificate authority.

**Infrastructure Layer**

The ADS Infrastructure Layer is a streamlined package of third party grid components such as Globus 2, MyProxy and SRB with customisations to allow authentication to an Lightweight Directory Access Protocol (LDAP) server. The components are all installed through automated deployment scripts with minimal configuration required.

SRB provides a uniform interface to heterogeneous data storage resources over a network (see Figure 3). It is a logical distributed file system based on a client-server architecture which presents users with a single global logical namespace or file hierarchy. As part of this, it implements a logical namespace (distinct from physical file names) and maintains metadata on data-objects (files), users, groups, resources, collections, and other items in a Metadata Catalog (MCAT). System and user-defined metadata can be queried to locate files based on attributes as well as by name.

SRB is middleware in the sense that it is built on top of other major software packages (various storage systems, real-time data sources, a relational database management system, etc.). It has callable library functions that can be utilized by higher level software. However, it is more complete than many middleware software systems as it implements a comprehensive distributed data management environment, including various end-user client applications. It has features to support the management and collaborative (and controlled) sharing, publication, replication, transfer, and preservation of distributed data collections.

SRB is a commercial product, free for use by academic institutions, and with full source code available. It is sometimes used in conjunction with computational grid computing systems, such as Globus Alliance, and can utilize the Globus Alliance Grid Security Infrastructure authentication system. SRB can store and retrieve data in archival storage systems such as HPSS and SAM-FS, on disk file systems (Unix, Linux, or Windows), as Binary Large Objects or tabular data in relational database management systems, and on tape libraries.

The Virtual Data Toolkit (VDT) is a grid software packaging system that installs several grid components, including MyProxy and the Globus libraries.

MyProxy 4 is a service that listens on port 7512, generating short-lived certificates called “proxies” for users upon request. It serves these over the network, allowing users access to remote services. ARCHER adds LDAP authentication to the standard MyProxy installation.

Many parts of ARCHER Toolset require a common Certificate Authority (CA) to sign certificates. ARCHER Data Services can set up MyProxy to function as a CA. This is useful in a testing or development environment.

**Service Layer**

The ADS Service Layer is composed of two web applications – ICAT and MCAText.

ICAT is a metadata storage service that implements the CCLRC Scientific Metadata Model version 2 to record information about scientific experiments. The data from the experiments itself is stored on the SRB, while the metadata is held in the ICAT. ICAT’s storage is implemented as a PostgreSQL database, which is installed through the Archer XDMS application.

MCAText is an ARCHER-developed web service layer over SRB and its MCAT database. It provides a high performance mechanism for other services to lookup authorisation information on content within SRB. It provides update notification to other services when content is modified, moved, or created. It is used by certain ARCHER tools, including the ICAT service and ARCHER Collaborative Workspace.

Figure 4 illustrates how all of the internal components interact in the standard ADS configuration.

### 3.3 Bringing it all Together

The main ARCHER project was completed in 2008. As a result of the work performed in ARCHER, researchers in general are now much closer to: having a place to collect, store and manage experimental data; deploying software tools focused on management of data and information; being able to easily customise a collaborative and adaptable portal web site relevant to their research field; having standardised and secure methods of storing, accessing, and analysing research results; and finding it easier to collaborate and share research datasets and information.

ARCHER has addressed many key issues in e-Research data management. It is enabling researchers to keep better track of their scientific data by organising them into intuitive and generic structures described by the CCLRC Scientific Metadata Model, and by making the research repository easily searchable. It is alleviating issues around the collaboration of large datasets by: supporting the storage of large research datasets, adopting a data-centric view; and by providing an initial set of collaborative tools that can be directly associated with the research data. It is also protecting the confidentiality and security of research data by limiting the access to a project team’s research data.

\(^*\)http://vdt.cs.wisc.edu/
\(^*\)http://www.my-proxy.com/
4 Design Methodology

This section presents the design methodology for constructing the proposed system.

4.1 The ICAT Schema

The proposed DPMS is based on the continued work from the ARCHER implementation of the ICAT service. This implementation provides a web services interface with the schema, and includes tight integration with the SRB as a file repository.

ICAT is primarily concerned with result sets from physical, automated experimentation. Figure 5 illustrates the ICAT schema. Represented within the schema are studies, which form the highest grouping of research activities. Studies contain many investigations which are the individual experiments. Investigations involve zero or more samples, which are the focus of the investigation, and produce datasets and data files.

However, ICAT can also represent in silico experiments, by defining a dataset type as an input dataset and not defining any samples. If we consider an investigation to be any process which produces data output from samples or other datasets, then we can model any transformation as a sample. This makes sense especially if we want to capture failed experiments, such as descriptor calculations which fail due to invalid or out of bounds structures.

One of the key extensions provided by the ARCHER-ICAT services is the ability to store auxiliary documents against ICAT elements. It allows listing, insertion, updating and deletion of these documents, given the ICAT element type and element ID as the key. We use this feature to store an XML document for each of the in silico investigation types discussed here.

4.2 User Source Code Integrity

Principle among the requirements for effective data lineage and provenance collection for in silico experiments, is capturing the exact workflow or scripting used to transform or process the data. This is particularly evident when analysis is performed using custom software, or scripted environments such as MATLAB\(^5\) or R\(^6\). The benefit which the user can leverage from these situations, which cannot necessarily be captured in a single set of configuration parameters, requires a more powerful approach.

Being able to identify the exact code used to perform a specific computation is essential. Given the ease with which small changes can be made to text-based scripts means robust and integrated methods of code validation are required.

Source code version control systems used in software development offer a capability to store, address and recall historic versions of a code base. Utilising this functionality within the provenance framework gives an opportunity to address the temporal software validation issue. However, the following constraints must be applied:

- The version control repository is maintained alongside the main provenance store. This ensures the same level of security from tampering, and also disaster tolerance.
- The code is always checked out from the repository by the execution system, and is not modified before execution.

\(^5\)http://www.mathworks.com/products/matlab/
\(^6\)http://www.r-project.org/
In the proposed system we consider the centralised version control application – Subversion\(^7\). Subversion stores text and binary files in a hierarchical directory tree. When a file is changed, added or deleted, it can be checked back into the central repository, creating a new revision. The user, or another process, can then check these files out at another location and they will be identical to the version used at the original file location. When changes are made, the user is able to view the difference between the working copy and those in the central repository, containing all previous revisions stored in that repository.

Provided the code is in a version control system, then the exact version of the software can be retrieved to process the dataset. The software revision is stored in a network accessible format within the provenance system as follows:

\[
<protocol>://<host>[:<port>]/<path>@<revision>
\]

The \(\langle protocol\rangle\) will typically be \url{http} or \url{https}, and \(\langle host\rangle\) is the network name of the subversion server. The \(\langle port\rangle\) will be supplied if it is non-standard, \(\langle path\rangle\) is the location of the code, and \(\langle revision\rangle\) is what version of the code will be used for this experiment.

4.3 Reporting and Auditing

By collecting provenance information in the aforementioned manner, it becomes possible to produce new tools to assist in the monitoring and supervision of experiment-based research. Through providing a query interface, the ICAT service can enable supervisors to remotely, and continuously monitor students and subordinates. This helps to allow for rapid rectification of experimental errors.

Flagging of other interesting artefacts within results also becomes possible. The uniform storage of these investigations means such annotations can be extracted reliably, and without risking the loss of possibly interesting and insightful avenues for future investigation.

5 Implementation

This section describes how to implement the proposed DPMS using the ICAT schema, a source code version control system, and the related components of the ARCHER project.

5.1 Data Format

Within the ICAT schema, investigations and datasets both have a type attribute which is used to distinguish the specific function of the element. The available types are not stipulated by the ICAT specification, which allows the schema to be used for many purposes. The following investigation types are defined for the in silico use cases:

- **Retrieval** – This is either downloaded from a website, or queried from a database;
- **Calculation** – Includes the running of format conversion and the calculation of molecular descriptors; and
- **Analysis** – The running of a workflow or application script which processes the data.

The following dataset types are defined to complement the investigation types:

- **Input Dataset** – These are the source datasets for the defined Calculation and Analysis investigation types. The data files may or may not be populated, depending on whether this is the primary registration of the dataset, or if it is the culmination of data from other datasets.
- **Output Dataset** – These are the files produced by Calculation or Analysis. Data files need to be populated to describe what is in each file.

By using the dataset type and the investigation type, a complete history of the data back to the original collection can be constructed.

Two supplementary documents are also stored against the investigation which details the operation. These are used to fully capture the metadata about the configuration of the investigations. These documents are stored as XML, and the schema is specific to the investigation type.

The first document is referred to as the Retrieval document. The purpose of the retrieval operation is to capture the source information about the dataset. This allows verification to occur at a later date. Furthermore, it can be cited correctly when used in investigations. The format of the retrieval document is as follows:

**Retrieval**:

- URI
- Retrieval time
- License
- Dublin Core metadata record \(^8\)

The second document is referred to as the Analysis and Calculation document. Analyses are computer applications which are run using the data. The metadata associated with them needs to be collected at the point of execution. This data includes the hardware and software environments which affect the execution of the analysis. Furthermore, application parameters also need to be captured so that the analysis can be repeated by an independent third party. As some applications require configuration files, these can be stored in the document, as can references to other input files. Input file validity is confirmed by also storing a SHA-256 \(^9\) checksum of the files. The format of the Analysis and Calculation document is as follows:

**Analysis and Calculation**:

- **Applications**
  - Name
  - Version
  - Operating System
  - Runtime
  - Command - command line
    * Library Path (name =)
      - Library name
    * Input Parameters
      - Name
      - Value

\(^7\)http://subversion.tigris.org/

\(^8\)http://dublincore.org/

\(^9\)Secure Hash Algorithm (SHA) is a family of hash functions used for verifying the integrity of a document. SHA-256 uses 32-bit words.
When the output files are available, they are moved off their initial storage into the data repository component of the system. This is achieved using a transfer agent which is started on the head node of the execution cluster. The agent is initialised with the details of the ICAT server, destination dataset, and the user’s X.509 certificate. It will copy the files into the data store and computes the checksum as this is happening. Upon completion, the details are written directly to the ICAT service against the dataset specified.

5.2 The ICAT Service and Security

The ICAT service exposes the ICAT schema via web services, providing basic creation, read, update, and delete operations for each of the elements. It also allows arbitrary XML documents to be stored against specific ICAT elements. The framework is presented in a manner that is easily extensible for adding new operations to the service, and altering some aspects of the existing service.

The ICAT-SRB services is an extension to the vanilla ICAT service, integrating file data storage using SRB. This implements a security model which is based around the object permissions within SRB. This is achieved by mirroring the core ICAT structure as folders and files within SRB, and then explicitly linking the ICAT records to these SRB objects using the location attribute on the given ICAT object. In the case of ICAT objects which have no direct mirror within the SRB hierarchy, the permissions are inherited from the closest parent which is mirrored. Lookup table objects are access controlled via a separate access control list in the service, as they are always world readable, and administered centrally.

Figure 6 illustrates the interactions between ICAT, SRB and MCAText. Achieving this integration requires both SRB and the ICAT services to be using X.509 authentication, so that the X.500 DN (Distinguished Name) of the authenticated user is used to match the user’s permissions via a call from the ICAT service to an SRB support service.

MCAText provides two key functions to authorised services:

- Read only lookups of SRB permission information;
- Notification of file and folder changes within SRB.

This provides integration of the permissions systems, and also allows ICAT to be kept consistent with object changes within SRB.

5.3 Application Portal

The principle objective of this project was to capture more of the computational experiment information from the users during experimentation. To achieve

---

10 X.509 is a cryptographic standard for a public key infrastructure for single sign-on and Privilege Management Infrastructure. X.509 specifies standard formats for public key certificates, certificate revocation lists, attribute certificates, and a certification path validation algorithm.
this, the cluster job submission portal was augmented with ICAT experiment metadata facilities. As job submission is handled by the portal, it is practical to automatically capture execution environment information directly from within the submission scripts. Initially the user is required to provide some information about the job they are submitting including:

- Source of the data (ICAT, web, etc.);
- Code version (obtained from Subversion);
- Experiment description;
- Which study this is a part of; and
- The names of the relevant output files.

Using the information provided, the portal can automatically create a new investigation within the appropriate study. Upon completion of the job, the output data is stamped with a checksum and injected into the institutional repository. This provides a permanent record of the particular experiment.

In some cases there is significant experiment parameter tuning required before any meaningful output is produced (for instance when fixing bugs). In this case it is undesirable to capture every experiment automatically, rather after the fact the practitioner may choose to capture an experiment. This is done by providing the same information as previously, except this investigation will be given a flag to indicate that it was not an automatic capture.

Furthermore, after an investigation has been captured, the practitioner may wish to annotate this run with notes about motivation for certain design or parameter decisions.

5.4 Command Line Assistant

A command line assistant is provided to assist with jobs run in batch mode, such as cluster environments. This can be used to call the actual application, and will produce an XML document in the analysis schema. This will need to be completed by the user. All other information is supplied automatically by the system, such as environment, input files and parameters. The resulting document is then submitted to the ICAT service along with the output details of the execution.

6 Case Study and Discussion

This section outlines a case study using the proposed DPMS and discusses its effectiveness in managing in silico provenance information.

6.1 Task

Consider the life cycle of a typical data mining exercise. It involves the following steps:

- Data collection or retrieval
- Data conversion and filtering
- Processing

Any of these steps may be performed many times, leading to a tree of descendant investigations originating from the initial dataset. For an institution which has many concurrent and historic lines of investigation, this information is critical when revisiting previous work, or for discovering colleagues who are deriving work from common internal, public or reference datasets.

6.2 Workflow

The study proceeds as outlined in the following workflow (see Figure 7):

1. Download reference data set
2. Register dataset with ICAT
3. Calculate descriptors
4. Store transform with ICAT
5. Perform analysis
6. Store results in ICAT
7. Repeat steps 5 and 6 refining the model parameters

In steps 1 and 2, the outcome is to retrieve this dataset from its public repository and place it in ICAT as a special “retrieval” investigation. This involves uploading the dataset into data storage area (i.e., SRB), which provides a local copy of the data and ensures that subsequent investigations are using the same dataset. This offers protection against changes to datasets in external repositories (both public and private), going unnoticed and corrupting investigations.

Next, steps 3 and 4 transform these input data models into a set of molecular descriptors which can be used within the data mining methods. This transformation is done using the E-DRAGON (Tetko et al. 2005) web portal. It produces a number of output files, which are processed into a single dataset file.

\[\text{E-DRAGON is the electronic remote version of the well known software DRAGON, which is an application for the calculation of molecular descriptors developed by the Q SAR Research Group. These descriptors can be used to evaluate molecular structure-activity or structure-property relationships, as well as for similarity analysis and high throughput screening of molecule databases.}\]
This step is registered in ICAT as a transformation investigation on the initial download investigation, having the input dataset being the output dataset from the previous step.

Once the data preparation stage is complete, the actual investigation can proceed. Steps 5 and 6 of the workflow are particularly exploratory, requiring a number of independent investigations to explore different approaches for generating a robust model. This includes executing three different types of model, and using different pre-processing steps. Each of these data mining processes are stored in ICAT.

Conclusions as to the appropriate type of model and pre-processing to use to obtain satisfactory results cannot be drawn from the results of the data mining steps which have just been completed. However, questions still remain as to whether this model is general enough to be run against another HIA dataset and produce reliable results. In this case there is alternative HIA investigation stored in the ICAT, that uses a unique dataset. A new investigation is started, reusing the model produced in the best performing investigation from the previous phase, and using the dataset from the alternative HIA study. This is illustrated as steps I, ii and iii in Figure 7.

6.3 Post Experiment Analysis

Should the performance of the model not be as high as desired, then the practitioner could check the other investigations run against the alternative HIA dataset. This may reveal that the dataset was found to be unreliable, or otherwise not representative of HIA datasets. Being able to visualise these relationships between investigations can be useful.

Figure 8 illustrates the longer term value of recording experimental provenance metadata with a system like ICAT. The schema and its implementation allows for the straightforward retrieval and comparison of results. ICAT also serves as an experiment activity record keeping system which can be utilised as an audit trail of time stamped records. This can be used in a variety of ways including fulfilling supervision and intellectual property requirements.

7 Conclusions

There are numerous inconsistencies with capturing data provenance in the Computer and Information sciences fields. This paper presented a DPMS based on the utility of the ICAT metadata storage service as a viable schema for representing in silico experiments. The system provides a portal interface to integrate ICAT with job execution. We have added a data repository which can handle arbitrary data size, complexity and type. When used in conjunction with a source code versioning system such as Subversion, end users and other parties can determine which version of the code was executed for the experimentation, and whether there have been any modifications to the code since the experimentation occurred.

The proposed system can be practically used to compare, validate and aid in the repetition of historic experiments. It allows for verification of data from external repositories/sources through the use of SRB. SRB is powerful in that it brings together data from multiple disparate sources over a distributed environment. This provides superior facilities for reporting/auditing of experiments and their results.

ICAT provides a uniform format for instrument, sensor and computation data. It also establishes a unified security procedure for verifying and obtaining datasets. This is based on the underlying security provided by SRB.

While the proposed system is based on extensions to ICAT proposed by the ARCHER project (i.e., MCAText), it works independently of the ARCHER system. Users are free to use any existing platform they desire to conduct experiments. The proposed system augments the existing platform with pre and post processes to manage and capture the desired provenance information. This approach is more flexible and versatile than previous data provenance capturing approaches as users do not have to migrate to entirely new platforms in order to harness data provenance recording features.

At present the experiment model does not apply to all forms of investigation (e.g., continuous streams). We aim to address this in future work. Other future work involves exploring the integration of other domain specific metadata requirements. Furthermore, there is scope for expansion of the reporting functionality based on actual requirements. Additionally, we will extend the work through a summary/meta-investigation of the use of a popular dataset. Finally, it is desirable to expand the proposed system to be a service integrated with other computational products.

Acknowledgment

The authors would like to thank the anonymous reviewers for their comments during the peer review process.

References


research’, IBM Press, Toronto, Ontario, Canada, pp. 5.


Buyya, R., Yeo, C.S. & Venugopal, S. (2008), Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities, in ‘10th IEEE International Conference on High Performance Computing and Communications (HPCC)’.


Foster, I., Vckler, J., Wilde, M. & Zhao, Y. (2002), Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation, in ‘14th International Conference on Scientific and Statistical Database Management (SSDBM)’.


Privacy and Anonymity in Untrusted Data Stores

Jarrod Trevathan, Wayne Read, Hossein Ghodosi and Ian Atkinson

Abstract

This paper describes a security problem involving an online data repository, which acts as a proxy for multiple companies allowing their customers to perform online services (e.g., pay invoices). The repository’s host is trusted to honestly fulfil its duties in maintaining the data in a manner consistent with each companies’ required services. However, the information stored by the repository remains private in that the repository’s host cannot openly read any companies’ operational data, nor does it learn the identities of any companies’ customers. We contrast several approaches describing their viability for web deployment using existing technologies. This is a fundamentally new security problem with no established literature or clearly defined cryptographic solution. The project originated from a commercial attempt to design a secure online data archive. A sample implementation of the system is presented that allows a customer to pay and view invoices online via the data repository using a popular and widely available small business accountancy application.

Keywords: Privacy, security, authentication, encryption, web hosting, e-Commerce

1 Introduction

Since the 1960s, database technology has rapidly permeated all facets of society and transformed the way in which humans store and process data. The advent of networks and e-commerce now allows database systems to be distributed and remotely accessed by users anywhere around the world. This technology has even enabled the establishment of cloud storage (Armbrust et al. 2009, Buyya et al. 2008). However, despite the overwhelming benefits, there are numerous security concerns that limit an online database’s effectiveness (Kandukuri et al. 2009, Kaufman 2009). Some of the problems facing an enterprise with regard to securing its data in a networked environment include:

1. Access Permission – Users should only be able to access information that they are entitled to.
2. Denial of Service – Users performing computationally intensive queries may swamp system resources, thereby denying other users access.
3. Logging – User access to data and modifications must be logged to ensure possible security threats are monitored.

These concerns have been sufficiently addressed in literature (e.g., see (Inmon & Hackathorn 1994, Rosenthal & Sciore 2000, Warigon 1997, Widom 1995)) and most commercial database management systems provide support for authentication, concurrency controls and recovery mechanisms. However, the solutions proposed are only for the standard database scenario where the repository is owned and maintained entirely by a single controlling enterprise. The enterprise has complete access to view and modify the data. However, this may not always be the case.

We propose a new system where the data repository stores data that is not its own. That is, the repository provides a service by managing data that belongs to other companies. This raises new privacy and security concerns that have not been previously covered in the literature. Specifically, the repository manager should not have the ability to freely browse archived data that belongs to someone else. All repository users must trust that the manager will not reveal or sell potentially sensitive information to unauthorised parties.

This paper describes a security problem involving an online data repository, which acts as a proxy for multiple companies allowing their customers to perform online services (e.g., pay invoices). The repository’s host is trusted to honestly fulfil its duties in maintaining the data in a manner consistent with each companies’ required services. However, the information stored by the repository remains private in that the repository’s host cannot openly read any companies’ operational data, nor does it learn the identities of any companies’ customers. We contrast several approaches describing their viability for web deployment using existing technologies. This is a fundamentally new security problem with no established literature or clearly defined cryptographic solution. Note that this problem is different from a third party service that just merely hires out online storage (as is typical with cloud storage applications). In the scenario we address, the third party is unable to access any of the content it stores, but still is able to ensure that only authorised users can access the information. While recent advances in this area have been made by (Gentry 2010, Wang et al. 2009), no literature on cloud security appears to specifically address this problem.

The project originated from a commercial attempt to design a secure data archive. A sample implementation of the system is presented that allows a company to upload invoices from a popular accountancy program onto the data repository. A customer can log into the data repository via the Internet and can pay the invoice, which is then downloaded by the company and stored in its own personal database.
While the resulting implementation is not necessarily ideal, this paper’s purpose is to highlight the specifics of this problem and the shortcomings of the existing cryptographic mechanisms that must be combined to provide the desired level of security, authentication and anonymity. We hope that this work will form the basis for future work, and also will inspire other researchers to propose more sound and rigorous mechanisms to solve the private and anonymous data repository problem.

This paper is organised as follows: Section 2 describes the problem motivation, the processes/parties involved, and the privacy and security requirements for such a system. Section 3 presents a protocol to register users and maintain anonymity. Section 4 contrasts the differing approaches for ensuring the privacy of the database contents. Section 5 discusses what security mechanisms can be used to protect communications between the parties involved. Section 6 presents a specific implementation of the data repository system using existing web technologies which allows for online invoice payment. Section 7 gives a threat analysis for the proposal, and Section 8 provides some concluding remarks.

2 Problem Motivation

This section describes the initial problem motivation, the parties and processes involved, and the privacy and anonymity requirements. It also specifically outlines the assumptions and practical constraints imposed upon the system.

2.1 The Data Repository Service

Basically, the Data Repository (DR) Host provides an online service for a company’s customers. This removes the need and expense for the company to maintain its own website/online services for its customers. The costs for a business to create and maintain a website include:

- Salary for web programmers and graphic artists who must design and update the website;
- Hardware/software for developing and hosting the website;
- Ongoing costs for Internet access; and
- Administrative costs for handling the incoming/outgoing data and transactions.

A central data repository can spread these costs out amongst its clients (i.e., multiple companies using the service) to provide a less expensive service. This is especially attractive for small business where cost may be the limiting factor that prevents them from engaging in e-commerce. For example, the DR Host can offer a subscription to the system for $100 a year, which is substantially less than the aforementioned costs.

To make this scenario more practical, the company should not be required to constantly be online by the DR Host (i.e., need not have a permanent connection with the DR Host). Instead the company can log on to the DR Host’s website at will. It is the DR Host’s duty to maintain a constant online presence so that the company’s customers can log on to the DR Host’s website and access the company’s services (i.e., the system is asynchronous).

The repository is scalable in that the Host can offer the service to multiple companies simultaneously using the same database. That is, the DR Host can offer separate web portals for each company using a single server. This gives the appearance of a unique website.
for each company. However, for ease of discussion, we will only refer to a single company throughout the paper (i.e., the discussion will be from the viewpoint of a single company using the service). (See Figure 1.)

Many security proposals require elaborate mechanisms that are clearly not practical for immediate commercial deployment. In general, it is desirable that the customer should only be required to use a standard web browser when accessing the DR Host’s website. That is, the customer should not have to install a complicated application to interact with the DR Host’s database. The DR Host must provide a quality service using only existing web technologies.

Note that this constraint is the single-most important factor for the proposed system. It is this factor which renders the majority of cryptographic solutions commercially unviable for the task. The reader must keep this constraint in mind when considering the discussions in the coming sections.

2.2 Parties and Processes Involved

This section briefly describes the parties and processes involved in the online data repository system. These procedures are elaborated upon in later sections.

There are three main parties involved:

- **Data Repository Host** – Manages an encrypted database of customer records and provides a web interface for accessing these records;
- **Company** – Uploads/downloads customer records to/from the DR Host; and
- **Customer** – Accesses the database via a DR Host’s web interface to view/update customer files.

The proposed system consists of the following procedures and desirable characteristics:

1. **Registration between the company and the DR Host** – the company registers with the DR Host and specifies the type of services it requires;

2. **Company uploads/downloads records to/from the DR Host** – Ideally the company does not need to be permanently online. Rather the company should be free to upload/download customer records to/from the DR Host’s database at any time;

3. **Solicitation of a company’s customers** – Initially the company solicits customers regarding the service available (i.e., advertises the service). No further direct interaction should be required between the company and the customer;

4. **Registration between the customer and the DR Host** – the customer registers with the DR Host, the customer can then access the DR Host’s website in the same manner as any other e-commerce application; and

5. **Customer uploads/downloads records to/from the DR Host** – the customer anonymously accesses the DR Host’s services via a web interface to retrieve and modify records stored in the database.

2.3 Privacy Requirements

In addition to the regular database security issues outlined in Section 1, the proposed scenario has many unique security and privacy concerns. This section outlines these concerns and assumptions regarding the system.

Firstly, a company (and the customer) may be uneasy in trusting the DR Host with its operational data. That is, an unethical DR Host may attempt to browse, modify or sell everything stored in its database. As a result, some preventative measures must be in place to ensure the DR Host cannot do this. However, this must be weighed against the amount of information the DR Host requires to perform its duties adequately.

The problem lies in how to encrypt/decrypt the records in the database without the DR Host gaining access to the records. This is compounded by the fact that a customer should not have to remember a lengthy encryption/decryption key, nor should they have to use special software to interact with the DR Host (but rather use a standard web browser). Furthermore, anonymity is an issue in that the DR Host must not keep records regarding customers’ identities and which companies they belong to. Instead, customers should be able to use the service in an anonymous manner. However, the DR Host must still be able to ascertain that a customer is genuine, and only has permission to access files that s/he entitled to.

Table 1 summarises the desirable security and privacy requirements for the data repository application. Once again, the underlying constraint is that these requirements must be achieved using existing technologies rather than theoretical constructs.

<table>
<thead>
<tr>
<th>Security Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Confidentiality</strong></td>
<td>The data repository contents remain concealed from the DR Host and those not privy to the information</td>
</tr>
<tr>
<td><strong>Accountability</strong></td>
<td>The DR Host can verify that a customer is legitimate and has authorised access to his/her records</td>
</tr>
<tr>
<td><strong>Anonymity</strong></td>
<td>The DR Host does not learn customer identities</td>
</tr>
</tbody>
</table>

2.4 Assumptions

The DR Host is semi-trusted. This means that the DR Host is trusted to maintain the records in a manner consistent with the company’s policy and will not tamper with the database primary keys (used for indexing the records). However, the DR Host is not trusted in that it is not permitted to browse the database contents.

This paper will contrast differing practical approaches for dealing with the privacy and security issues that are specific to the private data repository problem. The following sections present several solutions for providing security and privacy based on existing cryptographic/security techniques. Each approach is scrutinized in terms of how it addresses the...
Figure 2: The registration process between a company and the DR Host, and then between a customer and the DR Host.

The aforementioned privacy and security concerns (see Table 1), its implications for the assumption that the DR Host is semi-trusted, the additional assumptions raised by each approach employed, and its viability for commercial deployment using existing web technologies.

3 Authenticating the Company and Customers

This section outlines the basic registration process between the parties involved, how they can be authenticated in future dealings, and how a customer’s anonymity can be preserved.

Figure 2 illustrates the registration protocol. The first procedure in the system is for the company to register with the DR Host. This can be done off-line, whereby the company makes physical contact with the DR Host, provides authenticating information (e.g., taxation and account numbers), a description of the services required, and signs a contract (Step 1 in Figure 2). During this process the DR Host supplies the company with a unique username and password that it can use to log onto the DR Host’s site for the purpose of up/downloading customer records. The DR Host also supplies the company with some authenticating information regarding its current customers during this stage (Step 2 in Figure 2). This information is referred to as a nonse list. A nonse stands for nonsensical data, and is essentially a unique random string. Each customer is associated with exactly one nonse (no two customers have the same nonse). The DR Host controls the nonse generation process, and ensures that no two nonses are ever the same. The company is responsible for assigning each nonse to a customer and makes sure no two customer’s are given the same nonse. An example of a nonse list-customer association chart prepared by a the company is as follows:

```
Nonse   Customer
ML28d78z  JoeBanana
87k787.K2  HomerSimpson
2j1487zA  Unassigned
...
Gd23AF4u  Unassigned
```

In this example there are two customers who have been solicited for the private data warehousing service. The remaining nonses are designated as unassigned until the company decides to solicit some more customers. If a company runs out of nonses, they can request an additional list from the DR Host.

Registering a customer with the DR Host is a more complicated procedure and is done online. Prior to registration, the company must first solicit customers announcing that the service is available. This is done through email or a physical letter sent directly to a customer (Step 2 in Figure 2). During the solicitation process, the customer is provided with some unique information (i.e., a nonse) that s/he can then present to the DR Host to prove that s/he is a legitimate customer. While the nonse can be delivered to a customer via email or telephone, these mediums can still be intercepted. Although it is not perfect, physical mail is a more secure option (similar to the process involved in obtaining a personal identification number for a bankcard). However, this must be weighed up against the speed and convenience of email, and the practicality of having a customer deal with a lengthy nonse.

When registering with the DR Host, a customer presents his/her nonse as evidence that s/he is a legitimate customer of a company, and also provides an email address (Step 4 in Figure 2). The DR Host compares the nonse against the nonse list. If it is a valid nonse (and it hasn’t previously been used), the DR Host marks the nonse as active and supplies the customer with a unique username and password (Step 5 in Figure 2). S/he can use this information to log into the DR Host’s client-side portal and access the company’s records (specific to him/her). If the nonse is invalid (does not match any nonse generated by the DR Host), or has previously been registered (it is already in use, or has previously been presented to the DR Host), the customer’s registration request is rejected and s/he is informed to contact the company that initially solicited him/her.

The nonse registration system not only provides authentication, but also anonymity for a customer. That is, the DR Host knows that this customer is legitimate, but does not actually know who they are. While the DR Host does learn which company the particular customer is associated with, this knowledge still does not benefit the DR Host in any way as it still does not know the customer’s identity.

4 Encrypting the Database

This section describes approaches for encrypting the database contents so that it remains private. While the some of the discussion may appear to be at a high level and considered to be superficial common knowledge, its purpose is ensure that the reader is familiar with how encryption works. The goal is to emphasise that the private data repository problem, within the constraints of the problem definition (i.e., the DR Host is semi-trusted), cannot be readily solved by using basic encryption alone if the system is to be commercially viable.

4.1 Encryption and the Private Data Repository

Encryption allows two parties (referred to as the sender and the receiver) to communicate such that an eavesdropper cannot interpret the message. This is achieved by scrambling (encrypting) the message prior to transmission, and then unscrambling (decrypting) the message upon receipt. Before engag-
ing in communications, both the sender and receiver must agree upon how to scramble the message (called a cipher) and how to ensure that only they can access the data (using a key). The encryption’s strength depends on the key size. The larger the key length, the greater the protection offered.

The problem this application faces is how to encrypt the database contents so that the DR Host (and any other unauthorised party) cannot freely view the database. However, this must be done in a manner that allows the DR Host to still operate the database from an administrator’s perspective. That is, the DR Host must still have unfettered access to the database tables’ primary keys and system catalogue in order to operate the database. Furthermore, the customers must not have to remember/store a lengthy key, nor should they have to interact with the company once registration has been completed.

4.2 Externally Imposed Security Limitations

There exist government restrictions on the use of cryptography which have practical limitations for the private data repository application (see (McCullagh 2001)). It is argued that in the interests of national security, private individuals should not be able to encrypt their communications such that a government intelligence agency cannot interpret the message if deemed necessary. Instead, commercial applications are required to use weak encryption. This provides a given level of privacy against general security threats, but can be eventually broken by a government agency with relative ease.

An alternative to weak encryption is key escrow (see (Micali 1993)). With an escrow system, a third party (referred to as an escrow agency), possesses some information that allows it to decrypt the encrypted information if necessary. The information held by the agency can either be a copy of the key itself, or some information that enables it to deduce the key. The agency will only decrypt the information when presented with the equivalent of a court order.

4.3 Private Information Retrieval

A Private Information Retrieval (PIR) protocol allows a user to retrieve an item from a server in possession of a database without revealing which item s/he is retrieving (see (Chor et al. 1998)). One trivial, but very inefficient way to achieve PIR is for the server to send an encrypted copy of the database to the user. This protocol gives the user information theoretic privacy for his/her query. Other approaches include making the server computationally bounded, or to assume that there are multiple non-cooperating servers, each having a copy of the database.

PIR is not applicable to a private data repository application. This is because the DR Host still has access to the entire repository contents even though they do not know which particular piece of information was retrieved during a query. A private data repository application requires that the DR Host does not know the repository contents and does not know what information the user has queried.

4.4 Key Exchange and Storage

Key exchange and storage are among the most vexing problems for the proposed data repository system.

Private key encryption involves using a single key to encrypt and decrypt the data. An eavesdropper who does not know the key cannot interpret the data. However, the problem with this approach lies in key exchange. That is, the sender and the receiver must exchange a secret key prior to communicating.

Figure 3: Securing the database contents using End-To-End Encryption

In terms of the private data repository application, the company and the customer must exchange a secret key prior to placing records in the data repository to ensure the DR Host and others cannot read the database contents. Key exchange cannot simply be performed by email as the key can be intercepted. If the DR Host conducts this attack, it would enable it access to all the stored data for the customer.

There are elaborate protocols for exchanging keys between two parties, but the general requirement is that both parties are online. This conflicts with the requirements outlined in Section 2 that the company does not need to be permanently online either during registration, or throughout the general system operation.

An alternative approach is public key encryption which uses separate keys for encryption and decryption (Diffie & Hellman 1976). This avoids the key exchange problems associated with private key encryption. The receiver publishes his/her public key (i.e., it is accessible to anyone) and keeps his/her private key secret. The sender encrypts the message using the receiver’s public key. Upon receipt, the receiver decrypts the message using his/her private key. Public key encryption requires an additional party (referred to as the Key Distribution Centre (KDC)) to maintain the receiver’s public key (i.e., to solve the key exchange issue).

In terms of the private data repository problem, both the company and the client would have to publish public keys on the KDC. When either party desires to upload a record, they obtain the public key from the KDC for encryption. However, this approach is rather cumbersome and requires everyone to trust the KDC (essentially trust is just being moved to another party). Although, SSL uses public key cryptography, the certificate authorities are already established. It would be expensive and impractical to maintain KDC services specifically for the private data repository application. Furthermore, public key encryption is also extremely slow (compared to private key encryption), and requires large key sizes. Due to these factors, public key encryption cannot be practically implemented for the private data repository scenario.

4.5 Encryption Function Placement

Assuming that key exchange has already occurred, there are two main approaches to where the encryption can be performed:

**End-to-End Encryption.** The company encrypts the data and stores it in the DR Host’s database. The customer then retrieves the data from the DR Host’s database and decrypts it. In the reverse process, the customer modifies the data, encrypts it and then stores it back in the DR Host’s database. The company then retrieves the information, decrypts it, and updates its own database. Figure 3 illustrates this process by indicating the placement of the encryption function.
While this approach is highly secure, it does have some practical limitations when being implemented with existing web technologies. The main issue is that the approach requires special purpose software to be installed on the customer's machine. That is, dedicated software to perform encryption and decryption. This cannot be achieved with a standard web browser, which means that the application now becomes platform dependent (i.e., separate applications are required for Windows and Mac). Furthermore, there are some security issues regarding how to store the key. Key sizes are required to be large (up to 1024 bits). It is not practical to force a human to remember this. The key cannot be stored in a cookie, nor on the user’s hard drive anywhere. During solicitation, it is unreasonable to print the key in the solicitation letter and expect the user to type it into the encryption/decryption program every time they want to use the service.

A Java Applet could be used to go some part of the way in resolving this issue. However, once again, this requires the user to download the Applet first prior to using the service. This is likely to deter users with slower connections. The problems regarding key storage still remain. Furthermore, there are still some problems with platform dependency, and users must ensure that their browsers are Java compatible.

The concluding sentiment for end-to-end encryption using existing web technologies is that while it is highly secure, it is also restrictive and expensive to develop (in terms of writing applications for each platform). This seriously impedes on the system’s practicality and commercial viability.

Application Encryption/Decryption on the DR Host. This approach allows the DR Host to perform the encryption and decryption. While the database remains encrypted, decryption is performed by the DR Host’s software before transmission to the customer (see Figure 4). The data stored in the repository still remains secret provided the DR Host does not tamper with the application software. This would be backed by a written privacy agreement between the DR Host and the company stipulating that the DR Host would not act in any dishonest manner. This setup still prevents any external attack (that does not pass through the DR Host’s application software) from accessing the database contents. However, the problem with this approach is that it erodes from the claims that the repository is truly private. That is, even more trust is now being placed in the DR Host.

The concluding sentiment for application encryption/decryption using existing web technologies is that it is easy and inexpensive to develop. It also avoids the problem of key exchange. However, it decreases the level of confidentiality offered by the data repository service. Although, when coupled with the nonce system, this approach can achieve confidentiality through anonymity. That is, even if the DR Host breaks into the application level encryption they still will not know the customer identities (as the customer’s personal details are not stored in the data repository). Only the soliciting company knows the mapping between the customers’ nones and identities.

5 Encrypting/Authenticating the Communications

This section outlines the threats that may be encountered during the transmission of information between the parties in the private data repository application.

5.1 Transmitting Information

Information transmitted between the company and DR Host, and between the DR Host and Customer is vulnerable. This can be observed by an outside party who could merely be eavesdropping on messages, or who has more sinister intentions such as modifying, fabricating, or blocking the messages. This is especially a problem when sensitive data such as credit card numbers and passwords are being transmitted. In this application, Secure Sockets Layer (SSL) can be used to establish a secured session to facilitate sending messages. The SSL protocol uses both public-key and private-key techniques to securely transfer information. Figure 5 illustrates the vulnerable communications between all parties in the system, and the placement of encrypted SSL sessions to prevent eavesdropping and other security threats.

5.2 Email Security Issues

Any email sent by the DR Host to either the company or the customer is not protected by the SSL session. As such, an eavesdropper can still read any passwords emailed. A customer obtains a password during the registration procedure. It is also possible for a customer to update a password at his/her discretion. These two procedures are relatively safe as the password can be communicated in real time under the protection of the SSL session. However, problems arise in the situation where a customer has forgotten his/her password. The typical remedy is for the customer to contact the bank in question. If the information is correct, then the bank will inform the customer (over the phone) what the password is. This has obvious problems as the customer’s password is now exposed to anyone observing network traffic.

Online banking websites commonly have a telephone number, which a customer can call in this situation the customer would then provide some identifying information (e.g., name, address, date of birth, etc.). If the information is correct, then the bank would inform the customer (over the phone) what the password is. However, this remedy would not work with the private data repository application as the DR Host does not know the customer’s identity. All the DR Host knows is the customer’s nonce and email address. Password notification could be achieved by asking the customer for his/her nonce, but it is probable that the customer may not remember or have kept records regarding his/her nonce.
An alternative is to use Privacy Enhanced Mail (PEM). PEM allows mail to be signed and encrypted using certificates similar to SSL. PEM is based on a concept called Web of Trust in which friends authenticate each other. Provided that the recipient knows and trusts someone who has authenticated the sender, then they have some degree of confidence in the sender. For encryption to occur, key exchange must take place. This is usually by the recipient sending an email to the encrypting party which contains a key. However, PEM is not widely used due to various difficulties involving public key infrastructure and problems with MIME encoding. It is also not really suitable to e-commerce applications, but rather personal email.

6 Private Invoicing System

This section presents an implementation of the private data repository system. The implementation allows customers to pay invoices online that are stored as Mind Your Own Business (MYOB) information in a company’s database.

- How the System Works -

Mind Your Own Business, is the name of an Australian multinational corporation that provides software and services to small and medium businesses. This software mostly entails accountancy applications. The invoice system extends MYOB by allowing online storage and transfer of MYOB information over the Web.

Figure 6 illustrates the basic operation of the invoice system. It involves three parties:

1. **DR Host** – Hosts an encrypted database of customer MYOB information and provides web hosting services on a company’s behalf.
2. **Company** – Has its own database containing MYOB information. The company uploads customer records to the data repository, downloads payments from customers, and uploads reconciled payments.
3. **Customer** – Accesses the database via the company’s web portal (created and hosted by the DR Host) to view/pay invoices.

The invoice system consists of the following software components:

- **MYOB Application** – Installation of MYOB’s business accountancy software.
- **Company** – This is a Windows application used by the company to interact with its MYOB database and the DR Host. It is responsible for:
  - Customer registration/solicitation;
  - Uploading/downloading MYOB information;
  - Company MYOB database settings and credentials; and
  - Web Service connection/login settings.
- **Web Services** – This application resides on the DR Host’s web server and accepts connection requests from Company applications. Web Services performs:
  - Customer registration/solicitation;
Figure 8: Company customer records and solicitation screen

- Uploading/downloading MYOB information;
- Database encryption;
- Hosting customer portals;
- Customer and company authentication; and
- Searching and paying customer invoices.

- **Customer Portal** – This is a unique web interface for each company registered with DR Host. Once registered, customers can log in to view and/or pay invoices that are saved in the data repository.

The invoice system has been implemented using Microsoft .NET and SQL Server. The remaining sections describe the specific processes involved and implementation issues.

### 6.1 Company Setup

When a company registers with the DR Host, it obtains a copy of the Company application. Figure 7 illustrates the Company MYOB Database and Site Connection Settings. the company enters the location of its MYOB database (along with user credentials) and provides details of the Web Services Port to connect to. Every time a company starts up the Company application, a connection dialogue (similar to dial up internet) is displayed. If the user selects the connect option, the Company application attempts to establish a connection with the Web Services. During this initial connection, the Web Services downloads any pending payments to the Company application.

### 6.2 Customer Solicitation

Figure 8 illustrates the Company Application’s Customer Records Screen (which is populated by the MYOB database and the Web Services). Customer solicitation consists of the following steps:

1. The company decides which customer to add from its MYOB database.
2. The company enters an email address and password for the customer.
3. The company clicks the “Update and Inform Customer” button. Two things happen during this stage:
   (a) The customer information is transferred to the Web Services and the data repository is updated.
   (b) The Web Services generates a GUI ID (i.e., a nonce) and sends an invitation to the email address supplied by the Company application. This differs slightly to the protocol outlined in Section 3 in that the DR Host is performing the nonce assigning process and notifying the customer. However, as the DR Host does not know the identity of who it is assigning a nonce to, this has the same effect as the protocol in Section 3.
4. Upon receiving the email, the customer clicks on a link (containing the GUI ID), and is directed to a registration page where s/he authenticates him/herself to the DR Host using the username and password contained in the invitation email. The Web Services implicitly uses the GUI ID to determine the validity of the customer’s registration request.
5. If the registration is successful, the Company application receives an acknowledgement from the Web Services the next time the Company application communicates with the Web Services.

### 6.3 Database Encryption

Database encryption occurs at the application level by the Web Services. The key is stored in the executable. Short of reverse engineering the executable, any human observing the Web Services will be unable to retrieve the key.

The Web Services uses the Triple DES encryption algorithm (NIST) that is part of the .NET security library. The Triple DES encryption algorithm is based on the Data Encryption Standard (DES) algorithm and conforms to U.S. laws on the use of cryptography (see Section 4.2). Triple DES is a symmetric (private key) block cipher algorithm and uses a key size of 192 bits. Note that at the time of writing, Advanced Encryption Standard (AES) was not supported by the .NET security library. However, there is no limitation within the private data archival problem on which encryption algorithm that can be used.

Once a table is uploaded, each field in a record is encrypted. The encryption process proceeds as follows:

1. A special table in the database (called Encryption Control) determines which fields in a table
are to be encrypted. This ensures that primary keys and other important indexing information is not lost.

2. For each row in a given table, each field (specified by the Encryption Control table) for the row is concatenated together. Each field is delimited by an ampersand (&).

3. The URL Encode Algorithm is applied to the concatenated string to ensure that all HTML special characters do not interfere with the ampersand delimiters.

4. The concatenated string is encrypted using the Triple DES encryption algorithm and is stored in an additional field called Cipher Text.

5. For each field that has been encrypted, a masking value is applied (also specified in the Encryption Control table). For example, a customer’s first name may be substituted with “XXXXXX” and his/her account balance substituted with “0.00”.

Decryption is essentially the reverse process:

1. Decrypt the encrypted string contained in the Cipher Text field.

2. Run the URL Decode Algorithm on the concatenated string to remove special HTML characters.

3. Extract each field from the concatenated string using the ampersand delimiters and store back into their original fields in the table (using the Encryption Control table as a reference).

6.4 Paying Invoices and Reconciling Payments

Figure 9 illustrates an example Customer Portal. Each portal is tailored specifically to each particular company (i.e., it appears as if it were the company’s own web site). Once a customer logs in, they can view and pay invoices. To pay an invoice, the customer flags the invoice as paid and then payment proceeds. To pay invoices, the customer must first log in to their own web site. Once a customer logs in, they can view and pay invoices. To pay an invoice, the customer flags the invoice as paid and then payment proceeds according to the methods agreed upon with the company. For example, via bank transfer, or through a payment provider such as PayPal or eWay. When the company receives the modified MYOB information from the customer, it is the company’s responsibility to follow up on the payment’s status.

6.5 Communications Protocol

The Company Application and the Web Services engage in the following communications:

1. Invite Customer – This uploads a customer’s email address and password to the Web Services to initiate the solicitation email.

2. Update Customer Details – Updates a customer’s email address and activity status with the Web Services.

3. Download Latest Payments – Downloads a dataset to the Company application containing payments made via the Customer Portal.


Transferring Records

Uploading records from the Company application to the Web Services is an inefficient process if the entire database is uploaded every time. The invoice system employs a hash system for improving upload efficiency. This consists of the following steps:

1. The Web Services computes a hash for each customer record.

2. The hash is concatenated with a customer’s record id and is sent to the Company application.

3. The Company application also computes a hash for each customer record.

4. The Company application compares the two hash values to determine which records to update in the data repository.

5. The Company application then transfers only the customer records that have changed or been deleted since the last upload.

7 Threat Analysis

This section gives a security and threat analysis for the proposed invoicing system.

Obviously the major weakness lies in the assumption that the DR Host is semi-trusted. If the DR Host wanted to disrupt a company’s operations, it would simply alter the database indices (primary keys) thus causing the wrong records to be retrieved/modified, or prevent access entirely. However, this scenario is unlikely as the DR Host is performing a service for the company. Therefore disrupting the company’s operation would only serve to brutalise the DR Host through lost revenue in that the company discontinues using the DR Host’s services.

The DR Host is responsible for the nonce generation process. It is feasible that the DR Host may generate duplicate nonces and/or deny customers access to the system. However, once again this would not benefit the DR Host in the long term for the aforementioned reason. The DR Host keeps track of active nonces and launches an investigation upon obtaining a duplicate nonce. This effectively thwarts anyone from stealing a customer’s nonce after they have registered and using it to fraudulently register. If a nonce is found to be in contention (i.e., it has already been registered), the customer applying for registration is directed to contact the soliciting company, and then an investigation is launched into who registered the nonce first.

Using application level encryption by the DR Host requires the company and the DR Host to sign a privacy agreement. This is a major weakness in the system’s privacy as the company must implicitly trust the DR Host not to browse the database contents. It is difficult to police this agreement and/or prove when the DR Host has breached its responsibilities. However, storing the key within the Web Services executable provides some level of protection, as essentially the application must be pulled apart to retrieve the key. If for some reason the key is lost, then the DR Host will lose the ability to decrypt its records. As a company maintains its own database locally, roll back from this situation can occur up to the last download from the DR Host. That is, the DR Host will lose all customer transactions that have not been downloaded by the company.

**Notes:**

2https://www.paypal.com

3http://www.eway.com.au
Although SSL is widely used, it has some limitations. Firstly, SSL is designed to provide point-to-point security. In the case where multiple intermediate nodes exist between the two endpoints, point-to-point security fails and end-to-end security is required. Secondly, SSL encryption is at the transport level rather than at the application. Messages are encrypted only during transmission over network. Other mechanisms are required to handle security of the messages in an application or disk.

During registration, a customer is provided with a username and password to access the service. It is possible that the DR Host can create a profile on individuals. For example, which records a customer accesses and his/her usage patterns. However, since the DR Host does not actually know who a customer is, and cannot decipher what the contents of the records are, this information is largely useless.

In this application, customers can only view and pay invoices. A denial of service attack is largely benign as there are no computationally intensive queries being executed by the DR Host. Viewing invoices requires a SELECT query to be executed to return rows that correspond to current and past invoices. The number of past invoices reviewable can be restricted to a given time range to avoid the burden of retrieving all invoices in existence for the user. Paying invoices only requires an INSERT or UPDATE query to be executed, which does not raise any concerns. The system may still be vulnerable to denial of service attacks involving flooding the DR Host with fake network traffic to prevent any legitimate users from accessing the service. However, protecting against such an attack is outside the scope of this paper.

8 Conclusions

This paper describes an online data repository, which acts as a proxy for multiple companies allowing their customers to perform online services (e.g., pay invoices). The system spreads costs out among multiple companies and provides web hosting, thereby decreasing the normal costs associated with developing and maintaining an individual e-commerce application. The DR Host is trusted to honestly fulfill its duties in running the data in a manner consistent with each companies’ required services. However, the information stored by the repository remains private in that the DR Host cannot openly read any companies’ operational data. This is achieved by encrypting the database’s contents (although some serious practical limitations have been identified). The DR Host also does not learn the identities of any companies’ customers. Each customer is issued with a unique nonce that is used for identifying purposes. The DR Host does not know the association between the client’s identity and the nonce they are using. This paper contrasts several approaches describing their viability for web deployment using existing technologies. An example implementation of the system is presented that uses MYOB to allow a customer to pay invoices online via the DR Host’s website.

This paper has presented a fundamentally new cryptographic problem with no existing solution. While some of the discussion may seem to be at a high level, this is only for the purpose of showing what security building blocks can be used to help solve the problem, and the apparent strengths and weaknesses of each approach employed. While it may appear that this is a straightforward encryption problem, further analysis of the requirements and the constraints of using existing technologies during the implementation severely compounds things. No single existing security mechanism appears to be adequate.

Instead the private data archival service requires several mechanisms to be applied in unison. The selection of certain approaches has definite trade-offs and undesirable consequences for other aspects of the system. It is left as an open problem for the cryptographic/security community to propose a more rigorous and mathematically sound solution.

References


An Empirical Study of Learning from Imbalanced Data

Xiuzhen Zhang  Yuxuan Li
School of Computer Science and IT, RMIT University
GPO Box 2476V, Melbourne 3001, Australia
Email: {xiuzhen.zhang, yuxuan.li}@rmit.edu.au

Abstract

No consistent conclusions have been drawn from existing studies regarding the effectiveness of different approaches to learning from imbalanced data. In this paper we apply bias-variance analysis to study the utility of different strategies for imbalanced learning. We conduct experiments on 15 real-world imbalanced datasets of applying various re-sampling and induction bias adjustment strategies to the standard decision tree, naive bayes and k-nearest neighbour (k-NN) learning algorithms. Our main findings include: Imbalanced class distribution is primarily a high bias problem, which partly explains why it impedes the performance of many standard learning algorithms. Compared to the re-sampling strategies, adjusting induction bias can more significantly vary the bias and variance components of classification errors. Especially the inverse distance weighting strategy can significantly reduce the variance errors for k-NN. Based on these findings we offer practical advice on applying the re-sampling and induction bias adjustment strategies to improve imbalanced learning.

Keywords: Bias-Variance Analysis, Imbalanced Learning

1 Introduction

In many applications class distribution is imbalanced, and the minority class is by far the primary interest. In these applications, typically the purpose of classification learning is to correctly predict the minority class. For example predicting defects in source code is of uttermost importance in software development projects, but defects only occur at a modest ratio of 5-10%. Accurate prediction of software defects can significantly reduce costs for software development.

Class imbalance has been reported to hamper the performance of standard classification models, whose aim is usually to optimize the overall accuracy. For example, the standard decision tree model tends to be overwhelmed by the majority class and ignore the minority class when making a decision about class labels. Re-sampling and adjusting induction biases have been popular approaches to combating class imbalance. Changing the prevalence of positive and negative examples by sampling is a widely used method for addressing class imbalance. Strategies include random under-sampling of the majority class, random over-sampling of the minority class and more advanced intelligent over-sampling techniques (Kubat & Matwin 1997, Chawla et al. 2002). Adjusting the induction bias to favour the minority class is another method to achieve accurate classification on the minority class.

A natural question is what effect these imbalanced learning strategies have on the behaviour of standard learning algorithms. In particular, how well a model fits the problem under consideration and to what extent a model is affected by variation in class distribution. To this end we employ the bias and variance analysis of classification errors (Kohavi & Wolpert 1996) to improve our understanding of the behaviours of different learning algorithms in the presence of class imbalance, and the effectiveness of sampling and imbalance induction bias adjustment on different learning models.

With the bias-variance decomposition, three types of classification errors are distinguished: the bias errors are the systematic errors associated with the learning algorithm and the problem domain, the variance errors are caused by variations in samples and the intrinsic errors are associated with the inherent uncertainty of the problem domain. Generally high bias errors indicate that a model is not correct for the problem domain and high variance errors indicate unstable classification by the model. Intrinsic errors are associated with noise of the problem domain and is independent of the learning algorithm.

We employ the bias and variance decomposition of classification errors to study the behaviour of three representative learning algorithms, the C4.5 decision tree algorithm (Quinlan 1993), the naive bayers (NB) (Good 1965, Duda & Hart 1973, Langley et al. 1992), and the k-nearest neighbour (k-NN) (Aha & Kibler 1991). We also study how random under- and over-sampling and advanced sampling techniques (See Section 3) vary the bias and variance components of errors for learning algorithms. We conduct a large-scale empirical study on 15 imbalanced datasets from the UCI repository and other disciplines.

Our main findings include: Imbalanced class distribution impedes the performance of standard learning algorithms in general, but depending on the learning algorithm, having varying effects on the bias and variance components of errors. The re-sampling strategies have varying effects on the bias or variance of learning algorithms. On the other hand adjusting the induction bias can significantly reduce the bias or variance components of errors, depending on learning algorithms. Based on these analysis we offer practical advice on applying the various re-sampling and induction bias adjustment strategies to combat the
1.1 Related Work

A few empirical studies have studied and compared different sampling techniques (Japkowicz & Stephen 2002, Drummond & Holte 2003, Hulse et al. 2007). However, no consistent conclusions have been drawn from these studies. Most of these studies use a few datasets for experiments and so the conclusion is hard to generalize. A large scale experimental study was conducted in (Hulse et al. 2007), but datasets used in the study are not publicly available. It was found in (Hulse et al. 2007) that the effectiveness of re-sampling for imbalanced learning depends on the evaluation metrics and base learning algorithms. All these previous studies have examined the effectiveness of imbalanced learning strategies on classification accuracy. In this paper we focus on explaining the behaviour of imbalanced learning strategies with the bias and variance decomposition of classification errors. Our bias and variance analysis relates the inconsistent behaviour of re-sampling to that it does not generally have consistent effect on the bias or variance errors of learning algorithms. Importantly we offer practical advice on how to combine re-sampling strategies for effective imbalanced learning.

When classification errors (misclassification costs) of different classes are distinguished, accuracy maximization is replaced with cost minimization – high cost is associated with misclassifying minority samples. Cost-sensitive learning methods ( Domingos 1999) have been proposed to learn from imbalanced class distribution. In (Elkan 2001) the problem of optimal learning with different misclassification cost is studied. It is shown that in theory that rebalancing the positive and negative distribution has little effect on the decision tree and Bayesian methods. However this general theoretical result does not necessarily suggest that re-sampling strategies do not work in specific applications.

In an excellent survey by Weiss (Weiss 2004), techniques for imbalanced learning were reviewed. Sampling and adjusting decision bias are recognized as a commonly used technique for dealing with rarity, but no conclusion was drawn regarding their effectiveness. With some recent developments, advanced sampling techniques were proposed (Liu et al. 2006) for specific imbalanced learning applications. However the general utility of these techniques is yet to be studied. The bias and variance analysis of classification errors (Kohavi & Wolpert 1996) is a widely used approach to provide insight into the error performance of classifiers. It has been used in various studies to compare the relative performance of different learning models, for example (Bauer & Kohavi 1999, Webb 2000, Putten & Someren 2004). To the best of our knowledge, it has not been used to study the problem of imbalanced classification.

2 Re-Sampling Strategies for Imbalanced Learning

Based on the assumption that standard learning methods perform better with equal class distribution, re-sampling training instances has been proposed for imbalanced learning.

2.1 Random Under-sampling and Over-sampling

Random under-sampling and over-sampling training instances are two basic methods of re-sampling for imbalanced learning. With under-sampling, examples of the majority class are randomly eliminated so as to achieve balanced class distribution. With over-sampling, examples of the minority class are randomly duplicated to achieve even class distribution. In essence random over-sampling does not introduce new examples to directly bias the induction process. Some studies have shown that, compared with under-sampling, simple over-sampling is less effective at improving recognition of the minority class (Drummond & Holte 2003). However another study that used artificial domains came to the opposite conclusion (Japkowicz & Stephen 2002).

2.2 Advanced Sampling Methods

A more advanced sampling method is to combine under-sampling and over-sampling to achieve balanced class distribution. This potentially can remedy the drawbacks when under-sampling and over-sampling are used separately. The Synthetic Minority Oversampling Technique (SMOTE) (Chawla et al. 2002) generates minority-class examples by adding examples from the line segments that join the k minority-class nearest neighbours. This presumably leads to better generalization compared with random over-sampling. It was shown that a combination of over-sampling the minority class using SMOTE and under-sampling the majority class can achieve better classifier performance than only under-sampling the majority class. However, the effect of SMOTE alone on imbalanced learning has not been extensively studied.

3 Adjusting Induction Bias for Imbalanced Learning

In this section we discuss three popular base learning algorithms and where applicable, strategies adjusting their induction bias for imbalanced learning.

3.1 The Decision Tree

Some strategies have been proposed adjusting the decision tree induction to be more sensitive to imbalanced class distribution (Hulse et al. 2007):

- For imbalanced class distribution, pruning a decision tree can over generalize and completely ignore the positive class, and so decision trees are fully grown without pruning.
- Based on similar consideration, the minimal number instances for leaves of a decision tree is set to one rather than a number > 1.
- With Laplace smoothing (Good 1965) the probability for the positive class at a leaf node is estimated as \( \frac{L_p+1}{L_p+L_n+2} \), where \( L_p \) and \( L_n \) are respectively the number of positive and negative samples at the leaf. It has been shown that Laplace smoothing improves the tree performance for skewed class distribution.

3.2 The Naive Bayes

The Naive Bayes (NB) is a simple probabilistic induction model based on the Bayes Theorem (Duda &
Proceedings of the Twenty-Second Australasian Database Conference (ADC 2011), Perth, Australia

Induction bias  
- Bias error
- Variance error

Correct  
- Bias error
- Variance error

Incorrect  
- Bias error
- Variance error

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Induction bias</th>
<th>Correct</th>
<th>Incorrect</th>
</tr>
</thead>
<tbody>
<tr>
<td>C4.5</td>
<td>medium</td>
<td>varying</td>
<td>varying</td>
</tr>
<tr>
<td>k-NN</td>
<td>weak</td>
<td>low</td>
<td>high</td>
</tr>
<tr>
<td>NB</td>
<td>strong</td>
<td>low</td>
<td>low</td>
</tr>
</tbody>
</table>

Table 1: The bias-variance relationship for C4.5, k-NN and NB

Hart 1973). NB estimates probabilities based on the attribute independence assumption. Although this assumption does not hold for many problems, NB often exhibits competitive classification accuracy compared with other learning algorithms. NB has a very strong induction bias and does not have any parameters that can be adjusted for imbalanced class distribution.

3.3 The k-Nearest Neighbour

With the k-nearest neighbour (k-NN) (Aha & Kibler 1991) algorithm, class labels of the k training instances closest to a test instance help determine the class label of the test instance. Inverse distance weighting is to weigh the vote of each neighbour according to the inverse of its distance from the test instance (Mitchell 1997). By taking the weighted average of the k distances nearest to the test instance, smooths out the impact of isolated noisy training instances. Furthermore, it lifts the weight of instances from the minority class closest to the test instance — a point that has been largely overlooked by existing studies.

4 The Bias-Variance Analysis

The bias-variance analysis of classification errors is a useful tool for analysing classifier behaviour. This analysis decomposes classification errors into three terms, derived with reference to the performance of a learning algorithm when trained with different training sets drawn from some reference distribution of training sets:

- **Squared bias** denotes the systematic component of classification errors — how closely a learner describes the decision surfaces for a domain.
- **Variance** describes the component of classification errors from sampling — how sensitively a learner responds to variations in the training sample.
- **Intrinsic noise** measures the degree to which the target quantity is inherently unpredictable, which equals the expected cost of the Bayes optimal classifier.

There have been several proposals for the definition of the three terms for classification learning. The definition by Kohavi and Wolpert (Kohavi & Wolpert 1996) is widely used and is the definition we will use in this study. Given that an error has cost 1 and a correct prediction has cost 0, the expected error rate for a target function f and a training dataset of size m is

\[
err = \sum_x P(x)(noise^2_x + bias^2_x + variance_x)
\]

where x ranges over the instance space, and P(x) is the prior probability of x.

In practical experiments it is impossible to estimate the intrinsic noise. The algorithm proposed in (Kohavi & Wolpert 1996) generates a bias term that includes the intrinsic noise. In their method, the training dataset is divided into a training pool and a test pool randomly. Each pool contains 50% of the training instances. Fifty training sets are generated from the training pool by random sampling. Classifiers are trained on each of the 50 training set, and bias and variance errors are estimated from the classifiers on the test set.

Generally there is a bias-variance tradeoff (Kohavi & Wolpert 1996). When adjusting a learning algorithm so that it is more sensitive to the training samples, its bias errors shrink but the variance errors increase. Learning models that overfit the given training data often have high variance errors — their results depend closely on the given training data and thus vary for different training datasets. On the contrary, learning models with a strong induction bias are less likely to overfit and bias is a source of prediction errors if the induction bias of the model is not correct for a domain.

General description of the C4.5 decision tree, k-nearest neighbour and Naive Bayes learning algorithms in terms of their effect on the bias and variance components of in classification errors is presented in Table 1. With the strong attribute-value independence assumption during classification, Naive Bayes has a strong induction bias. If the induction bias of NB is correct for the problem domain, then NB demonstrates low bias errors otherwise high bias errors. Without any representation model, the classification decision of k-NN does not have induction bias and its classification errors mainly come from variations in the distribution of training data. With a decision tree as the representation model, C4.5 has a medium level of induction bias. As a result the classification errors of C4.5 can come from the bias, variance or both components.

We can now characterise performance of the three base learning algorithms for imbalanced learning in terms of bias-variance decomposition. We can also characterise the effect of various re-sampling and induction bias adjustment strategies on the bias and variance components of errors.

5 Experiment Design

Our study will focus on the two-class problem with a minority (positive) class and a majority (negative) class. We compile datasets from various sources to study the utility of re-sampling and induction bias adjustment strategies for classification.

Fifteen real-world datasets from highly imbalanced (the minority 4.35%) to moderately imbalanced (the minority 30%) are used in our experiments, as listed in Table 2. UCI (Asuncion & Newman 2007) imbalanced 2-class datasets include those from natural 2-class domains, and those constructed by choosing a minority class as the positive and the remainder as negative instances. The Oil dataset (Kubat et al. 1998) (marked with *) has been extensively used in imbalanced learning experiments. PC1, CM1 and KC1 (marked with *) con-
tain metrics data at the module level for predicting defects in NASA software development projects (http://mpse.jpl.nasa.gov/index.html).

In our experiments, we use classifiers J48, NB and IBk of the WEKA (Witten & Frank 2005) data mining software for the base algorithms C4.5 (Quinlan 1993) decision tree, NB and k-NN. The base algorithms with default settings, which usually are designed for uniform class distribution, are compared against their settings for adjusting induction bias for skewed class distribution. Specifically, for J48 the imbalance-favourable settings are without pruning, with Laplace-smoothing and that minimum one instance is allowed for a leaf node. For IBk, the imbalance-favourable settings are \( k=3 \), and inverse-distance weighted voting. There are not any parameter settings for adjusting bias for imbalanced distribution.

We use the instance re-sampling filters in WEKA to implement the re-sampling strategies in Section 3. For under-sampling, the majority class is randomly under-sampled with replacement so that it has the same number of instances as the minority class. For over-sampling, the minority class is randomly over-sampled so that it has the same number of instances as the majority class. The SMOTE filter in WEKA is used for the SMOTE over-sampling strategy.

6 The Bias-Variance Analysis of Imbalanced Learning

In our experiments we employ the bias and variance decomposition software in the WEKA toolkit to estimate the squared bias and intrinsic noise combined error and the variance error component for classification algorithms. The bias and variance decomposition algorithm in WEKA precisely follows the approach of (Kohavi & Wolpert 1996), as described in Section 4.

6.1 The bias-variance decomposition for base learning algorithms

Fig. 1 shows the bias and variance decomposition of expected errors for the base algorithms C4.5, k-NN and NB on 15 datasets in our experiments. Generally for all three base algorithms, the bias component is the dominant source of errors. Not surprisingly NB has the highest bias component of errors — except on Oil where bias comprises 43.94% of errors, on all other datasets bias is the bigger proportion of errors, comprising on average 81.02% of errors. C4.5 and k-NN demonstrates varying bias-variance decomposition on 15 datasets, with the bias portion of errors ranging from 43.82% (C4.5 on Vehicle) to 98.10% (C4.5 on Flag).

The BVD profile for base algorithms differ on each dataset. For example on the most imbalanced Oil dataset, the bias component of errors for C4.5, k-NN and NB are dramatically different, 57.89% for C4.5, 79.79% for k-NN and 43.94% for NB respectively. On the Vehicle dataset, the bias component is respectively 43.82% for C4.5, 51% for k-NN and 92.94% for NB.

Our analysis suggests that imbalanced class distribution has different effect on the base learning algorithms and it varies significantly for different problems. This complex profile of bias-variance component suggests that learning from imbalanced class distribution is a challenging problem.

6.2 The bias-variance decomposition for sampling techniques

A relatively large number of instances in the training dataset is needed to ensure accurate estimation of errors. In our experiments the smallest dataset (Hepatitis) contains 155 instances, which we consider sufficiently large. Undersampling the majority class to match the minority can result in some datasets have a very small number of instances. We chose datasets whose total number of instances is at least 100 after under-sampling. As a result only 10 datasets are included in our experiments of the bias-variance decomposition for the random under-sampling strategy, as is shown in Fig. 2. To compare under-sampling against other sampling techniques, the same datasets were used for the experiments of the other sampling strategies.

From Fig. 2 it can be seen that generally random under-sampling increases both the bias and variance errors for all three base learning algorithms, and the increase in variance errors is more pronounced than that in the bias errors. k-NN demonstrates the most consistent and significant response to under-sampling — on all 10 datasets both its bias and variance components of errors significantly increase, and the increase in variance is more pronounced. C4.5 is also very sensitive to under-sampling, and shows increment in both bias and variance errors on all 10 datasets. In contrast NB is not so sensitive to the under-sampling strategy. On Kc1 the bias errors for under-sampling
Figure 1: The bias-variance decomposition for base algorithms C4.5 (J48), k-NN (IBk) and NB

Figure 2: The bias-variance decomposition for the under-sampling strategy
Figure 3: The bias-variance decomposition for the over-sampling strategy

Figure 4: The bias-variance decomposition for the SMOTE over-sampling strategy
Figure 5: The bias-variance decomposition for base C4.5 and imbalanced C4.5

Figure 6: The bias-variance decomposition for base $k$-NN and imbalanced $k$-NN
remain the same as that of the standard NB.

In Fig. 3, generally for random over-sampling, the three base learning algorithms do not show significant changes in the bias errors on most datasets. On the other hand random over-sampling has different effects on the variance of the three base learning algorithms — C4.5 and k-NN show increase in variance, while NB does not demonstrate change in variance errors on most datasets.

In Fig. 4, generally SMOTE over-sampling does not show significant changes in either the bias or variance errors on most datasets (except for the bias component of C4.5 on Haberman), and this is universally true for all three learning algorithms.

As a summary, our experiments have demonstrated that either random over-sampling or SMOTE intelligent over-sampling does not significantly change the bias errors of the three base learning algorithms. This can be explained by that the generated new samples are either replicates or near replicates of existing positive samples and they do not produce effect on the decision boundary for classes. Over sampling generally can also negatively affect the variance errors of the decision tree and k-nearest neighbour models, and it does not change the variance of NB.

In contrast, under-sampling significantly changes the bias and variance of base algorithms, due to the fact that some “important” samples affecting the decision for class boundary may have been removed. However, generally the effect is “negative”, that is the bias and variance errors of all algorithms are exacerbated rather than reduced.

6.3 The bias-variance decomposition of induction bias adjustment techniques

Fig. 5 and Fig. 6 show the bias and variance decomposition of expected errors for C4.5 and k-NN respectively on 15 datasets. It can be seen from Fig. 5 that for C4.5 the imbalance-favourable induction adjustment strategies, namely no pruning, minimal one instance for leaf nodes and Laplace smoothing, do not change the bias errors of the decision tree model on most datasets, but they significantly increase the variance errors on many datasets (p-value=0.00055 in the Wilcoxon signed rank test). Given that these strategies mainly affect the decisions towards the leaves of a decision tree, it is not surprising that the variance of the decision tree algorithm increases significantly on most datasets — most variance errors come from leaves at the bottom of the tree. In contrast bias errors towards the root of the tree are mostly unaffected by these bias adjustment strategies and therefore the bias errors of the algorithm do not change on most datasets.

In Fig. 6 the “inverse distance weighting” heuristic significantly reduces the variance component of k-NN significantly on all 15 datasets, with p-value = 0.00083. It is also noteworthy that the strategy has never exacerbated the bias errors on any dataset. Furthermore on CM1, SPECT, Hepatitis, Splice, Haberman and German, bias errors are significantly reduced.

Comparing the imbalance bias adjustment strategies of C4.5 and k-NN, the strategies in the decision tree algorithm has focused on modifying the representation tree for classification, especially towards the leaves at the bottom of the decision tree. Such strategies exacerbate the variance errors of the decision tree model. As a learning algorithm without explicit model representation, the imbalance induction bias adjustment of k-NN reduces both the bias and variance errors of the learning algorithm. This shows that the strategy has improved both the generality and stability of the k-NN algorithm.

7 Discussions and Conclusions

In this paper we have studied the re-sampling approach and the adjusting induction bias approach for employing standard learning algorithms for imbalanced classification. The re-sampling strategies we consider include random over-sampling, random under-sampling and SMOTE intelligent over-sampling. We employ bias-variance analysis to study the behaviour of re-sampling and imbalance bias adjustment on 15 real-world imbalanced datasets for popular algorithms, including the decision tree, Naive Bayes and k-nearest neighbour.

We have found that imbalanced class distribution impedes the performance of standard learning algorithms in general, but depending on the learning algorithm, having varying effects on the bias and variance components of errors. For the Naive Bayes algorithm, class imbalance mainly presents as a high bias problem, whereas for the decision tree and k-nearest neighbour models, errors can come from either the bias or variance component, depending on the application domain.

Over-sampling alone, either randomly or intelligently like SMOTE, does not have significant impact on the bias of any of the three learning algorithms. It exacerbates the variance errors of the decision tree and k-NN to different degrees but does not change the variance of the Naive Bayes. Random under-sampling on the other hand, exacerbates the bias and variance errors of all three learning algorithms. Our practical advice in this regard is therefore to apply the sampling strategies on problems with low bias errors and to intelligently combine the over-sampling with under-sampling to reduce the variance errors. More research is needed to investigate how to best combine under-sampling and over-sampling.

Our experiments on C4.5 has shown that the strategies adjusting the imbalance induction bias for the decision tree model as described in Section 3 can exacerbate the variance errors, while such strategy for the k-NN model can reduce the variance as well as bias errors. So for the decision tree model the imbalance bias adjustment strategies should be executed with care. Specifically they should be applied to problems with low variance errors. In contrast the imbalance induction bias adjustment strategy for the k-NN algorithm is strongly recommended.

The simple Naive Bayes model, with a strong induction bias, presents as a high bias problem with the imbalanced class distribution. It is noteworthy that our experiments show that the Naive Bayes model is a stable model whose bias and variance are not sensitive to the various sampling techniques. An alternative promising approach to improving the Naive Bayes model for imbalanced learning may be to reduce the bias component by relaxing the “naiveness” of the induction process.

Acknowledgements

The authors thank Robert Holte for providing the Oil dataset.

References


Drummond, C. & Holte, R. C. (2003), C4.5, class imbalance, and cost sensitivity: why under-sampling beats over-sampling, in ‘Workshop on Learning from Imbalanced Data Sets II’.


Effective Scheduling Algorithm for On-Demand XML Data Broadcasts in Wireless Environments

Yongrui Qin\textsuperscript{1} Hua Wang\textsuperscript{1} Jitian Xiao\textsuperscript{2}

\textsuperscript{1}Department of Mathematics & Computing
University of Southern Queensland, Australia,
Email: yongrui.qin@usq.edu.au, wang@usq.edu.au
\textsuperscript{2} School of Computer and Security Science
Edith Cowan University, Australia,
Email: j.xiao@ecu.edu.au

Abstract

The organization of data on wireless channels, which aims to reduce the access time of mobile clients, is a key problem in data broadcasts. Many scheduling algorithms have been designed to organize flat data on air. However, how to effectively schedule semi-structured information such as XML data on wireless channels is still a challenge. In this paper, we firstly propose a novel method to greatly reduce the tuning time by splitting query results into XML snippets and to achieve better access efficiency by combining similar ones. Then we analyze the data broadcast scheduling problem of on-demand XML data broadcasts and define the efficiency of a data item. Based on the definition, a Least Efficient Last (LEL) scheduling algorithm is also devised to effectively organize XML data on wireless channels. Finally, we study the performance of our algorithms through extensive experiments. The results show that our scheduling algorithms can reduce both access time and tuning time significantly when compared with existing work.

Keywords: wireless environment, data broadcast, on-demand, XML, multi-item, scheduling algorithm

1 Introduction

With the rapid development of wireless network technologies, users with mobile devices (such as palmtops, PDAs, WAP phones and so on) can access a large amount of information at anytime from anywhere. Over the past few years, mobility and portability have created an entire new class of applications. For example, information services, including news, stock quotes, airline schedules, weather report and traffic information, are becoming more and more popular and helpful. Logically, information access via these wireless technologies can be classified into two basic ways: point-to-point access and broadcast (Imielinski et al. 1997, Xu et al. 2002).

Point-to-point access employs a pull-based approach where a mobile client initiates a query to the server which in turn processes the query and returns the result to the client over a point-to-point channel. It is suitable for lightly loaded systems in which wireless channels and server processing capacity is not severely contended.

On the other hand, data broadcast is an efficient way for public information delivery to a large number of mobile users. It offers great scalibility, good power consumption, and efficient bandwidth utilization (Imielinski et al. 1997, Xu et al. 2002). In addition, it allows an arbitrary number of clients to access data simultaneously and is thus particularly suitable for heavily loaded systems. Recently, there has been a push for such systems from the industry and various standard bodies. For example, born out of the International Telecommunication Union’s (ITU) International Mobile Telecommunications “IMT-2000” initiative, the Third Generation Partnership Project 2 (3GPP2 2007) is developing Broadcast and Multi-cast Service in CDMA2000 Wireless IP network.

There are two typical broadcast modes for data broadcast (Xu et al. 2002): 1) Broadcasting Mode. Data is periodically broadcast on the downlink channel. Clients only “listen” to that channel and download data they are interested in; 2) On-Demand Mode. The clients send their requests to the server through uplink channel and the server considers all pending requests to decide the contents of next broadcast cycle. In this paper, we focus on on-demand data broadcasts.

Access efficiency and power conservation are two important issues in wireless data broadcast system since mobile clients are typically powered by batteries with limited capacity. Accordingly, two critical metrics, access time and tuning time are used to measure the system’s performance (Imielinski et al. 1997, Xu et al. 2002). Access time refers to the time elapsed from the moment a query is issued to the moment it is answered while tuning time refers to the time a mobile client stays in active mode to receive the requested information.

Aiming at reducing tuning time, Imielinski et al. (1997), Xu et al. (2002), Lee & Zheng (2005) study air indexing techniques. They introduce some auxiliary data structures in broadcast to indicate the arrival time of each data item on a wireless channel. As a result, mobile clients know the arrival time of the requested data items in advance and can switch to the energy-saving mode (dorm mode) during waiting. Therefore, the advantage of air index is reducing tuning time and thus a longer battery life can be attained.

Broadcast schedule determines what data items to be broadcast by the server and also the order of data items on wireless channels. Acharya et al. (1995), Acharya & Muthukrishnan (1998), Åkesson & Franklin (1999), Sun et al. (2003), Huang et al. (2010) (in print)
investigate scheduling techniques which aim to reduce access time. These studies are under the premise that each user query requires only one data item. Other work studies the multi-item queries scheduling problems (Chung & Kim 1999, Lee et al. 2002, Sun et al. 2008).

Besides the traditional flat data information, such as records in relational databases, more and more information are described in semi-structured format over the past few years. XML has rapidly gained popularity as a standard to represent semi-structured information, and is also considered an effective format for data transmission and exchange.

**Motivation** To a large extent, scheduling XML data is similar to multi-item scheduling problem. However, previous multi-item scheduling algorithms mainly take advantage of access frequencies of data items and different queried result sets containing multiple items (Chung & Kim 1999, Lee et al. 2002, Sun et al. 2008) but not the sizes of data items as well. In contrast, data items (or XML files) have a variety of lengths in XML data broadcasts and thus the lengths of data items should be taken into account. Furthermore, in traditional data broadcasts, data items are usually considered atomic, but in XML data broadcasts, the data items are no longer atomic. Therefore, when scheduling XML data, we should consider not only the order of data items on the wireless channel, but also the structured characteristics of XML data.

To the best of our knowledge, little scheduling work but also the structured characteristics of XML data, we discuss two naive schemes on how to match XML data with mobile users’ queries. On the basis of these two schemes, we make use of the structured characteristics of XML data and propose to only broadcast as less redundant information as possible to reduce tuning time. We also put forward a more practical scheme which combines similar queried results to achieve better performance of access time with a little overhead. Then we analyze the scheduling problem of on-demand XML data broadcasts and devise an improved scheduling algorithm to achieve better access efficiency. In summary, the main contributions of this paper are:

- We propose to pre-process XML data before broadcast to reduce both tuning time and access time. Taking advantage of the structured characteristics of XML data, we discuss two naive schemes on how to match XML data with mobile users’ queries. Then a more practical scheme is put forward to reduce access time with a little overhead to the optimal tuning time.

- We analyze the scheduling problem of on-demand XML data broadcasts and give a formal definition of the efficiency of a data item. Based on this definition, we then devise a Least Efficient Last (LEL) scheduling algorithm to effectively schedule XML data on wireless channels. In addition, computing complexity and theoretical analysis of LEL scheduling algorithm are also given.

- We perform extensive experiments to study the effectiveness of our solutions. These experiments show that our solutions can achieve better performance when compared with existing work.

We proceed with related work in Section 2. Section 3 proposes a pre-processing technique for on-demand XML data broadcasts to reduce tuning time and to achieve better access efficiency. Section 4 analyzes the scheduling problem of XML data in on-demand broadcasts and puts forward an improved scheduling algorithm. Section 5 presents our experimental study and evaluates the performance of the proposed approach. Finally, Section 6 concludes this paper.

## 2 Related Work

Recently, a lot of work dealing with XML data broadcast has appeared. Chung & Lee (2007), Park et al. (2005, 2006) address the performance optimization of query processing of XML streams in wireless broadcast. On the other hand, Qin et al. (2009), Sun et al. (2009) design some indexing techniques for XML data broadcasts based on existing XML indexing techniques. However, their work mainly focuses on air indexing techniques and does not discuss the scheduling problem in XML data broadcasts.

Multi-item scheduling problem is also related to the scheduling problem of XML data. Multi-item scheduling problem is proved to be a NP-Complete problem (Chung & Kim 1999). Also a scheduling method for multi-item queries called QEM is introduced, which opened up a new perspective in this field. In addition, the measure Query Distance (QD) is defined. It shows the coherence degree of a query’s data set in a schedule. Chung & Kim (1999) prove it could represent the AT of the query. The basic idea of QEM is to expand the data of each query according to their access frequencies.

Lee et al. (2002) propose Modified-QEM. It loosens the restriction that the QD of previously expanded queries cannot be changed. In fact a “move” action could be executed so that the QD of the new coming query is optimized. By doing “move” at specified times, the algorithm improved the QEM performance.

Chang & Hsieh (2004) propose another algorithm called Improved QEM. It employs the association rules in data mining to discover the relationship among data items and then applies QEM method on data sets instead of queries. Improved QEM method does consider different length of data items. However, it is still QEM based and is proven to be less efficient than the method proposed by (Sun et al. 2008).

Sun et al. (2008) put forward a scheduling algorithm named LRL (short for Least Required Last) algorithm. Since the queries which will be fully satisfied by the last broadcast data item have the longest access time, it is reasonable to have as fewer queries as possible to be fully satisfied by the last data item. Thus, the last broadcast data item should have the least access frequency. Up to now, LRL scheduling algorithm shows the best results when compared with other work. However, this algorithm does not make use of different sizes of data items.

## 3 Pre-processing XML Data

Consider the queried results of different mobile users’ queries submitted almost at the same time. In traditional data broadcast, the queried result comprises of a set of data items, such as $d_1$, $d_2$, $d_3$ and so forth. These data items are independent of the queries which means the queries will not affect the content of the data items. In other words, a matched data item is regarded as atomic item and cannot be divided into smaller ones. The queries match the whole data item

---

1. Fully satisfied’ means that the issuers of these queries have received all data items they required.
other than some parts of its content. As a result, traditional scheduling algorithms only need to consider the placements of data items that are selected by mobile users’ queries.

In contrast, the queried results of XML data are no longer atomic data items, but a chunk of structured data that satisfies users’ queries. This chunk of structured data is usually part of a whole XML file. Actually, different queries may request the same XML file. However, only parts of the file match these queries and these parts might be quite different from each other. To explain it in another way, the queried results in each XML data file are dependent on users’ queries that are sent to the server. Thus, the queried results should not be regarded as atomic data items (see Figure 1). If each XML file is treated as an atomic data item, which is similar to traditional data broadcast, mobile users who have submitted queries that match some parts of this XML file all need to download the whole XML file from the wireless channel. It obviously brings down the overall access efficiency and energy efficiency. Therefore, in order to enhance the broadcast system performance, a pre-processing phase is needed in on-demand XML data broadcasts. According to this finding, we first discuss two naive schemes for the pre-processing phase.

3.1 Two Naive Schemes

There are two naive schemes to pre-process XML data in on-demand broadcasts. One scheme follows the concept of traditional data broadcast and does not take advantage of the structured characteristics of XML data while the other scheme only broadcasts XML snippets or sub-trees those mobile users’ queries

In this case, the total tuning time is 12, if simply in terms of nodes (4 nodes for each query).

2. Snippet Scheme: In this scheme, by taking the structured characteristics of XML data into consideration, the server regards XML files as non-atomic items and all XML files are dividable. The server broadcasts every XML snippet that satisfies any of the mobile users’ queries. For example, in Figure 1, there are three XML snippets that satisfy the three queries respectively. All these snippets are different from each other and so the server broadcasts each of them on the wireless channel. The mobile users of each query can just download the required XML snippet and skip the other two snippets. Therefore, the total tuning time is 9 (that is 3+2+4), if in terms of nodes.

On the one hand, Atomic scheme is commonly used in previous work (Chung & Lee 2007, Park et al. 2005, 2006, Sun et al. 2008, 2009, Qin et al. 2009) since it results in least broadcast content on the wireless channel. On the other hand, Snippet Scheme is just a theoretical model and is never used in previous work. This is because although it can achieve the best tuning time for each mobile user, it can cause a lot of redundant content to be broadcast on the wireless channel. Thus, lots of mobile users have to spend longer waiting time. Clearly, in the atomic scheme, most of the mobile users download a large amount of redundant XML data and thus the tuning time is extreme large while in the snippet scheme, each mobile user only downloads the exact XML data they require and thus the tuning time is optimal. Nonetheless, the snippet scheme is still not the ideal one because it does not consider the similarity among different snippets and thus better access time can still be expected. Hence, we put forward a more effective and practical scheme in the following subsection.

3.2 Combining Scheme

In the combining scheme, we further make use of the similarity of different XML snippets based on the snippet scheme. In this way, we can reduce the overall access time with a little tuning time overhead.

For instance, in Figure 1, since the queried results of q1 and q2 are very similar, we can directly use the queried result of q1 to satisfy q1. Thus, the users of q1 need to download only one more node (in this case, node b in the XML file instance). Actually, the queried results of q1 and q3 can be combined into one since they are quite similar. In this example, the optimal total access time of the snippet scheme is 16 nodes (2 nodes waiting time for q2, 5 nodes waiting time for q3, 9 nodes waiting time for q3), while in the combining scheme, the optimal total access time is 14 nodes (2 nodes waiting time for q2, 6 nodes waiting time for q3, 7 nodes waiting time for all three queries). Moreover, the optimal total access time for the atomic scheme is 15 nodes (5 nodes waiting time for all three queries). Therefore, the combining scheme can achieve better access efficiency when compared with the two naive schemes. Moreover, it causes only a little tuning time overhead to the optimal result in theory (the total tuning time is 10 nodes, only 1 more node than that of the snippet scheme, which is the optimal one). Hence, it can provide much better balance between access time and tuning time (See the below Table 1, the numbers in italic are the best values among three schemes).
Table 1: Comparison of three schemes for Figure 1 in terms of XML nodes

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Access Time</th>
<th>Tuning Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>atomic scheme</td>
<td>15</td>
<td>12</td>
</tr>
<tr>
<td>snippet scheme</td>
<td>16</td>
<td>9</td>
</tr>
<tr>
<td>combining scheme</td>
<td>14</td>
<td>10</td>
</tr>
</tbody>
</table>

In the combining scheme, we need to define a parameter $sim$ (short for similarity) to measure the similarity of two XML snippets. Suppose that there are two XML snippets with lengths of $L_1$ and $L_2$ respectively, and the length of the combined one is $L_3$. Clearly, we have $\min(L_1, L_2) \leq \max(L_1, L_2) \leq L_3$.

**Definition 1.** The similarity of two XML snippets $sim$ is defined as follows:

$$sim = \frac{L_3}{\min(L_1, L_2)}$$

Obviously, $sim$ always satisfies $sim \geq 1$ and $sim \geq \frac{\max(L_1, L_2)}{\min(L_1, L_2)}$. A smaller $sim$ indicates more similarity between the two XML snippets before combined while a larger $sim$ indicates less similarity. If $sim = 1$, then the two XML snippets are exactly the same. Therefore, if we suppose to combine only XML snippets that are very similar to each other, we just need to combine snippets that have small $sim$ values. In Section 5 we will show that combining XML snippets with $sim$ values equal to or smaller than 1.4 can achieve the best overall performance.

Note that, in the combining scheme, the combined result is always part of the original XML file instance, which means the local structured information will remain unchanged. In the example in Figure 1, when combining the two similar XML snippets, we just use queries $q_1$ and $q_3$ together to match against the XML file instance and then we can get the combined result of these two snippets. However, the so-called combining stage is not so straightforward.

We do not have to generate separate queried results of users’ queries first and then combine those results. But actually we just adopt some XML data filtering techniques to find the combined result efficiently. By using the XML data filtering techniques (Vagena et al. 2007, Diao et al. 2003), we can determine which parts of a specific XML file match some queries efficiently. Then we can calculate the $sim$ after the filtering has finished by just identifying which part of the filtering result is matched the queries. After that, take $q_1$ and $q_3$ for instance, if the $sim$ value indicates that the queried results of $q_1$ and $q_3$ should be combined, then the filtering result of $q_1$ and $q_3$ is directly treated as a new XML snippet. We should continue the calculations with other queried results until no $sim$ value of any two XML snippets is equal to or smaller than the specified threshold of $sim$ value, such as 1.4 (in this case, we can firstly exclude XML snippets with $sim \geq \frac{\max(L_1, L_2)}{\min(L_1, L_2)} > 1.4$ and then find possible combining results).

At the client side, the clients just need to download the XML data they are interested in. However, as mentioned before, in the combining scheme, since the XML data is made up of combined snippets, the clients may download XML data that contains other information they are not interested in but required by other clients. This is due to the combination of different queried results in a snippet but better balance between access efficiency and power conservation can be expected.

4 Scheduling XML Data

At this stage, the scheduling problem of on-demand XML data broadcast is quite similar to the scheduling problem in multi-item data broadcast. The scheduling problem of multi-item data broadcast is known as a NP-Complete problem (Chung & Kim 1999). Up to now, a large body of studies has been done to solve the scheduling problem of multi-item data broadcast. However, previous work mainly considers access frequencies of data items and different queried result sets containing multiple items. In XML data broadcast, the data items have a variety of lengths and thus scheduling algorithms should take it into account. Moreover, existing scheduling algorithms suggest that the server only broadcast part of the queried results of the pending queries in one broadcast cycle (or one schedule) in order to achieve better access efficiency. This idea is under the premise that the data items are all atomic and each data item may be required by different queries. However, in on-demand XML data broadcast, data items are no longer atomic and thus each broadcast cycle aims to satisfy all pending queries that were issued during the previous broadcast cycle period. Based on this observation, we analyze the new model of scheduling XML data in on-demand broadcast and then put forward an improved scheduling algorithm in the following.

4.1 Analysis of Scheduling On-Demand XML Data Broadcasts

Consider the broadcast scenario of on-demand XML data broadcasts in Figure 2. At time $t_0$ and $t_1$, queries $q_i$ and $q_j$ arrive respectively. Then a new broadcast cycle begins at time $t_2$. This cycle aims to satisfy all the queries that were issued during the last broadcast cycle period, such as queries $q_i$, $q_j$, and so on. In this cycle, the queries $q_i$ and $q_j$ will be satisfied at time $t_i$ and $t_j$ accordingly. The cycle finishes at time $t_3$ in the end. For queries $q_i$ and $q_j$, the total access time ($AT$) can be calculated as follows:

$$AT = (t_i - t_0) + (t_j - t_1)$$

$$= (t_i - t_2 + t_2 - t_0) + (t_j - t_2 + t_2 - t_1)$$

$$= (t_i - t_2 + t_j - t_2) + (t_2 - t_0 + t_2 - t_1)$$

![Figure 2: A scenario of on-demand XML data broadcasts](image)

From the above calculation, we can see that the only variable factor that affects $AT$ is the total wait time starting from the beginning the cycle, that is $(t_1 - t_2 + t_j - t_2)$. As a result, in on-demand XML data broadcasts, the server does not need to consider the arrival time of each query when scheduling XML data on wireless channel but only need to minimize the total query wait time from the beginning of the current broadcast cycle, and in the above case, that is $(t_1 - t_2 + t_j - t_2)$ for both queries $q_i$ and $q_j$.

4.2 LEL (Least Efficient Last) Scheduling Algorithm

LEL scheduling algorithm (Sun et al. 2008) shows the best performance when compared with other existing
multi-item scheduling algorithms. It mainly considers access frequencies of data items and different queried result sets containing multiple items. However, in on-demand XML data broadcasts, the sizes of data items can vary in a very wide range. Thus data item size should be taken into consideration. In this subsection, we devise an improved scheduling algorithm using similar strategy in LRL algorithm.

First of all, we introduce some notations which will be used in the rest of the paper:
- \( d_i \): a data item (XML file) stored in the server
- \( D \): the set of data items that will be broadcast. \( D = d_1, d_2, \ldots, d_m \)
- \( L_i \): the length of data item \( d_i \)
- \( q_i \): a query issued by one or more mobile users
- \( Q \): the query set \( Q = q_1, q_2, \ldots, q_n \) that were issued during the last broadcast cycle
- \( QS(d_i) \): the query set in which all queries require data item \( d_i \)
- \( freq(q_i) \): the access frequency of \( q_i \)
- \( \sigma \): the broadcast schedule of a broadcast cycle
- \( FQS(d_i) \): given a schedule \( \sigma \), the query set that will be fully satisfied by item \( d_i \)

For a given schedule \( \sigma \) and a given query set \( Q \), it is easier to identify what queries will be fully satisfied when a data item is broadcast if examining from the last item to the first item in schedule \( \sigma \). For example, if \( d_m \) is the last item of schedule \( \sigma \) and is required by \( q_1, q_2 \) and \( q_3 \), then all these three queries will only be fully satisfied after they receive \( d_m \). Then after removing \( d_m \) from \( \sigma \) and removing \( q_1, q_2 \) and \( q_3 \) from \( Q \), we perform the same check on the updated \( \sigma \) and \( Q \).

Taking similar steps, we can work out what queries (other than \( q_1, q_2 \) and \( q_3 \)) will be fully satisfied by the new last item of \( \sigma \). By doing this repeatedly, the access time of all queries could be easily figured out. Moreover, since the last broadcast item produces the longest access time, it is reasonable that the last item should have the least access frequency. Furthermore, taking the lengths of data items into account, we propose to broadcast the least efficient data item as the last item in a schedule.

**Definition 2.** The efficiency of a data item \( d_i \) can be defined as follows:

\[
Eff(d_i) = \frac{\sum_{q \in FQS(d_i)} freq(q)}{L_i}
\]

Based on this definition, when we choose a new broadcast cycle, we first examine the efficiency of each data item and then select out the less efficient items to broadcast later and those items with higher efficiency will be broadcast earlier. The LEL (Least Efficient Last) scheduling algorithm can be described in the following.

Note that, in order to calculate \( Eff(d_i) \) for the first scheduled data item \( d_i \), we initially set all \( FQS(d) \) the same as \( QS(d) \) for every item \( d \) in \( D \). Moreover, suppose the data item set \( D \) contains \( m \) items, then step 1 and step 4 both take \( O(m) \) time and step 1 and step 4 will both repeat \( m \) times. Therefore, the computing complexity of LEL scheduling algorithm is \( O(m^3) \), which is the same as LRL (Sun et al. 2008).

### LEL Scheduling Algorithm:

1. select an item \( d \) from data item set \( D \) which has the smallest \( Eff(d) \)
2. place item \( d \) in the last vacant position of broadcast schedule \( \sigma \)
3. remove item \( d \) from \( D \)
4. update \( FQS(d') \) for every item \( d' \) in \( D \)
5. repeat step 1 to step 4 until \( D \) becomes empty

LRL algorithm has the property that exchanging the broadcast order of any two successive data items in a schedule \( \sigma \) (generated by LRL algorithm) will not achieve better overall access efficiency (Sun et al. 2008). However, LEL algorithm does not guarantee to hold this property.

In order to analyse the similar property of LEL algorithm, according to the finding in subsection 4.1, we only need to consider the wait time starting from the beginning of current broadcast cycle. Suppose two successive data items \( d_i \) and \( d_{i+1} \) in \( \sigma \) (generated by LEL algorithm) will be exchanged, then only the wait time of queries that are fully satisfied by data items \( d_i \) and \( d_{i+1} \) will be affected. Suppose the total length of data items broadcast before \( d_i \) is \( L' \), then the total wait time \( T \) of those fully satisfied queries (just because of receiving data items \( d_i \) and \( d_{i+1} \)) is

\[
T = \sum_{q \in FQS(d_i)} freq(q) \times (L' + L_i) + \sum_{q \in FQS(d_{i+1})} freq(q) \times (L' + L_i + L_{i+1}) \tag{1}
\]

After data items \( d_i \) and \( d_{i+1} \) are exchanged, the total wait time \( T' \) of those fully satisfied queries for the new schedule \( \sigma' \) is

\[
T' = \sum_{q \in FQS'(d_{i+1})} freq(q) \times (L' + L_{i+1}) + \sum_{q \in FQS'(d_i)} freq(q) \times (L' + L_i + L_{i+1})
\]

Since data items \( d_i \) and \( d_{i+1} \) are successive, we can infer that after these two items are exchanged, the whole set of fully satisfied queries of these two items remains unchanged. Thus we have

\[
FQS(d_i) \cup FQS(d_{i+1}) = FQS'(d_i) \cup FQS'(d_{i+1}) \tag{2}
\]

According to the definition of \( FQS \), we also have

\[
FQS(d_i) \cap FQS(d_{i+1}) = \emptyset \quad FQS'(d_i) \cap FQS'(d_{i+1}) = \emptyset
\]

Then we have

\[
T - T' = \sum_{q \in FQS'(d_{i+1})} freq(q) \times L_i - \sum_{q \in FQS(d_i)} freq(q) \times L_{i+1} \tag{3}
\]

Therefore, if we have \( T - T' \leq 0 \), then LEL algorithm holds the similar property of LRL algorithm; if
$T - T' > 0$, we call it a violation of the property. In other words, violation means exchanging these two successive data items can result in better access efficiency. However, detecting all these violations in a schedule generated by LEL algorithm could help to improve the overall access efficiency very little. This is because, according to LEL algorithm, when scheduling data item $d_{i+1}$, we must have

$$Eff'(d_{i+1}) \leq Eff(d_i) \leq Eff'(d_i)$$ (4)

Then according to (3) and the definition of $Eff$, if $T - T' > 0$, we must have

$$Eff(d_i) \leq Eff'(d_{i+1})$$

Then we must also have $L_i \leq L_{i+1}$, otherwise if $L_i > L_{i+1}$, we have

$$\sum_{q \in FQS(d_i) \cup FQS'(d_{i+1})} freq(q)$$

$$= Eff'(d_i) \times L_i + Eff'(d_{i+1}) \times L_{i+1}$$
$$= Eff(d_i) \times L_{i+1} + Eff(d_{i+1}) \times L_i + [Eff(d_i) \times (L_i - L_{i+1}) - Eff'(d_{i+1}) \times (L_i - L_{i+1})]$$
$$< Eff'(d_i) \times L_{i+1} + Eff'(d_{i+1}) \times L_i$$
$$\leq Eff'(d_{i+1}) \times L_{i+1} + Eff'(d_{i+1}) \times L_i$$

This is impossible because we have (2). Therefore, we have $L_i \leq L_{i+1}$. Also, if a violation happens, we have $T - T' > 0$. Then according to (3), (4) and the two schedules $\sigma$ and $\sigma'$, we must have

$$T - T' \leq \sum_{q \in FQS'(d_{i+1})} freq(q) \times L_i$$
$$\leq \sum_{q \in FQS(d_i)} freq(q) \times L_{i+1}$$

Then according to (1),

$$\frac{T - T'}{T} \leq \frac{\sum_{q \in FQS'(d_{i+1})} freq(q) \times L_i}{T}$$
$$< \frac{1}{1 + \frac{\sum_{q \in FQS(d_i) \cup FQS'(d_{i+1})} freq(q) \times (L_i' + L_i)}{\sum_{q \in FQS'(d_{i+1})} freq(q) \times L_i}}$$

Since for most of i ($1 \leq i < m$) in schedule $\sigma$, we have

$$L_i' + L_i = L_1 + L_2 + \ldots + L_i \gg L_{i+1}$$

Therefore we have

$$\sum_{q \in FQS(d_i) \cup FQS'(d_{i+1})} freq(q) \times (L_i' + L_i) \gg \sum_{q \in FQS'(d_{i+1})} freq(q) \times L_{i+1}$$

Then we can infer that if $T - T' > 0$, or in other words, if a violation happens, in most cases we should have

$$\frac{T - T'}{T} \to 0$$

As a result, most violations (if exist) cannot help to effectively improve the access efficiency. We will present the experimental results of the analysis in the next section.

## 5 Experiments

In this section, we mainly study the performance of combining scheme described in Section 3 and the effectiveness of LEL scheduling algorithm in previous section.

In our experiments, synthetic XPath queries are generated using the generator developed by Diao et al. (2003). All queries are distinct. The maximum depth of XPath queries is 6. Experiments are run on a synthetic data set: News Industry Text Format (NITF) DTD, and 500 XML documents are generated. The average depth of all documents is about 6.

Table 2 shows the descriptions of three parameters are varied in the experiments: the threshold of $sim$ ($T_{sim}$), the number of queries ($N_q$), and the probability of * and // in each query’s step ($prob$).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Range</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{sim}$</td>
<td>1 to 16</td>
<td>1.4</td>
</tr>
<tr>
<td>$N_q$</td>
<td>100 to 500</td>
<td>300</td>
</tr>
<tr>
<td>$prob$</td>
<td>10% to 30%</td>
<td>30%</td>
</tr>
</tbody>
</table>

### 5.1 Performance of Combining Scheme

Figure 3 (a) presents the average tuning time of each query when varying $T_{sim}$. TTBC denotes the average tuning time before using combining scheme while TTAC denotes the average tuning time after. When $T_{sim}$ is 1, the average tuning time is minimized and is equal to that of snippet scheme. When $T_{sim}$ is small, the amount of redundant data incurred by combining scheme keeps small as well. Also from the figure we can see that when $T_{sim}$ is smaller or equal to 1.6, the average tuning time increases very slowly. After that it grows rapidly. This is because a larger $T_{sim}$ can result in more redundant data to download for each query. However, when $T_{sim}$ is greater than 4, the average tuning time becomes stable since the redundant data is very close to that of atomic scheme, which contains the most redundant data amount all three schemes.

Figure 3 (b) shows the total data on the wireless channel when varying $T_{sim}$. LBC denotes the total length of data on the wireless channel before using combining scheme while LAC denotes the total length of data on the wireless channel after. As shown in the figure, the total length of data drops sharply by using our combining scheme. When $T_{sim}$ is 1, the length is almost one third of original data using snippet scheme. This is because although each query is different, but the matched snippets or sub-trees of each XML file are still likely the same. When $T_{sim}$ continues to grow, the total length of data continues to drop. However, when $T_{sim}$ is greater than 4, the total length of data becomes stable since the combined results are very close to that of atomic scheme and thus most possible combinations of snippets have been performed.

We study the effectiveness of LEL scheduling algorithm by comparing it with LRL scheduling algorithm (Sun et al. 2008), since it show the best access efficiency for multi-item on-demand data broadcasts in
5.2 Performance of LEL Algorithm

Figure 4 (a) presents the comparison between LEL and LRL scheduling algorithms when varying $N_q$. Generally, the average access time of LEL algorithm is only about 50% of LRL algorithm. When $N_q$ increases, the average access time of both LEL and LRL algorithms increases slowly as well because the more queries were issued to the server, the more data would be required and thus could result in more access time.

Figure 4 (b) depicts the comparison between LEL and LRL scheduling algorithms when varying prob. The average access time of LEL algorithm is about 40% to 65% of LRL algorithm. When prob increases, the average access time of both LEL and LRL algorithms increases as well. The reason for this is similar to Figure 4 (a). When prob is larger, more data will be required and thus could result in more access time. Moreover, when prob is 10%, the average access time of LEL algorithm is about 40% of LRL algorithm and when prob is 50%, the average access time of LEL algorithm grows faster than LRL algorithm and is about 65% of it.

Table 3 shows the results of fixing violations (defined in subsection 4.2). From the table, we can see that the average number of violations is about 875. However, the average improvement is only about 0.3%. Therefore, detecting and fixing all violations in the schedule generated by LEL algorithm brings very limited improvements to the overall access efficiency. Also, the results in the below table match our analysis in subsection 4.2 very well.

---

2Note that, according to subsection 4.1, we only compare the average wait time starting from the beginning of the current broadcast cycle (we still denote it as average AT in the figure, since it will not affect our comparisons).
Table 3: Violations fixing results

<table>
<thead>
<tr>
<th>N</th>
<th>Average AT (LEL) - KB</th>
<th>Average AT (LEL-VA) - KB</th>
<th>Number of violations</th>
<th>Improvement (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>673.11</td>
<td>765.96</td>
<td>1596</td>
<td>0.09</td>
</tr>
<tr>
<td>200</td>
<td>724.27</td>
<td>841.61</td>
<td>736</td>
<td>0.33</td>
</tr>
<tr>
<td>300</td>
<td>770.7</td>
<td>844.19</td>
<td>1596</td>
<td>0.62</td>
</tr>
<tr>
<td>400</td>
<td>842.38</td>
<td>844.19</td>
<td>391</td>
<td>0.13</td>
</tr>
<tr>
<td>500</td>
<td>847.07</td>
<td>844.19</td>
<td>1305</td>
<td>0.34</td>
</tr>
</tbody>
</table>

6 Conclusions

In this paper, we propose to pre-process XML data before broadcast to enhance the overall performance of on-demand XML data broadcasts. Firstly, taking advantage of the structural characteristics of XML data, we discuss two naive schemes (atomic scheme and snippet scheme) on how to match XML data with mobile users’ queries and based on these two schemes, a more practical scheme named combining scheme is put forward. It further makes use of the similarity of different XML snippets. Moreover, we analyze the scheduling problem of on-demand XML data broadcasts and define the efficiency of a data item. Based on this definition, we then devise a Least Efficient Last (LEL) scheduling algorithm to effectively organize XML data on wireless channels. Computing complexity and theoretical analysis of LEL scheduling algorithm are also given.

In our experiments, in combining scheme, by splitting query results into XML snippets and combining similar ones, the average access time can be effectively reduced (by about 14%) with a little overhead (about 8%) to the optimal tuning time in theory. More importantly, if compared with atomic scheme which is commonly adopted in previous work, the tuning time can be reduced by up to 75%. Hence, combining scheme can provide much better balance between access efficiency and power efficiency. Furthermore, in our experiments, the average access time of LEL algorithm is only about 40% to 65% of LRL algorithm. Therefore, by using LEL scheduling algorithm, the access efficiency can be improved significantly. In addition, we also demonstrate both theoretically and experimentally that if we detect and fix all violations of any two successive data items in a schedule σ generated by LEL algorithm, the average improvement of the overall access efficiency is about 0.3%, which is quite limited.

References


PartSS: An Efficient Partition-based Filtering for Edit Distance Constraints

Zhixu Li  Laurianne Sitbon  Xiaofang Zhou

School of Information Technology & Electrical Engineering
The University of Queensland, QLD 4072 Australia
Email: {zhixulii, zxf}@itee.uq.edu.au, l.sitbon@uq.edu.au

Abstract

This paper introduces PartSS, a new partition-based filtering for tasks performing string comparisons under edit distance constraints. PartSS offers improvements over the state-of-the-art method NGPP with the implementation of a new partitioning scheme and also improves filtering abilities by exploiting theoretical results on shifting and scaling ranges, thus accelerating the rate of calculating edit distance between strings. PartSS filtering has been implemented within two major tasks of data integration: similarity join and approximate membership extraction under edit distance constraints. The evaluation on an extensive range of real-world datasets demonstrates major gain in efficiency over NGPP and QGrams approaches.

Keywords: edit distance, partition-based, similarity join, approximate membership extraction

1 Introduction

Nowadays, with the fast growing of data sources and the inconsistency between them, our demands for approximate data integration also increase sharply. Two key tasks in data integration are: a) similarity join between data sources to reconcile different representations of the same entities and b) approximate membership extraction, which targets the identification of the substrings from an unstructured text which approximately matches with any member of a given large dictionary of known entities. The entities in the two tasks can be gene or protein names (bioinformatics), geographical locations (geotagging), person or product’s names (business information system) etc. Due to their importance, similarity join and approximate membership extraction have been studied in many areas such as data cleaning, information integration and bioinformatics.

These two tasks rely on string similarity functions. According to the requirements of different applications, various string similarity functions have been considered. Some applications may prefer to ignore the order of tokens appearing in the text, such as recognizing plagiarism which share almost the same word set as the original one, or identifying junk mails where some commercial keywords are frequently used. Therefore, token-based similarity functions such as jaccard similarity, cosine similarity etc. are to be used properly (Sarawagi and Kirpal 2004, S. Chaudhuri and Kaushik 2006, Arasu et al. 2006, Bayardo et al. 2007). Other applications are sensitive to token orders, such as retrieving results for typo keywords in information retrieval or recognizing homologous DNA strands in bioinformatics. In these situations, edit distance tends to be a good choice, as it measures the minimum number of edit operations, including insertion, deletion and substitution, to transform one string into another. Several works have used edit distance in similarity join and approximate membership extraction (Zobel and Dart 1995, Bilenko et al. 2003, Gravano et al. 2001, Wang et al. 2009).

Applying edit distance raises great efficiency challenges in Edit Distance similarity Join (EDJ) or Approximate Membership Extraction with Edit distance threshold (ED-AME) since its computation is expensive\(O(n^2)\) time complexity in standard dynamic programming (Wagner and Fischer 1974)). For EDJ between two sets of strings, a naive pairwise similarity join costs \(O(N(N^2 + n^2))\) running time, where \(N\) is the number of strings in each set and \(n\) is the average length of the strings. For ED-AME from a document \(D\) w.r.t. a dictionary of string entities \(R\), the naive enumeration method costs \(O(|D| \cdot |R| \cdot n^2)\) in running time.

To tackle the low efficiency problem brought by computing edit distance, several approaches have been proposed that raise the efficiency by pruning unmatched pairs of strings with filtering techniques. A widely adopted filter is the conversion of the edit distance constraint between two strings into the overlap between their \(q\)-grams sets (Gravano et al. 2001). The main inconvenient of this approach is the sensitivity to the choice of the size of \(q\)-grams. Another line of works are developing filtration based on finding common neighbors between strings. Bocek et al. proposed to generate \(\tau\)-deletion neighborhood sets by deleting no more than \(\tau\) characters at all possible positions of a string, where \(\tau\) is the given edit distance threshold. If two strings are within \(\tau\) edit distance to each other, they must share at least one \(\tau\)-deletion neighborhood. This filtering may only work well for short strings with a small edit distance threshold \(\tau\), otherwise the size of deletion neighborhood \(O(|s|^{\tau+1})\) can be large, where \(|s|\) denotes the length of the string \(s\). To further reduce the neighborhood size, an improved method called NGPP (stands for Neighborhood Generation with partitioning and prefix-based pruning) was proposed by Wang et al. (Wang et al. 2009). NGPP first partitions a string \(s\) into several partitions, then slightly adjusts the position and length of these partitions to generate variants for each partition. It is guaranteed that for any string \(s'\) within \(\tau\) edit distance to \(s\) there must be at least one partition of \(s'\) within 1 edit distance to a variant of the partition which lies at the same position in \(s\). Only 1-deletion neighborhoods are generated for each partition variant of \(s\) and the overall neighborhood size of \(s\) is \(O(|s|\tau + \tau^2)\).

NGPP greatly decreases the time and space complexity of EDJ or ED-AME, but it still need to generate 1-deletion neighborhoods and the neighborhood size also becomes large when \(|s|\) is a long string. In this paper,
we propose a novel partition-based filtering technique to be implemented with EDJ and ED-AME. The new partition scheme is \texttt{PlusOne} and partition variation generation rule are proposed, leading to the generation of $O(\tau^2)$ partition variants for string $s$, such that for any string $s'$ that within $\tau$ edit distance to $s$, there must at least one partition of $s'$ equals to a variant of the partition which lies at the same position of $s$. In order to prevent the generation of unnecessary partition variants, we also develop some non-trivial techniques to give a stricter variation generation rule for our method. By applying the prefix-based filtering (Wang et al. 2009), the size of partition signatures can be further reduced to $O(l_p \cdot \tau)$, where $l_p$ is the length of prefix set in the prefix-based pruning.

Our main contributions in this paper are summarized below:

- We propose a novel partition-based filter for EDJ and ED-AME using a new partition scheme \texttt{PlusOne} reducing the space complexity to $O(\tau^2)$ in addition to reducing the probability of false-positives.
- We develop a strict partition variant generation rule, which gives tighter upper-bounds to the range of shifting and scaling operations applied to partitions of the string.
- We apply this filter into both EDJ and ED-AME demonstrate the effectiveness of the filtration based on our partition-based scheme on several real-world datasets.

The outline of this paper is as follows: Section 2 covers related work. We introduce preliminaries in Section 3. Section 4 presents the novel partition-based signature scheme for edit distance. We then introduce how we adapt our scheme into EDJ and ED-AME in Section 5 and 6. The experimental study is given in Section 7.

2 Related Work

Edit distance is a widely-used distance function for strings, which can be computed in $O(n^2)$ time and $O(n)$ space using the standard dynamic programming (Wagner and Fischer 1974). In order to enhance the computing efficiency, some techniques have been proposed in the past several decades, such as the Four-Russians (Masek and Paterson 1980) which improves the time complexity into $O(n^2/\log(n))$, or a bit-parallel algorithm (Ukkonen 1983) which reaches an average time complexity of $O(n + d^2)$ (d is the edit distance between two strings).

Recently, edit distance was introduced into the Similarity Join (SJ) and Approximate Membership Extraction (AME) (Zobel and Dart 1995, Bilenko et al. 2003, Gravano et al. 2001, Wang et al. 2009), making efficiency problems arise. The state-of-art approaches to process edit similarity join or edit approximate membership extraction are mainly based on a popular filtration-verification framework. In the filtration step, a great amount of string pairs (or substring and entity pairs) are pruned and the similarity of the remaining pairs is calculated in the verification step.

A popular filtering scheme is to convert the edit distance constraint into a weaker count filtering on the number of matching q-grams. Q-grams are generated by sliding a window of width $q$ over a string. If two strings $s$ and $t$ are within edit distance $\tau$, they must share at least $LB_{s,t} = \max\{|s|, |t|\} - q + 1 - q \cdot \tau$ q-grams (Gravano et al. 2001). Other methods were also suggested to complement the count filtering. length filtering guarantees that $s$ and $t$’s length disparity should be within $\tau$. Position filtering requires that there are at least $LB_{s,t}$ matching positional q-grams (Gravano et al. 2001). Since generating all pairs of strings satisfying the count filtering rule is a bottleneck, the prefix filtering strategy was used later to quickly discard some unmatched pairs without accessing all of their q-grams (S. Chaudhuri and Kaushik 2006). As state-of-the-art techniques, Xiao et al. proposed location-based mismatch filtering and content-based mismatch filtering, which can be incorporated into previous strategies to make the edit similarity join be more efficient (Xiao et al. 2008). However, it is a dilemma to choose proper length of q-grams when using q-gram-based approaches to find approximate matches. On the one hand, $q$ should be smaller than $\frac{L_{min} + 1}{\tau}$ ($L_{min}$ is the shortest length of entity string) to guarantee at least one common q-gram between matching strings. On the other hand, short q-grams lead to long posting lists impacting the overall cost of similarity join (Xiao et al. 2008, Zobel and Dart 1995). Besides, q-gram-based approaches are less selective for short entities, and it is hard to share computation (Wang et al. 2009).

Both SJ and AME have been studied extensively in the literature. SJ is related to record linkage (Winkler 1999), name matching (Bilenko et al. 2003), data deduplication (Sarawagi and Bhamidipaty 2002) etc. AME is also known as Membership Checking or Approximate Entity Extraction. Besides edit distance, various alternative similarity functions have been used in SJ and AME. These works are also mainly based on the filtration-verification framework. In the filtration step, various kinds of signature schemes (S. Chaudhuri and Kaushik 2006, Chandel et al. 2006, Arasu et al. 2006, Gionis et al. 1999), or inverted list index (Chandel et al. 2006, Singhhal 2001), or combinations (Chakrabarti et al. 2008, Lu et al. 2009, Wang et al. 2009) have been proposed.

Also related field of SJ and AME, “approximate string matching” focuses on finding a pattern string approximately in a text. This problem has been studied extensively in algorithmic, information retrieval and natural language process communities. An excellent survey is given in (Navarro 2001).

3 Preliminaries

Our proposed method builds on previous filtering approaches that we briefly review in this section.

3.1 Neighborhood Generation-based filtering

A series of approaches based on neighborhood generation has been developed to deal with edit distance constraints. All the strings that are at most at $\tau$ edit distance away from a string $s$ constitute the $\tau$ neighborhood of $s$, formally represented as $U_\tau(s) = \{ s' | ed(s, s') \leq \tau \}$, where $ed(s, s')$ is the edit distance between $s$ and $s'$. Therefore, if string $t$ can approximate match with $s$, it must be contained in the $\tau$ neighborhood of $s$. However, since the size of the neighborhood $U_\tau(s)$ is $O(|s|^\tau \cdot \sum |\tau|)$ in practice, it is impossible to generate and use all the neighborhood directly.

The deletion neighborhood was proposed as a complement (Bocek et al. 2007). The $\tau$-deletion neighborhood of a string $s$ consists of all deletion variants of $s$, each of which is generated by deleting no more than $\tau$ characters at all possible positions of $s$. If another string $t$ is within the edit distance $\tau$ of $s$, then they share at least one common variant in their $\tau$ deletion neighborhood. Based on the property of deletion neighborhood, the FastSS algorithm proposed in (Bocek et al. 2007) successfully reduces the size of the neighborhood to $O(|s|^\tau)$. However this algorithm is suitable for short strings with a small edit distance threshold $\tau$. 

3.2 NGPP

The improved neighborhood generation-based method NGPP has been proposed (Wang et al. 2009) to further reduce the size of the deletion neighborhood to $O(l_p \cdot \tau^2)$, where $l_p$ is the length of the prefix set in the prefix-based pruning. Now we briefly introduce NGPP.

The partition scheme defined in NGPP partitions a string into $k_T = \lceil \frac{\tau + 1}{m} \rceil$ partitions, with the first $k_T - 1$ partitions of length $\lceil \frac{m}{k_T} \rceil$, and the $k_T$ (the last) partition taking the rest of the string, where $m$ is the length of the string. For a string $s$ and any of its $\tau$-neighborhood string $s'$ (a string within $\tau$ edit distance of $s$), at least one partition of $s'$, subject to appropriate amount of shifting and scaling operation, is within edit distance 1 to the partition at the same position of $s'$. The shifting and scaling operations are defined to generate variations for partitions of $s$.

- **Shifting** by $a$ moves the partition $s[i..j]$ by $a$ positions to $s[(i + a) .. (j + a)]$.
- **Scaling** by $b$ changes the length of $s[i..j]$ by $b$ to $s[i...(j + b)]$.

Table 1 presents the constraints on the shifting and scaling ranges for each partition of string $s$, according to the possible length and position dispatch between $s$ and its $\tau$-neighborhood string.

<table>
<thead>
<tr>
<th>Partition</th>
<th>Shifting Range</th>
<th>Scaling Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>the first one</td>
<td>$[0]$</td>
<td>$[-2, 2]$</td>
</tr>
<tr>
<td>the last one</td>
<td>$[-\tau, \tau]$</td>
<td>equals to shifting amount</td>
</tr>
<tr>
<td>others</td>
<td>$[-\tau, \tau]$</td>
<td>$[-2, 2]$</td>
</tr>
</tbody>
</table>

Once partition variants are generated for a string $s$, 1-deletion neighborhood variants are generated for each partition variant. Thus $O(\tau \cdot |s| + \tau^2)$ deletion neighborhood variants are generated for $s$. In order to measure if another string $s'$ is a $\tau$-neighborhood string of $s$, all partition variations (at least one) are generated for $s'$.

**Example 1** Given a string $s$ = "Leonardo DiCaprio" and a misspell string $t$ = "Leonardo Diecbrio". When $\tau = 3$, we have $k_T = 2$. According to the partition scheme, $s$ and $s'$ can be partitioned as follows, where we use # to represent the blank between words:

$s$ = [{Leonardo}, {#DiCaprio}]
$s'$ = [{Leonardo#}, {Diecbri#}]


The 1-deletion variants for all partitions of $s$ are:

- [{Leonardo}, 1] and [{Leonard}, 1]
- [{Leonardo}, 1] and [{Leonardo#}, 1]
- [{Leonardo#D}, 1] and [{rdo#DiCaprio}, 2]
- [{do#Diacprio}, 2] and [{#DiCaprio}, 2]
- [{icaprio}, 2] and [{caprio}, 2]

Since the first partition of $s'$, [{Leonardo#}], shares a 1-deletion variant [{Leonardo#}] with $s$'s partition variation [{Leonardo#}], $s'$ is within $\tau$ edit distance from $s$.

When $s$ and $s'$ are long, not only space complexity increases to store the variations of $s$, but also time complexity in matching partition variants of $s'$ against those of $s$. In order to reduce these complexities, a prefix-based pruning was introduced. When a partition is longer than a prefix length $l_p$, only 1-deletion neighborhood variants are generated for the $l_p$-prefix of the neighborhood.

**Example 2** Continuing with Example 1, With $l_p = 3$, the following 1-deletion variants are generated from the prefixes:

\[
\begin{align*}
&\{\text{Leo}\}, 1 \\
&\{\text{rdo}\}, 2 \\
&\{\text{Do#}\}, 2 \\
&\{\#\text{Di}\}, 2 \\
&\{\text{Dic}\}, 2 \\
&\{\text{ica}\}, 2 \\
&\{\text{cap}\}, 2
\end{align*}
\]

For each string $s'$, a neighborhood of $O(l_p \cdot \tau^2)$ variants are generated and searched against all variants of $s$. The approach we propose in this paper is another pure partition-based approach without generating deletion variants, which only generates $O(|s| + \tau^2)$ partition variants. Combining with prefix-based method, the size of variants can be further reduced to $O(l_p \cdot \tau)$.

3.3 PartEnum

PartEnum is another partition-based approach proposed for edit distance constraints (Arasu et al. 2006). Based on the pigeon hole principle, it first divides a string into several partitions, then generates signatures for each partition using an enumeration scheme. This scheme guarantees that if two strings are matched within a given edit distance threshold, they must share at least one signature. It shows good performances for small edit distance threshold. However, since the performance of PartEnum is greatly dependent on the partitioning parameters, it is hard to set the parameters that work well for both short and long entities (Wang et al. 2009).

4 PartSS: A New Partition-Based Filtering

In this section, we introduce a new partition-based filter – PartSS, which stands for Partition-based filter with Shifting and Scaling operations. We present our partition scheme PlusOne firstly, then develop some non-trivial techniques to tighten the partition variants set.

4.1 Partitioning

The principle of PartSS is directly inspired by that of NGPP. By setting the partition number to $k$ disjoint partitions, according to the pigeon hole principle, there exists at least one partition which only need at most $\lceil \frac{1}{k} \rceil$ edit operations for changing $s$ into any other string $s'$ which edit distance to $s$ is within $\tau$. In NGPP $k_T = \lceil \frac{\tau + 1}{m} \rceil$. In order to avoid the generation of a deletion neighborhood, we choose to set $k_T = \tau + 1$, so that there exists at least one partition in $s$, after subjecting to some shifting and scaling, that directly equals to the partition in the same position of $s'$.

**PlusOne Partition Scheme:** Given an edit distance threshold $\tau$, PlusOne partition scheme partitions a string $s$ into $k_T = \tau + 1$ partitions, where the first $n_1 = k_T - |s| \% k_T$ partitions have length $l_1 = \lceil \frac{|s|}{k_T} \rceil$, and the other $n_2 = |s| \% k_T$ partitions have length $l_2 = \lceil \frac{|s|}{l_T} \rceil$, such that $n_1 \cdot l_1 + n_2 \cdot l_2 = |s|$. (As shown in Fig. 1)

![Figure 1: The Partitions of String s under PlusOne Scheme](image)

Due to the length and position deviation between corresponding partitions in $s$ and $s'$, we have to adjust the
length and position of partitions in $s$. Here we adopt the two operations (shifting and scaling) defined in NGPP as introduced in Section 3. Under PlusOne, the baseline rule on how to apply shifting and scaling operations to every partition of a string is given below and summarised in Table 2. This baseline rule is similar to the one used in NGPP (Wang et al. 2009).

**Rule 1** (Baseline Variants Generation Rule) Given the edit distance threshold $\tau$, a string $s$ is partitioned into $\tau+1$ partitions under the PlusOne partitioning scheme. Then for the $i$-th partition of $s$:

- if $i = 1$, we scale the partition within the range of $[-1,1]$ to generate its variants.
- if $i = \tau + 1$, we scale and shift the partition to the same amount within the range of $[-\tau,\tau]$ to generate its variants.
- if $1 < j < \tau$, we scale the partition within the range of $[-\tau,\tau]$, while also shifting it within the range of $[-2,2]$, to generate its variants.

<table>
<thead>
<tr>
<th>Table 2: Baseline Shifting and Scaling Rule for Each Partition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partition</td>
</tr>
<tr>
<td>-----------</td>
</tr>
<tr>
<td>the first one</td>
</tr>
<tr>
<td>the last one</td>
</tr>
<tr>
<td>others</td>
</tr>
</tbody>
</table>

Based on our proposed PlusOne partitioning scheme and partition variants generation rules, the partitions have the following properties:

**Property 1** (Partition Variants Property) Given a string $s$ and an edit distance threshold $\tau$, if we partition $s$ with PlusOne partitioning scheme and generate partition variants according to the baseline rule above, for any string $s'$ which satisfy $\text{l}(s,s') \leq \tau$, there must be at least one partition in a position of $s'$ (also partitioned with PlusOne) matching with a variant of the partition in the same position of $s$.

As mentioned in Section 3, NGPP generates $O(m \cdot \tau^2)$ variants for a string, whereas the baseline rule based on PlusOne reduces the size of variants to $O(\tau^2)$.

### 4.2 Tightening the Shifting and Scaling Ranges

In the baseline rule, the boundaries of the shifting and scaling ranges are set very loose to guarantee full coverage. On the other hand, loose boundaries also lead to more false-positives, since more unpromising matches will be considered as candidate matches. In order to give a minimum coverage of all $\tau$-neighborhoods, we expect to find a stricter rule to tighten the boundaries of shifting and scaling ranges, such that unnecessary variants of partitions are not generated. Meanwhile, this new rule should also satisfy Property 1.

For convenience of presentation, we differentiate the original shifting and scaling operations into four different operations: negative shifting, positive shifting, negative scaling and positive scaling. The shifting or scaling amount of any of the four operations is a non-negative numeric referred to as the variation amount of that operation.

In order to achieve minimum coverage, we give different upper-bounds to the variation amount of each operation applied to partitions at different positions. A straightforward approach is to enumerate all $\tau$-neighborhoods of the string $s$ for any neighborhood string $s'$, there must exists one partition of $s$ that requires the least variation amount of an operation to become the corresponding partition of $s'$. We call this partition as the key partition in transforming from $s$ to $s'$ (denoted $p_{\text{key}}^{\tau+\gamma}$). When there are more than one key partitions in a transformation, only the one at the first position is considered. To reach a minimum coverage, we deem that only the operations applied on key partition in each transformation are necessary. Accordingly, the following property on the tightest upper-bound of operations to each partition holds:

**Property 2** (Tightest upper-bound Property) Given a string $s$, the tightest upper-bound of an operation on a partition of $s$ is the maximum variation amount of the operation to that partition when it acts as a key partition.

Although it is impractical to enumerate all $\tau$-neighborhoods of $s$, they can be classified into 4 types according to their length and position deviation to $s$, and we can discuss the upper-bounds of the variation amount of the four operations in the four different transformation situations below:

**Four Transformation Situations**: Given the edit distance threshold $\tau$, assume we need $\alpha$ insertion, $\beta$ deletion and $\gamma$ substitution to transform $s$ into a $\tau$-neighborhood string $s'$ such that $\alpha + \beta + \gamma \leq \tau$. Since the insertion and deletion operations introduce position and length changes, all strings within $\tau$ edit distance to $s$ can be classified into one of the four situations depending on the value of $\alpha$ and $\beta$. Based on the PlusOne partition scheme, at most two kinds of partitions with different length are generated for each string. Here we assume the first $n_1$ partitions have length $l_1$, and the remaining $n_2$ partitions have length $l_2$ in $s$. According to the PlusOne scheme, $l_1 + 1 = l_2$, $n_1 + n_2 = \tau + 1$. We also define $l_0 = l_1 - 1$, $l_3 = l_2 + 1$, which are possible lengths of partitions in $\tau$-neighborhood strings as shown in Fig. 2.

- **Situation 1**: If $0 \leq \alpha - \beta \leq n_1$, then the first $n_1 - (\alpha - \beta)$ partitions have length $l_1$, and the other ones have length $l_2$.
- **Situation 2**: If $n_1 \leq \alpha - \beta \leq \tau$, then the first $n_1 + (\tau - \alpha + \beta + 1)$ partitions have length $l_2$, and the other ones have length $l_3$.
- **Situation 3**: If $0 \leq \beta - \alpha \leq n_2$, then the first $n_1 + (\beta - \alpha)$ partitions have length $l_1$, and the other ones have length $l_2$.
- **Situation 4**: If $n_2 \leq \beta - \alpha \leq \tau$, then the first $n_1 + (\tau + \alpha - \beta + 1)$ partitions have length $l_0$, and the other ones have length $l_1$.

In the remainder of this section, we use $\sigma_{\pm}(i)$ and $\sigma_{\pm}(i)$ to denote the variation amount of the positive and negative shifting operations and $\omega_{\pm}(i)$ and $\omega_{\pm}(i)$ to denote the variation amount of positive and negative scaling operations that have to be applied to the $i$-th partition. We also use $\overline{\sigma}(i)$, $\overline{\sigma}(i)$, $\overline{\omega}(i)$, $\overline{\omega}(i)$ to denote the upper-bounds of their variation amounts. It is worth noting that: $\overline{\sigma}(i)$ only needs to be discussed in situation 1 and 2, while $\overline{\omega}(i)$ only needs to be discussed in situation 3 and 4. We only provide the detailed deduction process in situation 1 as the deduction process for the other three situations follows the same principle.

**Situation 1**: $0 \leq \alpha - \beta \leq n_1$, since $\alpha + \beta \leq \tau$, we have: $0 \leq \alpha \leq \frac{\tau + n_1}{2\tau}$. According to the difference between partition lengths at the same position of $s$ and $s'$, the partition set can be divided into three disjoint consecutive parts. We use $p_1$ and $p_2$ to denote the last partition...
position within the first and second parts respectively such that: \(0 \leq p_1 = n_1 - (\alpha - \beta) \leq n_1 - p_2 = n_1\).

Positive scaling \(\omega_+(i)\): We can easily observe the length variation for partitions of three different parts in Figure 2. Equation 1 gives the value of \(\omega_+(i)\) for the \(i\)-th partition.

\[
\omega_+(i) = \begin{cases} 
0 & (1 \leq i < p_1) \\
1 & (p_1 + 1 \leq i \leq n_1) \\
0 & (n_1 + 1 \leq i \leq \tau + 1) 
\end{cases}
\]

Positive shifting \(\sigma_+(i)\): Equation 2 gives the maximum value of \(\sigma_+(i)\) for the \(i\)-th partition, where \(\alpha_{\text{pre}}\) denotes the insertions applied to the partitions before the \(i\)-th one.

\[
\sigma_+(i) = \begin{cases} 
0 & (i = 1) \\
\alpha_{\text{pre}} + i - p_1 + 1 & (2 \leq i \leq p_1) \\
\alpha_{\text{pre}} + p_1 - n_1 & (n_1 + 1 \leq i \leq \tau + 1) 
\end{cases}
\]

Since the function given in Equation 2 is declining, the condition on the \(i\)-th partition acting as the key partition is that at least one edit operation should be applied to each partition after the \(i\)-th one (except for those whose variation amount of positive scaling equals to that of the \(i\)-th one). Given \(\alpha_{\text{pre}} \leq \alpha \leq \frac{\tau + n_1}{2}\), the maximum value of \(\alpha_{\text{pre}}\) is:

\[
\alpha_{\text{pre}} = \begin{cases} 
\min(p_1, \frac{\tau + n_1}{2}) & (2 \leq i \leq p_1) \\
\min(i - 1, \frac{\tau + n_1}{2}) & (p_1 + 1 \leq i \leq n_1) \\
\min(\tau, \frac{\tau + n_1}{2}) & (n_1 + 1 \leq i \leq \tau + 1) 
\end{cases}
\]

Considering the maximum value of \(\alpha_{\text{pre}}\), the maximum value of \(\sigma_+(i)\) is:

\[
\sigma_+(i) = \begin{cases} 
0 & (i = 1) \\
p_1 & (2 \leq i \leq p_1) \\
p_1 + 1 & (p_1 + 1 \leq i \leq n_1) \\
\frac{\tau + n_1}{2} & (n_1 + 1 \leq i \leq \tau + 1) 
\end{cases}
\]

Given the range of \(p_1\), the range of \(\sigma_+(i)\) or \(\sigma_+(i)\) and its corresponding value of \(\omega_+(i)\) is in Table 3 and 4.

Table 3: \(\sigma_+(i)\) and corresponding \(\omega_+(i)\) 1

<table>
<thead>
<tr>
<th>(i)-th Partition</th>
<th>(\sigma_+(i))</th>
<th>(\omega_+(i))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i = 1)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2 \leq i \leq n_1</td>
<td>0, (i)</td>
<td>0</td>
</tr>
<tr>
<td>1 \leq i \leq \tau</td>
<td>0, (i - 1)</td>
<td>1</td>
</tr>
<tr>
<td>(n_1 + 1 \leq i \leq \tau + 1)</td>
<td>0, (\frac{\tau + n_1}{2})</td>
<td>0</td>
</tr>
</tbody>
</table>

Negative shifting \(\sigma_-(i)\): Equation 5 gives the maximum value of \(\sigma_-(i)\) for the \(i\)-th partition, where \(\beta_{\text{pre}}\) denotes the deletions applied to the partitions before the \(i\)-th one.

\[
\sigma_-(i) = \begin{cases} 
0 & (i = 1) \\
\beta_{\text{pre}} + i - p_1 - 1 & (2 \leq i \leq p_1) \\
\beta_{\text{pre}} - p_1 + n_1 & (n_1 + 1 \leq i \leq \tau + 1) 
\end{cases}
\]

Since the function given in Equation 5 is increasing, the condition on the \(i\)-th partition acting as the key partition is that at least one edit operation is applied to each partition before the \(i\)-th one, which do not give any constraint on the value of \(\beta_{\text{pre}}\). Given \(\beta_{\text{pre}} \leq p_1 - p_2 + \alpha \leq \frac{\tau - n_1}{2} + p_1\), the maximum value of \(\beta_{\text{pre}}\) is \(\frac{\tau - n_1}{2} + p_1\).

Considering the maximum value of \(\beta_{\text{pre}}\), the maximum value of \(\sigma_-(i)\) is:

\[
\sigma_-(i) = \begin{cases} 
0 & (i = 1) \\
\frac{\tau - n_1}{2} + p_1 & (2 \leq i \leq p_1) \\
\frac{\tau - n_1}{2} + i - 1 & (p_1 + 1 \leq i \leq n_1) \\
(n_1 + 1 \leq i \leq \tau + 1) 
\end{cases}
\]

Given the range of \(p_1\), the range of \(\sigma_-(i)\) or \(\sigma_-(i)\) and its corresponding value of \(\omega_-(i)\) is in Table 5 and 6.

Table 5: \(\sigma_-(i)\) and corresponding \(\omega_-(i)\) 1

<table>
<thead>
<tr>
<th>(i)-th Partition</th>
<th>(\sigma_-(i))</th>
<th>(\omega_-(i))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i = 1)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2 \leq i \leq n_1</td>
<td>0, (\frac{\tau - n_1}{2} + i - 1)</td>
<td>0</td>
</tr>
<tr>
<td>1 \leq i \leq \tau</td>
<td>0, (\frac{\tau - n_1}{2} + i - 1)</td>
<td>1</td>
</tr>
<tr>
<td>(n_1 + 1 \leq i \leq \tau + 1)</td>
<td>0, (\frac{\tau - n_1}{2} + i - 1)</td>
<td>0</td>
</tr>
</tbody>
</table>

Summary: Following the same analysis to generate cor-respondences values for situations 2, 3 and 4, the overall tight partition variants generation rules are deduced as presented in Table 7-10. The four tables give different upper-bounds to the variation amount of each operation applied to partitions at different positions.
Table 6: \( \sigma_-(i) \) and corresponding \( \omega_+(i) \) in situation 1

<table>
<thead>
<tr>
<th>( i )-th Partition</th>
<th>( \sigma_- (i) )</th>
<th>( \omega_+ (i) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( i = 1 )</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( 2 \leq i \leq n_1 )</td>
<td>( [0, \tau - n_1 + i - 1] ) ( \frac{\tau}{2} )</td>
<td>1</td>
</tr>
<tr>
<td>( n_1 + 1 \leq i \leq \tau + 1 )</td>
<td>( [0, \tau - n_1 + 1] ) ( \frac{\tau}{2} )</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 7: \( \sigma_+ (i) \) and corresponding \( \omega_-(i) \)

<table>
<thead>
<tr>
<th>( i )-th Partition</th>
<th>( \sigma_+ (i) )</th>
<th>( \omega_- (i) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( i = 1 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( 2 \leq i \leq n_1 )</td>
<td>( [0, i - 1] ) ( \frac{i}{n_1} )</td>
<td>1</td>
</tr>
<tr>
<td>( i = n_1 + 1 )</td>
<td>( 0, \frac{\tau - n_1}{2} ) ( \frac{\tau}{2} )</td>
<td>0</td>
</tr>
<tr>
<td>( n_1 + 2 \leq i \leq \tau + 1 )</td>
<td>( [0, i - n_1 + 1] ) ( \frac{i}{n_1 - 1} ) ( \frac{\tau}{2} )</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 8: \( \sigma_+ (i) \) and corresponding \( \omega_-(i) \)

<table>
<thead>
<tr>
<th>( i )-th Partition</th>
<th>( \sigma_+ (i) )</th>
<th>( \omega_- (i) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( i = 1 )</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( 2 \leq i \leq n_1 )</td>
<td>( [0, \tau - n_1 + 1] ) ( \frac{\tau}{2} )</td>
<td>1</td>
</tr>
<tr>
<td>( i = n_1 + 1 )</td>
<td>( 0, \frac{\tau}{2} ) ( \frac{\tau}{2} )</td>
<td>0</td>
</tr>
<tr>
<td>( n_1 + 2 \leq i \leq \tau + 1 )</td>
<td>( [0, i - 2] ) ( \frac{\tau}{2} )</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 9: \( \sigma_-(i) \) and corresponding \( \omega_+(i) \)

<table>
<thead>
<tr>
<th>( i )-th Partition</th>
<th>( \sigma_- (i) )</th>
<th>( \omega_+ (i) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( i = 1 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( 2 \leq i \leq n_1 )</td>
<td>( [\tau - n_1 + 1] ) ( \frac{\tau}{2} )</td>
<td>1</td>
</tr>
<tr>
<td>( i = n_1 + 1 )</td>
<td>( 0, \tau ) ( \frac{\tau}{2} )</td>
<td>0</td>
</tr>
<tr>
<td>( n_1 + 2 \leq i \leq \tau + 1 )</td>
<td>( [0, i - 2] ) ( \frac{\tau}{2} )</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 10: \( \sigma_-(i) \) and corresponding \( \omega_-(i) \)

<table>
<thead>
<tr>
<th>( i )-th Partition</th>
<th>( \sigma_- (i) )</th>
<th>( \omega_- (i) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( i = 1 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( 2 \leq i \leq n_1 )</td>
<td>( [1, \min(n_1, \tau - n_1 + 1)] ) ( \frac{\tau}{2} )</td>
<td>1</td>
</tr>
<tr>
<td>( n_1 + 1 \leq i \leq \tau + 1 )</td>
<td>( [0, n_1] ) ( \frac{n_1}{2} ) ( \frac{\tau}{2} )</td>
<td>1</td>
</tr>
<tr>
<td>( n_1 + 2 \leq i \leq \tau + 1 )</td>
<td>( [n_1, \tau + n_1 + 3] i ) ( \frac{\tau}{2} )</td>
<td>0</td>
</tr>
</tbody>
</table>

Example 3 Based on the same string and setting as Example 1, when \( \tau = 3 \), \( k_o = 4 \), according to the partition scheme, \( s \) and \( t \) can be partitioned into:

\[
s = \{\text{Leon}_0, \text{ardo}, \#\text{Dic}, \text{aprio}\}
\]

\[
t = \{\text{Leon}_0, \text{ardo}, \#\text{Dic}, \text{aprio}\}
\]

The next step, shifting and scaling operations should be applied to the partitions of \( s \) to generate variants. Through calculation, we have the tightened shifting and scaling ranges for partitions of \( s \) as shown in Table 11.

The second partition of \( t \) is nardo, and can be found in the variants of the second partition of \( s \). Therefore, \( t \) passes the filtration and becomes a candidate string within the 3 edit distance to \( s \).

4.3 Combining with Prefix-based Filtering

The size of the variants set can be further reduced by adopting the prefix-based filtering used in (Wang et al. 2009). A fix length \( l_p \) prefix of each variant can reduce the number of the variants without leading to any false-negatives. By applying the prefix-based filtering (Wang et al. 2009), the size of partition signatures can be further reduced to \( O(l_p \cdot \tau) \).

Example 4 Continuing with example 3, by setting the prefix length to \( l_p = 3 \), we have the prefix variants of \( s \) as shown in Table 12.

5 Implementing PartSS Filtering in EDJ

Given two sets of strings \( R \) and \( S \), the task of edit similarity join (or similarity join with an edit distance threshold \( \tau \)) is to find all pairs of strings \( (r, s) \) \((r \in R \land s \in S)\) such that their edit distance is no larger than the given threshold \( \tau \), i.e., \( \{ (r, s) | \text{ed}(r, s) \leq \tau, r \in R, s \in S \} \).

To avoid pairwise comparison between two string sets \( R \) and \( S \), an inverted list is usually leveraged, with one set of strings indexed with their signatures in the inverted list. In this inverted list, a signature \( \text{sig} \) is mapped to a posting list \( I_{\text{sig}} \), in which each entry records the \( \text{id} \) and other information about the string which has \( \text{sig} \) in its signature set. Strings in the other set are taken as queries to search against the inverted list. Candidate matches of a query string are retrieved by generating signatures for the query string and searching them against the inverted list. Final results can be found by verifying the true edit distance between the query string and retrieved ones. Filtering techniques can be applied during this query and candidate generation process.

We observe that most signature schemes (including PartEnum, Q-gram based) are symmetrical in applying the same signature generation method to both the indexed strings and the query strings. PartSS asymmetrically generates partition variants for indexed strings with both PlusOne partition scheme and partition variant generation scheme, then only generates partitions for query strings with the PlusOne partition scheme. In the inverted list, each entry in the posting list \( I_{\text{sig}} \) is in the form of \( (\text{rid}, \text{pid}) \), where \( \text{rid} \) refers to the \( \text{id} \) of the string which has \( \text{sig} \) as one of its partition variants, and \( \text{pid} \) refers to the position of the partition which has \( \text{sig} \) as one of its variants.
in \textit{rid} string. Then for each query string \(s\), \(\tau + 1\) positional partitions are generated and searched against the inverted list, with each partition in the form of \(\{\text{partition}, \text{pid}\}\). The retrieved indexed strings which satisfy the following two conditions become candidate matches of \(s\):

- **Positional Partition Filtering**: at least one partition (variant) should be in the same position as the retrieved index string and the string \(s\), i.e., \(\text{pid} = \text{pid}'\).

- **Length Filtering**: the deviation between the length of the retrieved partition and that of \(s\) should be within \(\tau\).

## 6 Implementing PartSS Filtering in ED-AME

Given a dictionary of string entities \(R\), for an input document \(D\), the task of approximate membership extraction with edit distance threshold \(\tau\) is to find all substrings from \(D\) within \(\tau\) edit distance from one entity in \(R\), i.e., \(\{D[i...j] \mid \exists r \in R, \text{ed}(D[i...j], r) \leq \tau\}\), where \(D[i...j]\) represents a substring located at positions from \(i\) to \(j\) in \(D\).

A straightforward way to apply PartSS in ED-AME is to index the dictionary \(R\) in an inverted list in the same way as we do in EDI, then enumerate all the possible substrings within the length \(L_{\min} - \tau, L_{\max} + \tau\) (\(L_{\min}\) and \(L_{\max}\) represent the shortest and longest length of entities in \(R\)) from the given document \(D\) as query segments. Candidate entities can be retrieved by probing each query segment against the inverted list. The final results of ED-AME can be obtained by verifying all query segments and entity pairs. Although promising a full coverage, this algorithm inefficiently exhausts too many query segments, some of which have no match at all.

A more efficient query document processing algorithm was proposed and combined with NGPP by Wang et al. (2009), which can jump across many unnecessary substring and entity pairs by leveraging fixed prefix of partitions. Here we briefly introduce how to combine this document processing algorithm with PartSS. In this algorithm: all dictionary entities are classified into long entities and short entities and indexed into two inverted index \(I_{\text{long}}\) and \(I_{\text{short}}\) respectively. For each long entity no shorter than \(k_r l_p\), we generate partition variants by the \textit{PlusOne} and \textit{Variant Generation} rule, then all \(l_p\)-prefixes of partitions are generated and then indexed into \(I_{\text{long}}\). For each short entity shorter than \(k_r \cdot l_p + \tau\), a prefix of \(\text{min}(|e|, l_p)\) is taken to generate its \(\tau\)-variant deletion neighborhood (see section 3.1), which are then indexed into \(I_{\text{short}}\) (note that entities between \(k_r \cdot l_p\) and \(k_r \cdot l_p + \tau\) are mapped into both indexes to ensure no match will be missing in the document processing (Wang et al. 2009)).

Once all dictionary entities are indexed, for a query document \(D\), we enumerate all \(l_p\)-length segments, denoted as \(\{\text{seg} = D[p...p + l_p - 1] \mid 0 \leq p \leq |D| - L_{\min} + \tau + 1\}\), and search them against the two inverted indexes in the following manner.

- **Searching against Long Index**: we use \(seg\) to search against the long index \(I_{\text{long}}\) to find all entities which have \(seg\) as a prefix of a partition variant at any position of the entity. Then for each entity \(e\) in the retrieved candidate set \(E\), assume \(seg\) is the prefix of variant of its \(\text{pid}\)-th partition, such that \(e\) might match with any substring which has \(seg\) as prefix of its \(\text{pid}\)-th partition in \(D\). To locate these substrings in \(D\), for any possible length \(m\) within \([|e| - \tau, |e| + \tau]\) (see the Length Filtering in section 5.2), the partition size can be decided as \(\left\lceil \frac{m}{k_r + \tau} \right\rceil\) for the first \(k_r\) \(\%\) partitions, and \(\left\lfloor \frac{m}{k_r + \tau} \right\rfloor\) for the other partitions (see the \textit{PlusOne} partition scheme in section 4.1), so that we can easily deduce the location of the substring (including starting and ending position) in \(D\). This procedure is called query segment instantiation.

- **Searching against Short Index**: we generate \(\tau\)-deletion neighborhood for each possible substring within the length \(\max(L_{\min} - \tau, l_p)\) in \(\text{seg}\). If the query segment is shorter than \(l_p\), we directly probe it against the index to find all possible match entities. Otherwise, a simplified query segment instantiation is needed to find all \(2\tau + 1\) possible query segments.

**Remarks**: The prefix length \(l_p\) determines whether a string should be indexed in the short or long inverted list. The larger the prefix length, the more strings indexed in the short inverted list. When \(l_p\) is larger than \(\max(L_{\min} - \tau, l_p)\), all strings are only indexed in the short inverted list such that this document processing algorithm degrades into pure deletion neighborhood generation-based algorithm FastSS (Bocek et al. 2007). In real-world application, an optimal value for \(l_p\) should be determined empirically.

## 7 Experimental Study

In this section, PartSS is compared to the two current state-of-the-art methods QGRAM or NGPP when applied to EDJ (Edit Distance similarity Join) or ED-AME (Approximate Membership Extraction with Edit Distance threshold) on several real-world data sets.

- **QGRAM**: The \(q\)-gram-based filter which not only contains the three filtering techniques including count, position and length filtering proposed in (Gravano et al. 2001), but also incorporate state-of-art techniques such as prefix filtering (S. Chaudhuri and Kaushik 2006), location-based filtering and content-based filtering (Xiao et al. 2008). In the experiments, we set different size of \(q\)-grams for strings with different length. The following optimal \(q\)-gram size for string \(s\) with different length is adopted from the paper (Wang et al. 2009).

\[
q = \begin{cases} 
2 & (1 \leq |s| \leq 13) \\
3 & (12 \leq |s| \leq 20) \\
4 & (|s| \geq 18)
\end{cases}
\]

- **NGPP**: The Neighborhood Generation with Partitioning and Prefix-based pruning, proposed in (Wang et al. 2009).

- **PartSS**: The Partition-based filtering with Scaling and Shifting operations proposed in this paper.

### 7.1 Experimental Settings

We evaluate the performance of the three filters in EDJ on the following three real-world datasets.

- **DBLP-Papers**: A set of 300k randomly selected paper titles from the DBLP database \(^1\).
- **GENE**: A random selection of 200k entries from more than 1M Gene/Protein lexicon generated from MEDLINE documents by (Tanabe and Wilbur 2004).
- **TEXAS**: 150k records from Texas \(^2\), a text dump of Broker and Sales Licensees database from the Texas Real Estate Commission. Texas has been used in (Li

\(^1\)http://www.informatik.uni-trier.de/~ Ley/db
\(^2\)http://www.trec.state.tx.us/LicensedDataDownloads/trecfile.txt
et al., 2007; Xiao et al., 2008). Each record in Texas is a concatenation of 19 attributes, including person names, addresses, and licence information.

We also evaluate the performance of the three filters in ED-AME on the three real-world datasets below.

- **DBLP-Persons**: A set of 250k person’s names randomly selected from the DBLP bibliography dataset as the dictionary. The documents are 5k references extracted from CiteSeer, each containing paper id, author, title and venue information.

- **GENE**: The dictionary is the GENE we mentioned above. The documents are 10k references from the TREC9 Filtering Track Collections. Each reference author, title and abstract fields.

- **CONLL**: The dictionary is composed of 8.2k entities including person’s names, organizations and locations, from the shared task of Conference on Computational Natural Language Learning 2003. The documents are 20K news articles from Reuters dataset.

The properties of those datasets are provided in Table 13 and 14 below.

### Table 13: Properties of the datasets used for evaluation I

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Size</th>
<th>Average Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBLP-Papers</td>
<td>300k</td>
<td>66.4</td>
</tr>
<tr>
<td>GENE</td>
<td>200k</td>
<td>22.4</td>
</tr>
<tr>
<td>TEXAS</td>
<td>150k</td>
<td>112.1</td>
</tr>
</tbody>
</table>

### Table 14: Properties of the datasets used for evaluation II

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Size</th>
<th>Average Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBLP-Persons</td>
<td>250k</td>
<td>13.9</td>
</tr>
<tr>
<td>CiteSeer</td>
<td>5k</td>
<td>123.2</td>
</tr>
<tr>
<td>GENE</td>
<td>200k</td>
<td>22.4</td>
</tr>
<tr>
<td>TREC</td>
<td>10k</td>
<td>1134.5</td>
</tr>
<tr>
<td>CONLL</td>
<td>8.2k</td>
<td>11.5</td>
</tr>
<tr>
<td>Reuters</td>
<td>20k</td>
<td>668.5</td>
</tr>
</tbody>
</table>

The edit distance threshold $\tau$ is set within range $[1,3]$, as it covers many important applications (Lee et al. 2007). We also adopt the local edit distance threshold $\tau'$ for strings with different length according to (Wang et al. 2009).

\[
\tau' = \begin{cases} 
\min(1, \tau) & (1 \leq |s| \leq 5) \\
\min(2, \tau) & (6 \leq |s| \leq 11) \\
\tau & (|s| \geq 12)
\end{cases}
\]

In this section, we mainly compare the performance of the three filters in EDJ and ED-AME through the following two measures: 1) Run_Time: The overall running time, which is the most important factor to judge if a filtering works well; 2) Cand_Size: The numbers of candidate pairs before the verification procedure, which can also reflect the effectiveness of the filtering. Besides, we also measure the size of index entries generated in ED-AME, which reflects the space complexity of filters. Our experimental environment is an Intel 2.13GHz Pentium Dual-Core processor, 2GB memory, running Windows XP Professional. All the approaches are implemented using Java.

### 7.2 Measure the Prefix Length

The “prefix length” $l_p$ is the only parameter in our method, hence its optimal value is empirically established through preliminary experiments. The effect of $l_p$ is measured on all datasets by setting $\tau = 1,2,3$.

As we can see from the detailed results in Figure 3, as the value of $l_p$ increases (before reaching $l_p = 10$ or $l_p = 7$ for CONLL), the size of index increases, while the size of candidate size decreases. It can be explained by the features of PartSS and FastSS: although the entries of FastSS are more selective than PartSS, FastSS usually generates more entries than PartSS (especially when $\tau$ is large). Meanwhile, the overall time to set up the index and do the edit similarity join first decreases, then increases. We conclude that the optimal value of $l_p$ for the GENE is 9 since it reaches the highest efficiency as shown in Figure 3[c]. Through experimental study on other data sets, we found that the optimal value of $l_p$ decreases as the average length of dictionary string entries increases. The optimal value of $l_p$ for other datasets are: $l_p = 6$ for DBLP-Persons, $l_p = 3$ for DBLP-Papers, $l_p = 2$ for TEXAS, $l_p = 8$ for CONLL.

### 7.3 Edit Similarity Join

We now compare the performance of PartSS and QGRAM filterings in EDJ on three datasets. As we can see in Figure 4(d),(e),(f), the general trend is that running time grows exponentially as the $\tau$ increases. For all settings, PartSS is more efficient than QGram. Although QGrams usually prunes even more even join pairs than PartSS (Figure 4(a),(b),(c)), the count filtering is still a bottleneck.

### 7.4 Approximate Membership Extraction

We compare the performance of filters in ED-AME over three datasets. Here the optimal value of $l_p$ for NGPP is: $l_p = 10$ for DBLP-Persons, $l_p = 10$ for GENE, $l_p = 7$ for CONLL. As $\tau$ increases, the running time of both methods grows exponentially (Figure 5(g),(h),(i)). For all settings, PartSS is more efficient than NGPP. As expected, the generation of deletion neighborhood for each partition of a query segment by NGPP leads to slower processing, although it shares almost the same searching strategy with PartSS.

### 8 Conclusion

In this paper, we proposed a novel partition-based filter PartSS for EDJ and ED-AME, which improves over the state-of-the-art method NGPP with the implementation of a new partitioning scheme and variants generation rules. Without generating deletion neighborhood, we need less space and time to filter more false positive string pairs than NGPP. We presented how this filter can be applied into both EDJ and ED-AME and an experimental study based on several real-world data sets demonstrated the effectiveness of the filtration based on our partition-based scheme.

As future works, we will apply this filter into real-world applications such as retrieving results for typo keywords in information retrieval or recognizing homologous DNA strands in bioinformatics. To reach the best performance in our target applications, we will determine how to combine several filter techniques.

### References

Figure 3: The Effect of Prefix Length on GENE Data Set

Figure 4: Experimental Results for Edit Similarity Join


Figure 5: Experimental Results for Approximate Membership Extraction with Edit Distance Constraints


A Triangular Decomposition Access Method for Temporal Data - TD-tree

Bela Stantic  Rodney Topor  Justin Terry  Abdul Sattar

Institute for Integrated and Intelligent Systems
Griffith University,
Queensland, Australia
Email: b.stantic, r.topor, j.terry, a.sattar@griffith.edu.au

Abstract

In this study, we investigate and present a new index structure, Triangular Decomposition Tree (TD-tree), which can efficiently store and query temporal data in modern database applications. TD-tree is based on spatial representation of interval data and a recursive triangular decomposition of this space. A bounded number of intervals are stored in each leaf of the tree, which hence may be unbalanced. We describe the algorithms used with this structure. A single query algorithm can be applied uniformly to different query types without the need of dedicated query transformation. In addition to the advantages related to the usage of a single query algorithm for different query types and better space complexity, the empirical performance of the TD-tree is demonstrated to be superior to its best known competitors. Also, presented concept can be extended to more dimensions and therefore applied to efficiently manage spatio-temporal data.

Keywords: Temporal Databases, Access Method

1 Introduction

A temporal database is one that supports some aspect of time distinct from user-defined time. Over the last two decades interest in the field of temporal databases has increased significantly, with contributions from many researchers (Date et al. 2002), (Snodgrass 2000). In the literature, two time lines of interest have been mentioned, transaction time and valid time. The valid time line represents when a fact is valid in the modelled world and the transaction time line represents when a transaction was performed. A bitemporal database is a combination of valid and transaction time databases (Date et al. 2002). Because temporal databases are in general append only, they are usually very large in size, thus efficient access method is even more important in temporal databases than in conventional databases (Dyreson et al. 1995). Many multidimensional access structures have been proposed and some of them have been recommended for handling temporal data (Kumar et al. 1998). The effectiveness of the majority of these index structures has been theoretically evaluated (Salzberg & Tsotras 1999). Proposed access methods for temporal data can be classified on the techniques used as follows:

- Extensions of data partitioning spatial indexing structures (Guttman 1984) such as the Segment R-tree (Kolovson & Stonebraker 1991), 4R-tree (Blinjutse & et al. 2000), or a number of partially persistent methods (Kumar et al. 1998),
- Modifications of regular B+-tree access structures such as the Fully Persistent B+-tree (Lanka & Mays 1991) and the Snapshot index (Tsotras et al. 1995),
- Techniques based on incremental structures such as the Time Index (Elmasri et al. 1990), Time Index+ (V.Kouramajian & et.al 1994) and the Monotonic B+-tree (R.Elmasri et al. 1993),
- Employing the existing B+-tree access structure by mapping of one-dimensional ranges to one-dimensional points, as is the case in MAP21 (Nascimento & Dunham 1999), mapping strategy that linearize the data like Interval Space Transformation method (IST) (Goh & et al. 1996) or managing the indices by two relational indexes the RI-tree (Krieger et al. 2000), (Enderle et al. 2005).

Data partitioning access methods, such as spatial indexes, use a spatial containment hierarchy that clusters data into bounding regions at the leaf level. The nearby internal nodes are then clustered into bounding region of the parent node forming a hierarchical directory structure. These regions may not represent the entire data space and could overlap. Overlapping is a problem for data partitioning access methods because even for a simple point query it may need to examine multiple paths. When open ended now-relative intervals (where the ending point of the temporal interval follows the current time) are represented with widely used maximum timestamp approach a significant overlapping between nodes and dead space causes very poor performance of the index (Stantic et al. 2004), (Stantic et al. 2003).

We intend to propose access method for temporal data that relies on the exploitation of the relational database systems built-in functionalities and to utilise the native Data Definition Language (DDL), Data Manipulation Language (DML) and to use PL/SQL procedure environment within the SQL standard.

A number of access methods for temporal data that utilise the relational database systems built-in functionalities have been proposed, including: MAP21 (Nascimento & Dunham 1999), Time Index (Elmasri et al. 1990), Interval B-tree (Ang & Tan 1994), those based on interval space transformation (Goh & et al. 1996) and RI-tree (Krieger et al. 2000).
We observe that the proposed access methods that rely on relational database systems built-in functionality, such as Time B-tree and those based on interval space transformation IST, have either space complexity problem or are generally tailored to be efficient only for specific query types. In (Kriegel et al. 2000) it has been shown that the RI-tree is superior to the Window-List (Ramaswamy 1997), Oracle Tile Index (T-Index) and IST-technique. In (Kriegel et al. 2001) work was extended and an algorithm for general interval relationships has been presented but there is still a need to tailor query transformation to the specific query types. It is our intention to propose an efficient access method for temporal data with logarithmic access time and guaranteed minimum space complexity that can answer a wide range of query types with the same query algorithm.

In this paper we present and investigate the Triangular Decomposition Tree (TD-tree) access method to index and query temporal data. In contrast to previously proposed access methods for temporal data this method can efficiently answer a wide range of query types, including point queries, intersection queries, and all nontrivial interval relationships queries, using a single algorithm, without dedicated query transformation.

The TD-tree is space partitioning access method. The basic idea is to manage the temporal intervals by a virtual index structure that relies on a two-dimensional representation of intervals (Stantic et al. 2010) and a triangular decomposition method. The resulting binary tree stores a bounded number of intervals at each leaf and hence may be unbalanced. As data is only stored in leaves, traversing the tree avoids disk accesses, and tree depth hence does not affect performance. Using the interval representation, any query type can be reduced to a spatial problem of finding those (triangular) leaves that intersect with the spatial query region. TD-trees can be implemented on top of a standard relational DBMS.

The efficiency of the TD-tree is due to the virtual internal structure so there is no need for physical disk I/O’s, query algorithm that ensures pruning, and efficient clustering of interval data. On top of the advantages related to the usage of a single query algorithm for different query types and better space complexity the empirical performance of the TD-tree is demonstrated to be superior to its best known competitors.

2 Related work

A number of index structures for temporal data are described in the literature (Salzberg & Tsotras 1999). The existing temporal access structures, as highlighted in section 1 fit in one of the following groups: (1) Extensions of data partitioning methods spatial indexing structures; (2) Modifications of regular B*-tree access structures; (3) Techniques based on employing the existing B*-tree access structure on mapping of one dimensional ranges to one dimensional point.

We will focus on indexing structures from group (3) which can be utilised by exploiting the structures and functionality of commercial RDBMSs and rely on the relational paradigm. We briefly discuss typical representatives from group (3) and highlight their advantages and disadvantages.

The Time Index (Elmasri et al. 1990) is an index structure for valid time intervals. It is a set of linearly ordered indexing points that is maintained by a B*-tree. The disadvantage of this approach is the space required for the index, as for each point in time a bucket of pointers refers to the associated set of valid intervals. Since an interval may be registered with several points in time, the space requirement is $O(n^2)$ for storing $n$ intervals. This is a problem, particularly for data with many long living tuples.

The Interval B-tree (IB-tree) (Ang & Tan 1994) overcomes the problems related to the extensive space usage of the Time index. It represents an implementation of the Edelsbrunner intervals tree using an augmented B*-tree rather than a binary tree. The main memory model of the interval tree is transformed into an efficient secondary storage structure that preserves the optimal space and time complexity. The disadvantage of this approach is the complex three-fold model, which requires a dedicated structure for each level. This makes the IB-tree less attractive from the viewpoint of time complexity.

The access method (ISP) (Goh & et al. 1996) is based on interval space transformation. Since the data space may grow dynamically at the upper bound, this method is well suited for appending intervals. It indexes lists created on different orders, start time, end time or duration. This access method is highly specialized with respect to the suggested mapping and can not efficiently answer more complex queries such as intersection query or point query.

The Hierarchical Triangular Mesh (HTM) (Szalay et al. 2007) suited for indexing the sphere and especially for astronomy data. It subdivides the half surface of a sphere into four spherical triangles of similar, but not identical, shapes and sizes. Every triangle is further subdivided into four smaller triangles. Division forms a balanced tree, which is then indexed with the Quad-tree. The HTM is highly dedicated for the data that have an inherent location on the celestial sphere. The HTM has been mentioned as it has triangles as a region as our method and to highlight the differences.

The Relational Interval Tree (RI-tree) is an access method for general closed interval data, it can be created for any relational or object-relational table containing intervals (Kriegel et al. 2000). Analytical and experimental evaluation of the RI-tree shows that the performance of this method is superior to the other approaches. This is achieved by introducing a virtual primary structure. Although the structure is space-oriented, the storage of intervals is object-driven so no storage space is wasted for empty regions in the data space. In (Kriegel et al. 2001) work was extended and an algorithm for general interval relationships has been presented but still there is a need for tailored query transformation to the specific query types. It is our intention to propose an efficient access method for temporal data that can answer wide range of query types with the same algorithm and that does not require tailored query transformation for different query types.

3 Representation of intervals and interval relationships

We assume a discrete, totally ordered time model with epochs in the range $[0, \lambda)$, for some (large) $\lambda > 0$. It is straightforward to map absolute timestamps into such a range of natural numbers, as every Unix system, for example, does. We consider only semi-open intervals $[t_s, t_e)$, where $0 \leq t_s < t_e \leq \lambda$. Each such interval can then be represented as a point $(t_s, t_e)$ in two-dimensional space as shown in Fig. 1. Here, the first coordinate represents the start, $S$, of the interval and the second coordinate represents the end, $E$, of
In particular, Allen described 13 distinct interval algebra relationships that may hold between pairs of intervals (Allen 1983), which we now consider. Let $I_{st} = [s_i, e_i]$ be a stored interval and $I_{qt} = [s_q, e_q]$ be a query interval with exactly the same starting and ending points (note query interval is closed on both sides in contrast to data interval which is semi-open according to the definition).

Each of the 13 IA relationships may now be represented as a region, line or point in our two-dimensional space. When we study Allen’s relationships with indexing and query evaluation in mind, we observe that they fall into two distinct groups.

- Relationships between two intervals that represent simple comparison of the same starting or ending points. Such relationships include those where both intervals start or end with the same epoch, e.g., the relationships ‘start’ $s$, ‘start-by’ $s_b$, ‘finish’ $f$ and ‘finish-by’ $f_b$, and those where one interval starts with the same epoch that the other ends, e.g., ‘before’ $b$, ‘after’ $a$, ‘meet’ $m$ and ‘meet-by’ $m_b$. Each of these eight relationships can be queried efficiently by one dimensional index structures such as $B^+$. This is because the problem is reduced to a simple comparison of two points, start or end.

- Relationships between two intervals that require the comparison of both start and end points of both intervals. These five relationships are ‘overlap’: $o$, ‘overlap-by’: $o_b$, ‘during’: $d$, ‘contain’: $c$ and ‘equal’: $e$. To efficiently answer these queries special access method is required.

From now on we focus on the problem of efficiently answering queries about relationships in the second group. We also study queries about the more general ‘meets’ relationship and its special case the ‘membership’ relationship (or ‘point’ query). The basic query types we consider are queries from group two plus intersection and point query.

The universe of intervals is: $U = \{[u_s, u_e], 0 \leq u_s < u_e \leq \lambda\}$. Please note that due to the definition of intervals semi-open $u_s$ must be only less than $u_e$ and can not be equal. Then Fig. 2 (a) shows a set of intervals $A, B, C, D$, a query point at $T_0$, and a query interval $I_{q} = (T_1, T_2)$. The result of each query type above is then a two-dimensional rectangle, or point, as defined below.

- Equality Query EMQ - checks if the database contains an interval which equals the query interval: $EMQ([i_s, i_e]) = \{[r_s, r_e] \mid r_s \leq i_s \wedge r_e = i_e\}$ is a point in two-dimensional space;

- Intersection Query (IQ), Fig. 2 (b) - finds all intervals that intersect the query interval: $IQ([i_s, i_e]) = \{[r_s, r_e] \mid (r_s < i_s \wedge i_e < r_e) \}$ is a rectangle $[(0, \lambda), (i_e, i_s)]$, in our example $i_s = T_1$ and $i_e = T_2$;

- Point Query (PQ), Fig. 2 (c) - also called timeslice query is a special case of intersection query it finds all intervals that contain the query point: $PQ(p) = \{[r_s, r_e] \mid r_s < T_0 < r_e\}$ is a special case of IQ and results in the rectangle $[(0, \lambda), (p, p)]$, in our example $p = T_0$;

- Contained-in Query (CQ), Fig. 2 (d) - finds all intervals that are contained in the query interval: $CQ([i_s, i_e]) = \{[r_s, r_e] \mid (r_s < i_s < r_e \wedge i_e < r_e) \}$ can be simplified to $\{[r_s, r_e] \mid (i_s < r_s < r_e \wedge i_e < r_e) \}$, maps to the rectangle $[(i_s, i_e), (i_e, i_s)]$;

- Enclosure Query (EQ), Fig. 2 (e) - finds all intervals that contain the query interval: $EQ([i_s, i_e]) = \{[r_s, r_e] \mid (r_s < i_s < r_e \wedge i_e < r_e) \}$ can be simplified to $\{[r_s, r_e] \mid (r_s < i_s < i_e < r_e) \}$, as $r_s < r_e$ and $i_s < i_e$, and results in the query box $[(0, \lambda), (i_s, i_e)]$.

The point of this analysis is that the evaluation of every query type can now be reduced to the spatial problem of finding all data intervals that belong to the rectangle associated with that query. In particular, this means that every query type can be evaluated by a common algorithm, which is what we now study. Note, to form a rectangular query region for particular query type, the query region can extend under the line $E = S$, as for example for intersection query Figure 2 (b) and containment query Figure 2 (d). Because $0 \leq i_s < i_e \leq \lambda$ no intervals will be registered under the line $E = S$ so extending query region under the line $E = S$ to form rectangular query region will not affect the answer.

---

**Table 1: Parameters and symbols definitions**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda$</td>
<td>Arbitrary maximum value</td>
</tr>
<tr>
<td>$s_i$</td>
<td>Interval Start</td>
</tr>
<tr>
<td>$e_i$</td>
<td>Interval End</td>
</tr>
<tr>
<td>$V$</td>
<td>Vertical axis of of Cartesian space</td>
</tr>
<tr>
<td>$H$</td>
<td>Horizontal axis of Cartesian space</td>
</tr>
<tr>
<td>$l$</td>
<td>maximum depth of the tree</td>
</tr>
<tr>
<td>$P$</td>
<td>Parent triangle</td>
</tr>
<tr>
<td>$C$</td>
<td>Child subtriangle</td>
</tr>
<tr>
<td>$d$</td>
<td>Direction of the triangle apex</td>
</tr>
<tr>
<td>$b$</td>
<td>Blocking factor</td>
</tr>
</tbody>
</table>

---

**Figure 1: Interval representation in two-dimensional space**

For point data there are only a few distinct query types, e.g., point queries and range queries, but for interval data there are many different query types, e.g., in particular, Allen described 13 distinct interval algebra (IA) relationships that may hold between pairs of intervals (Allen 1983).
The structure of our indexing method is based on the observation, that all data and query intervals of interest represented in two dimensional space lie in the isosceles, right-angle triangle with vertices at \((0,0), (0, \lambda)\) and \((\lambda, \lambda)\), which lies above the line \(E = S\). We call this triangle the basic triangle Figure 1. This is due to nature of interval space transformation and fact that \(i < i_e\).

Given that our region of interest is a triangle, our main proposal is to recursively decompose the basic triangle into two smaller triangles. This triangular decomposition of the basic triangle forms a tree which we call a TD-tree. This tree is not balanced in general. Data intervals (points in our two-dimensional space) are stored in the database in blocks associated with the leaves of the TD-tree. Figure 3 shows the second and third level of a unbalanced triangular decomposition. Arrows point to the “apex”, the right-angled vertex of the triangle, of each triangle.

In such a triangular decomposition, each triangle is uniquely identified by its apex position \((s, e)\), and its direction \(d\), the direction of the arrow from the midpoint of the triangle’s hypotenuse to the apex. Note that there are eight possible directions, corresponding to the eight points of the compass, all of which are shown in Fig. 4.

Given a triangle in this decomposition, its apex and direction uniquely determine the apex and direction of each of its two subtriangles. We call these subtriangles low and high children. Figure 4 shows, for each possible direction, which are the low and high subtriangles, and where the apexes of these two subtriangles are. Note that we number the possible directions 0 to 7 clockwise starting from direction “north”.

By observation of Fig. 4, we see that it is possible to define the apex position and direction of the subtriangles of a given triangle using the following two algorithms. Algorithm 1 computes the direction \(d\) of the lower \((L)\) and higher \((H)\) children subtriangles of a parent triangle \(P\) with direction \(d\), Algorithm 1.

Algorithm 2 computes the position \((s, e)\) of the apex of each subtriangle \(C\) of a parent triangle \(P\) at any level \(l\). This is possible only knowing the position of the parent apex and its level. From Fig 1 it is straightforward that the basic triangle apex is \((0, \lambda)\) and we accepted that the basic triangle has level 0. Without loss of generality, we may assume...
that \( \lambda = 2^k \), for some \( k > 0 \). To find the children’s apex position the adjustment length that has to be applied to the parent apex position as presented in Algorithm 2. Adjustment length depends only on level of partition \( l \) and \( k \). It can be calculated as:

\[
\text{length} = 2^k \ast \left(\frac{2^l}{2}\right)^{(l-1)}
\]  

(1)

Note that both child subtriangles of the parent triangle have the same apex position. Position of the child \( C \) apex \((s, e)\) will be calculated depending to the direction \( d \) of the parent \( P \) apex using the Algorithm 2.

Note that the level of the subtriangles of a triangle are one more than the level of the triangle. Note also that the resulting tree need not be balanced. In an unbalanced tree, different leaves may be at different levels. The shape of a tree depends on the distribution and density of data intervals.

Because we can identify the apex and direction of every node of a TD-tree, starting from the basic triangle, using the two algorithms 2, we do not need to store the internal tree nodes. Thus, a TD-tree is a virtual tree. All we need to store is the value \( \lambda \) and a reference to the root node.

The actual data intervals, together with information about the intervals, are stored in a table indexed by a leaf identifier. The tree is organised so that at most \( b \) data intervals are stored with each leaf, for some integer blocking factor \( b > 0 \). A node identifier is a binary string, stored as a (binary) integer, constructed as follows. The identifier of the base triangle or tree root is 1. If a node has identifier \( \phi \), the lower and upper children of the node have identifiers \( \phi 0 \) and \( \phi 1 \) respectively. The length of the identifier is thus one greater than the depth of the node.

Information about leaf nodes themselves are stored in a separate directory, containing an identifier and number of records per leaf. The root node stores the blocking factor \( b \) and current maximum depth of the tree.

### 4.2 Insertion algorithm

Insertion of data interval into a TD-tree is performed according to Algorithm 3. We first descend the tree from the root to the virtual leaf at maximum tree depth containing the interval. This is done arithmetically, without disk access, by repeatedly selecting the lower or upper child of each node depending on the value of the interval.

The leaf found is called “virtual” because that branch of the tree may have length less than the maximum depth. For example, the upper child of the root node in Fig. 5 labelled ‘g’ is a leaf on a path of length 2, whereas the tree has maximum depth 7, as it can be seen in Table 2.

#### Algorithm 2 Children apex position calculation

begin
switch \((P.d)\)
if \( d = 0 \) then
\( C.s := P.s; \)
\( C.e := P.e \ast \text{length}; \)
else if \( d = 1 \) then
\( C.s := P.s \ast \left(\frac{\text{length}}{\sqrt{2}}\right); \)
\( C.e := P.e \ast \left(\frac{\text{length}}{\sqrt{2}}\right); \)
else if \( d = 2 \) then
\( C.s := P.s \ast \text{length}; \)
\( C.e := P.e; \)
else if \( d = 3 \) then
\( C.s := P.s \ast \left(\frac{\text{length}}{\sqrt{2}}\right); \)
\( C.e := P.e \ast \left(\frac{\text{length}}{\sqrt{2}}\right); \)
else if \( d = 4 \) then
\( C.s := P.s \ast \text{length}; \)
\( C.e := P.e + \left(\frac{\text{length}}{\sqrt{2}}\right); \)
else if \( d = 5 \) then
\( C.s := P.s \ast \left(\frac{\text{length}}{\sqrt{2}}\right); \)
\( C.e := P.e \ast \left(\frac{\text{length}}{\sqrt{2}}\right); \)
else if \( d = 6 \) then
\( C.s := P.s \ast \text{length}; \)
\( C.e := P.e; \)
else
\( C.s := P.s \ast \left(\frac{\text{length}}{\sqrt{2}}\right); \)
\( C.e := P.e \ast \left(\frac{\text{length}}{\sqrt{2}}\right); \)
end if
end

#### Algorithm 3 Insertion

Input: object_for_insertion \( OBJ \), Directory \( D \), blocking_factor, max_population and max_depth of the tree
begin
Find maxregion at max_depth where \( OBJ \) would belong;
target_region = region in \( D \) with longest number of bits in common left to right with the maxregion;
Find target_region population;
if target_region population > max_population
\begin{enumerate}
\item Increment the population in D of target_region;
\item perform Split;
\end{enumerate}
end if
end
Given the sample decomposition from Fig. 5, the directory would be as shown in Table 2. This table shows the label, identifier (in both binary and decimal) and identifier extension (in both binary and decimal) for each leaf of the tree. The identifier extension is the unique extension of the leaf identifier with zeros so that it is of length \( l \), where \( l \) is the maximum depth of the tree. Values for binary identifier and both binary and decimal identifier extension are not stored as they can be calculated from decimal identifier and max depth of the tree ‘1’.

![Figure 5: Running example regular decomposition](image)

<table>
<thead>
<tr>
<th>Label</th>
<th>Identifier (binary)</th>
<th>Identifier (decimal)</th>
<th>Extension (binary)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>101</td>
<td>10</td>
<td>1000000</td>
</tr>
<tr>
<td>b</td>
<td>1010</td>
<td>10</td>
<td>1010000</td>
</tr>
<tr>
<td>c</td>
<td>10111</td>
<td>23</td>
<td>1011100</td>
</tr>
<tr>
<td>d</td>
<td>101100</td>
<td>44</td>
<td>1011000</td>
</tr>
<tr>
<td>e</td>
<td>1011010</td>
<td>90</td>
<td>1011010</td>
</tr>
<tr>
<td>f</td>
<td>1011011</td>
<td>91</td>
<td>1011011</td>
</tr>
<tr>
<td>g</td>
<td>11</td>
<td>3</td>
<td>1100000</td>
</tr>
</tbody>
</table>

Table 2: Directory for sample tree

We attempt to insert the data interval into the actual leaf that is an ancestor of the virtual leaf found by the above traversal.

If the identifier of the virtual leaf \( w \) containing the interval is \( z \), then the identifier of the actual leaf \( v \) that is ancestor of \( w \) is given by the longest identifier in the directory that is a prefix of \( z \). For example, if the identifier of the virtual leaf containing the interval is 1010010, then the identifier of the actual leaf in which the interval should be stored is 1010.

Considering the sample from Figure 5 and Table 2, let for example an interval, according to the start and end points, would belong to region on max depth ‘1010010’. This max depth region ‘1010010’ at the maximum depth doesn’t do not exist so it is required to locate the actual region to store the interval in. That region is given by the longest identifier in the directory that is a prefix of max depth region ‘1010010’ and in our case it is the region ‘1010’. Having found the region which to store the interval, we simply update leaf identifier in table with that identifier and in directory increment number of records that region holds by one.

To ensure efficient retrieval, we store at most \( b \) data intervals with each leaf. If a leaf already has that many intervals, we construct the two children of the leaf, replace the parent with the two children in the directory, distribute the current (and new) intervals between the two children as appropriate, and repeat this process recursively if all intervals go into the same child. If this operation increases the maximum tree depth, we record the new maximum depth. This split is performed according to Algorithm 4.

**Algorithm 4 Split**

Input: Split Region SR, Directory D, blocking_factor max_pop, max_depth of the tree

begin
while not both child regions population < max_pop do
divide data of SR into children
current_depth = SR depth + 1
if current_depth > max_depth then
max_depth = max_depth + 1
end if
if child region is POINT then
Exit
end if
end while
end

It is possible that all intervals in a region that has to be split are located within one newly created smaller region, which will cause a further split. Splits will be performed until intervals can be distributed between two child regions or the maximum split was reached (region represents a point). If maximum split was reached the population of the region is allowed to grow beyond blocking factor, which means that multiple blocks may associate with one region.

### 4.3 Query algorithm

Following the analysis of Section 3, we can assume that every query corresponds to a rectangular region of the two-dimensional interval space, defined by the top-left and bottom-right corners of this region. The task of query evaluation is to find all data intervals that occur within this query region. The particular region chosen depends on whether we are performing an intersection query, an overlaps query, a contains query, and so on, but in each case the query evaluation algorithm is identical, an important property of our approach.

Query evaluation itself proceeds in two phases Algorithm 5. In the first phase we find those TD-tree nodes which are contained entirely within the query region and those TD-tree leaves which overlap (but are not contained in) the query region. This phase accesses the disk to retrieve nodes from the directory and to retrieve data intervals from overlapping leaves. In the second phase, we return the intervals in the first set of nodes, and scan the intervals in the second set of leaves for those that occur in the query region. This phase requires no additional disk access.

The first phase may be implemented as follows. It takes as input the query region \( Q \) and the directory \( D \). It returns the set of data intervals that occur within \( Q \). This phase terminates with \( A1 \) containing the set of nodes whose descendent leaves are contained entirely within \( Q \), and \( A2 \) containing the set of leaves which overlap \( Q \). By testing whether the ancestor \( R \) of \( F \) is contained within \( Q \), we can select all leaves under \( R \) in one operation. This property of our algorithm significantly reduces the number of disk accesses and improves its overall performance. To test whether a triangle is contained within a rectangle or whether a
Algorithm 5 Query algorithm
begin
Input: Directory D, Query region Q
Output: Containing leaves A1, Overlapping leaves A2
add all nonempty leaves in D to a LIST
let length L be 1
while LIST is not empty do
let F be the first leaf in LIST
let R be the ancestor of F
whose identifier consists
of the first L digits of F’s identifier
if R is contained within Q then
add all leaves in LIST
with the same prefix as R to A1
and remove them from LIST
set L to 1
else if R is disjoint from Q then
remove all leaves in LIST
with the same prefix as R from LIST
set L to 1
else if R equals F then
add R to A2
and remove it from LIST
else
increment L
end if
end while
end

4.4 Deletion algorithm
Algorithm 6 Deletion removes regions from the directory that contain zero objects due to the decrement of population. Also, this algorithm merges two children into parent region if sum of population of booth children falls under the one third of the blocking factor.

Algorithm 6 Deletion
Input: object_for_deletion, Directory D, blocking_factor
begin
delete_region = region where object_for_deletion belongs
Delete object_for_deletion
decrement the population of delete_region
if combined population of delete_region and its sibling < blocking_factor/3 then
merge two children into parent regions
end if
end

4.5 Update algorithm
Update of the interval, which causes change in region where interval will belong. Update can be seen as delete and insert and therefore handled by Delete and Insert algorithms. Update is particularly common for now-relative data when interval ending point stops following the current time and have to be closed. When an update is required initially the deletion algorithm is applied, which may cause the merging of regions, then an insert is performed as explained in subsection 4.2.

5 Experimental evaluation
To show the practical relevance of our approach, we performed an extensive experimental evaluation of the TD-tree and compared it to the RI-tree (Kriegel et al. 2000).
The RI-tree was chosen, since it provides the same practically important properties as our approach. It is easy to implement and integrate, it uses standard RDBMS methods which provides scalability, update-ability, concurrency control and space efficiency. Furthermore it has been proven (Kriegel et al. 2000) that the RI-tree is superior to the Window-List (Ramawamy 1997), Oracle Tile Index (T-Index) and LST-technique (Goh & et al. 1996) so performance results of the TD-tree can be transferred to these indexing techniques. We could not compare our TD-tree with improved implementation of RI-tree (Ende et al. 2005) as it indexes Interval-and-Value tuples together while our method only index intervals.
All experimental results presented in this section are computed on eight 800MHZ CPU - SUN Ultra-Sparc II processor machine, running Oracle 10.2.0 RDBMS, with a database block size of 8k and SGA (System Global Area) of 500MB. At the time of testing database server did not have any other significant load. We used Oracle built-in methods for statistics collection, analytic SQL functions and the PL/SQL procedural runtime environment.

5.1 Data sets
In order to simulate different real applications scenarios we used different data distributions. The start position of the intervals was always uniformly distributed on the interval domain, while the duration was varied. Following data distributions have been considered:

- Uniformly distributed start and uniform distributed length within the range \([1, 10000]\) with 20% of uniformly distributed now-relative data.
- Uniformly distributed start and exponentially distributed length according to the exponential distribution function \(y = e^{-0.00041\times x}\) with 20% of uniformly distributed now-relative data.

Uniform distribution of interval start, appearance of now-relative data and exponential distribution of the duration reflects most real world applications where short intervals are more likely to occur than long intervals. We used maximum timestamp approach to represent current time. Furthermore, in real world applications there is usually a upper bound for the interval duration and in our case we have chosen 10,000 for the upper bound, not considering now-relative data, which are represented with maximum timestamp approach.

All data set distributions had separate relations with different number of tuples, 250,000, 500,000 and 1,000,000.
5.2 Query sets

In our experiment we tested performance on intersection queries and particularly on point query as its specific case. Because of the nature of our query algorithm, by comparing the data region with the rectangular query region, as has been shown in subsection 3, results for performance evaluation apply to the other query types.

The point query that timeslices the timeline at the current time was used to determine how access method performs with now-relative data. The point query that timeslices the timeline at the current time is considered to be the most important because most often we will ask queries about the current state of reality.

5.3 Update sets

Most often updates in Temporal databases happen when facts cease to be valid (in valid time databases) or tuple is logically deleted (in transaction time databases). In both cases ending time of interval that contain semantic for ‘now’ (now-relative data) is replaced with the current time. We tested performance of our TD-tree on updates of randomly selected now-relative interval data of 100 tuples. As explained in update algorithm subsection 4.5, to perform update it is required to perform delete from the previous region and insert interval into the new region.

5.4 Experiments

The same data set is used both for RI-tree and TD-tree testing experiment. The initial relations with structure Employment(ID, Name, Position, Start, End) were replicated and altered accordingly to suit each particular method.

Relations for testing the performance of the RI-Tree, were altered with column node, which is calculated for every row of data by algorithm as explained in paper (Kriegel et al. 2001). Two B+-tree composite indexes have been created LowerIndex (node, Start) and UpperIndex (node, End). A point query is performed by calling the dedicated procedure that collects leftnodes and rightnodes and then performs the transformed SQL statement as instructed in (Kriegel et al. 2000).

Relations for testing the performance of the TD-tree were altered with column Region, which is calculated according the algorithm as explained in subsection 4.2. The root node, which contains information for \( \lambda \), blocking-factor, adjustment date and maximum depth of the tree we stored as one tuple relation. Because TD-tree has only leaf nodes it can be organised as a list and stored in directory table. To ensure that the population of a region corresponds to one block, so it can be be efficiently retrieved, we introduced a blocking factor. We built relation Employment as index organised table using Region and ID as a primary key.

5.5 Results

To compare the space requirements for RI-tree and TD-tree, we considered tables with different number of rows. We generated tables with 250,000 rows, 500,000 rows and 1,000,000 tuples. All tables are altered to suit the particular approach and all required primary and secondary indexes are created. In Fig. 6 we show the space requirement for the TD-tree and RI-tree. Results represent the sum of used space for table, primary/secondary indexes and for the TD-tree we add required space for the directory table.

![Figure 6: Comparison of the space usage (Table plus indexes)](image)

To measure the query performance we used a data set of one million tuples. Results shown in Fig. 7 and Table 4 are for the point query with uniformly distributed start and exponentially distributed length with 20% of now-relative data. Results represent disk I/O and average CPU usage for different points on timeline which contain different answer sizes. We performed tests with all data distributions mentioned in subsection 5.1, but testing resulted in similar qualitative results as those presented here.

The Theory of Indexability (Hellerstein et al. 1997) identifies I/O complexity cost, measured by the number of disk accesses, as one of the most important factors for measuring query performance. Other measures of importance such as CPU usage and query response time are also used in conjunction with the number of disk accesses to assess the performance of the query processes.

![Figure 7: Physical disk I/O as a factor of answer size](image)

For the TD-tree the number of leaf regions accessed to answer the query is simply the number of regions returned in the Primary filter. Secondary filtering only does pruning so it does not require any additional disk access, it only adds CPU usage. When the answer is smaller, interval objects pruned with the secondary filter effect the performance of the TD-tree and number of answers per one physical disk I/O is relatively smaller. In Table 3 we can see that TD-
The TD-tree enables efficient usage of clustering of the data by one dimension, i.e. region, as every region associate with block size. Clustering data improves the query performance and reduces the number of physical I/O, as shown in Table 3, clustering ensures higher number of answers per physical disk I/O. In contrast, the RI-tree cannot efficiently use clustering of data as it has to decide which dimension to use start or end. If it is clustered by node it will not result in similar improvements, as in RI-tree node are fixed size and are too large to provide effective clustering.

In Figure 7 it can be seen that the virtual structure of the TD-tree, clustering of data and the query algorithm significantly reduces the physical disk I/O reads. This is particularly the case when the answer size is bigger due to the good clustering, which is achieved by dividing the regions as often as needed.

Due to the query algorithm, which is able to bulk include or reject relevant sub-trees, CPU usage is kept low, which together with low disk I/O results in better query response time, Table 4.

Our experimental testing shows that the TD-tree query algorithm performs well on other query types such as: during, contain, before and after. This is because it compares the data region with the query region and uses the same algorithm. It is important to mention that the RI-tree needs dedicated query transformation for specific query type. Despite the TD-tree performing well on before and after query types it has worse performance in comparison with the straight forward usage of one dimensional indexes, as was anticipated and highlighted in section 3. The TD-tree does not perform well on query types such as start and finish because the query region is a line. However, these query types can be efficiently answered with one dimensional index, which is also highlighted in section 3.

6 Conclusions

We described a new approach that is demonstrably better than existing approaches for handling temporal, and more generally, interval data. More specifically, in this study, we:

- Classified the traditional interval relationships between pairs of intervals and identified which relationships represent challenging tasks for temporal data access methods;
- Presented a two-dimensional interval space representation of intervals and interval relationships to reduce all interval relationship problems to simpler spatial intersection problems;
- Showed that a wide range of interval query types can be reduced to an intersection of data with a rectangular region, so one algorithm can be applied uniformly;
- Proposed the triangular decomposition tree (TD-tree) and associated algorithms that can efficiently answer a wide range of query types including the point or timeslice queries;
- Experimentally evaluated the TD-tree by comparing its performance with RI-tree, and demonstrated its overall superior performance.

The TD-tree is a unique access method as it uses tree structure and at the same time has some characteristics of hashing because it only stores data in

<table>
<thead>
<tr>
<th>Answer Size</th>
<th>TD – tree DiskI/O</th>
<th>Answers/</th>
<th>RI – tree DiskI/O</th>
<th>Answers/</th>
</tr>
</thead>
<tbody>
<tr>
<td>19365</td>
<td>6102</td>
<td>3.17</td>
<td>11902</td>
<td>1.63</td>
</tr>
<tr>
<td>74727</td>
<td>8952</td>
<td>8.35</td>
<td>43891</td>
<td>1.70</td>
</tr>
<tr>
<td>124280</td>
<td>12958</td>
<td>9.59</td>
<td>72426</td>
<td>1.72</td>
</tr>
<tr>
<td>163530</td>
<td>16012</td>
<td>10.21</td>
<td>107068</td>
<td>1.74</td>
</tr>
</tbody>
</table>

Table 3: Average number of answers per one Physical disk I/O

<table>
<thead>
<tr>
<th>Answer Size</th>
<th>TD – tree CPU</th>
<th>RI – tree CPU</th>
<th>TD – tree Resp. time</th>
<th>RI – tree Resp. time</th>
</tr>
</thead>
<tbody>
<tr>
<td>19365</td>
<td>276</td>
<td>293</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>74727</td>
<td>301</td>
<td>878</td>
<td>18</td>
<td>26</td>
</tr>
<tr>
<td>124280</td>
<td>383</td>
<td>1,321</td>
<td>27</td>
<td>42</td>
</tr>
<tr>
<td>163530</td>
<td>427</td>
<td>1,622</td>
<td>37</td>
<td>55</td>
</tr>
<tr>
<td>186795</td>
<td>450</td>
<td>1,942</td>
<td>42</td>
<td>63</td>
</tr>
</tbody>
</table>

Table 4: CPU Usage and query response time I/O

tree even for a small answer size has better factor of answers per physical disk reads. For the RI-tree the number of answers per physical disk read is not dependent on query load, however for the TD-tree, due to the secondary filter features, the number of answers per physical read is dependent on query load and reaches the best performance on larger query loads.

Beside the queries mentioned in subsection 5.2, we tested applicability and performance of the TD-tree on several other query types, such as during, contain, and even before and after. These results will be mentioned and analysed in the next subsection.

5.6 Comparative analysis

When making performance measurements of index structures it is important to not only consider response time but also other parameters such as space requirements, clustering, CPU usage, updates, and locking. In our analysis we have concentrated on space requirements, physical disk reads, CPU usage and clustering of data. Because both the RI-tree and TD-tree rely on the relational paradigm, updates and locking are handled well by the RDBMS itself.

The TD-tree requires only one virtual index structure, which means only leaf nodes have to be stored. The list of leaf nodes are stored in the directory table and its size is very small in comparison to the table itself. In our experiment the TD-tree directory for one million interval objects and uniformly distributed start and exponentially distributed length required only 26 data blocks. In addition to directory table there is a need for extra space considering that table is index organised by region, which is comparable with the primary index of RI-tree method.

The RI-tree requires two composite index structures lowerIndex and the upperIndex. One composite index is on node and Start - start of the interval, and another composite index is on node and End - end of the interval. The size of the indexes depend on the number of interval objects and in our experiment for one million interval objects required 6708 data blocks (3354 each index), which is significantly bigger than the 26 blocks required for the TD-tree directory. For this reason, the total number of blocks required for Employment table and index structures for TD-tree is much smaller than the number of total blocks required for RI-tree table and index structures. This difference increases with increasing number of interval objects, as shown in Fig. 6.

The TD-tree is a unique access method as it uses tree structure and at the same time has some characteristics of hashing because it only stores data in
leaf nodes. In contrast to hashing methods that do not perform well on range queries, TD-trees can efficiently answer a wide range of different query types. This efficiency is ensured as it is possible to quickly include or prune all relevant and irrelevant subtrees without requiring disk accesses to the internal nodes, using only the query algorithm proposed in this paper. It is important to mention that the management of the virtual structure is done automatically by using database triggers, which fire on insert and calculate and updates the region of the record and also increments the region in the directory. If required, it also initiates and performs split. Similarly, database triggers fire on update/delete and performs actions in line with Delete and Insert Algorithm.

As a wide range of query regions of interest can be reduced to rectangles, it is possible to answer such queries using a single algorithm without requiring any query transformation. This itself, and fact that the TD-tree can be incorporated within commercial RDBMS, makes the TD-trees superior to other methods proposed for temporal data.

Presented concept can be extended to more dimensions and therefore applied to spatio-temporal data.

References


Snodgrass, R. T. (2000), Developing Time-Oriented Database Applications in SQL, Morgan Kaufmann.


Accelerating Spatial Join Operations using Bit-Indices

Elizabeth Antoine, Kotagiri Ramamohanarao, Jie Shao and Rui Zhang

Department of Computer Science and Software Engineering
The University of Melbourne, Victoria, Australia
Email: {eantoine, rao, jsh, rui}@csse.unimelb.edu.au

Abstract
Spatial join is a very expensive operation in spatial databases. In this paper, we propose an innovative method for accelerating spatial join operations using Spatial Join Bitmap (SJB) indices. The SJB indices are used to keep track of intersecting entities in the joining data sets. We provide algorithms for constructing SJB indices and for maintaining the SJB indices when the data sets are updated. We have performed an extensive study using both real and synthetic data sets of various data distributions. The results show that the use of SJB indices produces substantial speedup ranging from 25% to 150% when compared to Filter trees.

Keywords: spatial join bitmap indices, hierarchical representation, size separation, spatial locality

1 Introduction
Spatial database systems have gained a lot of interest in the research community due to the increasing popularity of automated processes in fields such as remote sensing, cartography and earth sciences. Spatial objects in CAD applications are mostly points, lines and polygons. However, more complex objects appear in applications that include robotics, virtual reality and cartography. Spatial objects take various shapes and manipulating such objects becomes computationally expensive. Approximations such as the Minimum Bounding Rectangle (MBR) are used to ease the processing. An object is represented by the smallest axis-parallel rectangle that completely contains it. In spatial database systems, the type of queries can be broadly categorized into single-scan and multiple-scan queries. Processing of single-scan queries requires at most one access to an object and therefore, the execution time is linear to the number of objects stored in the corresponding relation [Brinkhoff et al., 1993]. Examples of single-scan queries are window and point queries. However, multiple-scan queries access an object several times and therefore, execution time is superlinear to the number of objects. The most important multiple-scan query in a spatial database system is spatial join. The join operation (spatial join) finds pairs of objects (each from a joining data set) that have a specific spatial relationship.

The spatial join, just like the join operation in the relational databases, is a computationally expensive operation. The reason is that for large databases, pages may need to be fetched from disk more than once in order to compute the join. Many indexing techniques have been proposed for the storage and manipulation of spatial data that are able to execute spatial join operations with great efficiency. A method to organize rectangles based on their sizes was proposed by [Abel and Smith, 1983], and a similar approach was proposed by [Six and Widmayer, 1988] to extend grid files to represent rectangles rather than points. The method proposed by [Orenstein and Manola, 1988] uses the hierarchical representation and Z-ordering in evaluating range queries. The Quad-CIF tree proposed by [Kedem, 1982] uses the size separation method to organize rectangles. A similar method was proposed by [Guenther, 1991]. An efficient processing of window query is important for the efficiency of spatial joins. The methods listed above have been proved experimentally to be efficient for window queries but the experimental results for spatial join performance are not available.

The R-tree join algorithm [Brinkhoff et al., 1996] uses a buffer pinning technique that tries to keep the relevant blocks of entities in the buffer in order to minimize block re-reads and is efficient in the processing of spatial joins. Filter trees proposed by [Sevcik and Koudas, 1996] perform the spatial join by combining the hierarchical representation, size separation and spatial locality methods, and the number of block reads to perform the spatial join does not exceed the required minimum. Filter tree join algorithm outperforms the R-tree join algorithm by reading each block of entities at most once. However, for data sets of low join selectivity, the number of blocks processed for Filter trees is excessive compared to the number of blocks that have intersecting entities.

The goal of this work is to provide a method for accelerating spatial join operations by using Spatial Join Bitmap (SJB) indices. The file organization is based on the concepts introduced in Filter trees. The SJB indices keep track of blocks that have intersecting entities and make the algorithm to process only those blocks. We provide algorithms for generating SJB indices dynamically and for maintaining SJB indices when the data sets are updated. Although maintaining SJB indices for updates increases the cost in terms of response time, our method is beneficial when the data set is randomly distributed. For a given set of relations, SJB indices are generated during the first run of the spatial join and are maintained to perform the subsequent join operations much faster.

The sequel of the paper is organized as follows. Section 2 explains how the file organization is done based on the concepts introduced in Filter trees, algorithms for insertion and deletion of entities and algorithms for query processing. Section 3 describes our proposed method, the SJB indices. Section 4 presents an analysis of the SJB indices. Section 5 reports the experimental results, Section 6 presents the related
work and Section 7 gives the conclusion.

2 File Organization for SJB Indices

The file organization is based on the hierarchical representation, size separation and spatial locality concepts introduced in Filter trees [Sevcik and Koudas, 1996]. In contrast to Filter trees, our algorithm is implemented with the assumptions that the spatial entities are not only 2-dimensional rectangles, but also points and lines; and moreover, there are updates on the data entities. The file organization is done in two phases: generate level files, and ordering of entities in level files. To be self-contained, each phase is explained below.

*Generate level files:*

The concept of hierarchical representation is used here. Each entity is associated with a level that corresponds to a particular granularity of space partitioning [Sevcik and Koudas, 1996]. The algorithm hierarchically partitions the data space and groups the data entities into level files. The number of levels is denoted by \( L \). At every level \( l (l = 0, \ldots, L - 1) \), the number of partitions is \( 4^l \) and each level is composed of \( 2^l - 1 \) equally spaced lines in each dimension. An entity intersected by any line of level \( l \) belongs to level \( l - 1 \). This method of partitioning places the large entities at the higher levels and the small entities at the lower levels with high probability. In some cases, entities which are smaller in size may be placed in the higher levels when they overlap with the grid lines of a higher level. The partitioning method reflects the concept of size separation, i.e., entities of different sizes tend to be associated with different levels.

Figure 1 illustrates the process of recursive partitioning and shows how the entities with different sizes are associated with their corresponding levels. Entity \( i \) that intersects the grid lines of level \( l \) is fully contained in level \( l - 1 \) and hence belongs to level \( l - 1 \). In Figure 1(a), entity \( a \) is contained in the \( \frac{1}{2} \) by \( \frac{1}{2} \) partition and hence associated with level 1. However, entity \( c \) is located on the grid line of level 1 and hence associated with level 0. Entity \( d \) is much smaller and is fully contained in the \( \frac{1}{2} \) by \( \frac{1}{2} \) partition so it is associated with level 3 and entity \( b \) is in level 2. Figure 1(b) illustrates the partitioning of entities into level files.

The lines and points fall in the level whose grid lines intersect with the location of the entities or the level of the entity happens to be the lowest level to which the remaining entities fall. Figure 1 illustrates this process of finding the level of entities such as points and lines. In this example, entity \( e \) strikes the grid line of level 3 and hence associated with level 2. Similarly, entity \( f \) strikes the grid lines of level 2 and belongs to level 1. Entity \( g \) is neither contained in any partition nor does it intersect any grid line and hence belongs to the lowest level which is level 3.

*Ordering of entities in level files:*

The ordering of the entities in the level files can be done by any space-filling curve in order to cause entities in a portion of the multidimensional space to map to contiguous portions of data storage space as much as possible, is used [Sevcik and Koudas, 1996].

![Recursive Partitioning Example](image)

Figure 1: Recursive Partitioning Example

2.1 Insert and Delete Algorithms

In this section, we focus on the insertion and deletion of entities in the file structure described above. Insertion of a new entity is done by finding the level of the entity, calculating the Z-order value and adding the entity descriptor of the entity at the appropriate location in the corresponding level file.

**Algorithm 1: InsertEntity**

**input:** entity \( IE \), level files of data set \( DS \).

**begin**

Find the level of \( IE \), \( l \)
Compute Z-order value, \( Z \)
Store the specifications of \( IE \) and its Z-order value in the corresponding level file \( l \) of data set \( DS \) in the order of \( Z \)

**end**

**Algorithm 2: DeleteEntity**

**input:** entity \( IE \), level files of data set \( DS \).

**begin**

Find the level of \( IE \), \( l \)
Locate the entity descriptor based on Z-order value \( Z \) in the level file \( l \) of data set \( DS \)
Delete the entity descriptor

**end**

Deletion of an entity is done by finding the level of the entity and its Z-order value, searching the entity in the corresponding level file based on the Z-order value and deleting the respective entity descriptor. Algorithms 1 and 2 illustrate the insertion and deletion of entities.

2.2 Algorithms for Query Processing

In this section, we briefly describe how spatial join and window queries are executed using the file organization structure explained above.

**Spatial Join:**

The level files are merged to perform the spatial
join which typically resembles an L-way merge sort. The spatial join is performed as follows. Let A and B denote the two joining relations. Let $A^i(Z_m, Z_n)$ denote a page of the l-th level file of A containing entities with Z-order values in the range $(Z_m, Z_n)$, and $B^i(Z_m, Z_n)$ denote a page of the l-th level file of B containing entities with Z-order values in the range $(Z_m, Z_n)$ respectively. For level files $l = 0, \ldots, L - 1$:

- Process entries in $A^i(Z_m, Z_n)$ with those contained in $B^{l-i}(Z_m, Z_n)$ for $i = 0, \ldots, l$.
- Process entries in $B^i(Z_m, Z_n)$ with those in $A^{l-i}(Z_m, Z_n)$ for $i = 1, \ldots, l$.

The ranges of $i$ differ in the two steps in order to avoid matching $A^i(Z_m, Z_n)$ and $B^i(Z_m, Z_n)$ twice. As an example of how the spatial join is performed, consider relations A and B in Figure 2. The relations A and B have entities that extend up to level 3. Data entries in partition 6 of level 3 in relation A are compared with the data entries in partition 6 of level 3 in relation B and with the data entries in the enclosing partitions at the higher levels of relation B. The data entries in other partitions are not considered as the partitions are disjoint. In a similar fashion, the data entries in partition 6 of relation B are compared with the data entries in the enclosing partitions at the higher levels of relation A.

**Window Queries:**

In order to answer a window query, each level file need to be searched. However, searching within each level is made efficient by identifying the relevant partitions. If $k$ is the lowest level at which the query window is fully enclosed in a partition, then only entities that belong to a single interval need to be processed at levels 0 through $k$. For the levels below $k$, there will be generally two or more intervals involved. The detailed explanation and experimental results of window query processing is given in [Sevcik and Koudas, 1996]. Window query processing for SJB indices is similar to the Filter trees as SJB indices are not generated.

**Algorithm 3: Join**

input: level files of the joining relations; $DS1, DS2$.
Let $DS1^l(Z_m, Z_n)$ denote a page of the l-th level file of $DS1$ containing entities with Z-order values in the range $(Z_m, Z_n)$ and $DS2^l(Z_m, Z_n)$ denote a page of the l-th level file of $DS2$ containing entities with Z-order values in the range $(Z_m, Z_n)$.

begin
  for $l \leftarrow 0$ to $L - 1$ do
    for $i \leftarrow 0$ to $l$ do
      process entries in $DS1^l(Z_m, Z_n)$ with entries in $DS2^{l-i}(Z_m, Z_n)$
    for $i \leftarrow 1$ to $l$ do
      process entries in $DS2^l(Z_m, Z_n)$ with entries in $DS1^{l-i}(Z_m, Z_n)$
  end

(Note: The ranges of $i$ differ in the two ‘for’ loops in order to avoid the processing of matching $DS1^l(Z_m, Z_n)$ and $DS2^l(Z_m, Z_n)$ twice.)

end

3 Spatial Join Bitmap (SJB) Indices

SJB indices are generated dynamically as the entities are inserted in the relation. Consider the relations A and B, if an entity is inserted in relation A, then the SJB indices are generated for the partition the inserted entity belongs to. The SJB indices are generated or maintained by processing the inserted entity with the entities in the corresponding partitions of the joining relation B. A more detailed explanation is given in Section 3.3. The SJB index shows the existence of intersecting entities in the corresponding partition: ‘1’ represents the existence of intersecting entities and ‘0’ otherwise.

3.1 Algorithms for generating SJB indices

The SJB indices are generated as follows: Let $S^l(Z)$ denote a partition at level $l$ containing entities of relation S with Z-order value given by $Z$. Consider relations A and B, for levels $l = 1, \ldots, L - 1$:

- For every partition $A^l(Z)$, mark ‘1’ if there are intersecting entities in $B^{l-i}(Z)$ for $i = 0, \ldots, l$; mark ‘0’ otherwise.
- For every partition $B^i(Z)$, mark ‘1’ if there are intersecting entities in $A^{l-i}(Z)$ for $i = 1, \ldots, l$; mark ‘0’ otherwise.

The ranges of $i$ differ in the two steps in order to avoid matching $A^l(Z)$ and $B^i(Z)$ twice. Consider relations $DS1$ and $DS2$ that have entities extending up to level $l$. Let $pDS1$ be the partition of level $l$ in relation $DS1$ whose data entries are to be compared with the data entries in partition $pDS2$ of level $l$ in relation $DS2$ and with the data entries in the enclosing partitions ($pDS2 - 1, pDS2 - 2, \ldots, 0$) at the higher levels ($l - 1, l - 2, \ldots, 0$) of relation $DS2$. For the partition $pDS1$, the SJB index would be generated for $pDS2, pDS2 - 1, pDS2 - 2, \ldots, 0$. For relation $DS2$, let $pDS2$ be the partition of level $l$ in relation $DS2$ with the data entries in the enclosing partitions ($pDS1 - 1, pDS1 - 2, \ldots, 0$) at the higher levels ($l - 1, l - 2, \ldots, 0$) of relation $DS1$. $pDS2$ is
not compared with pDS1 as the processing is already done. For the partition pDS1, the SJB index would be generated for pDS2, pDS2−1, pDS2−2, . . . , 0 and for partition pDS2, the SJB index would be generated for pDS1−1, pDS1−2, . . . , 0.

As an example of how the SJB indices are generated, consider relations A and B in Figure 2. The relations A and B have entities that extend up to level 3. Data entries in partition 6 of level 3 in relation A are compared with the data entries in partition 6 of level 3 in relation B and with the data entries in the enclosing partitions at the higher levels of relation B. Hence, there will be four entries of either ‘0’ or ‘1’ for the partition 6 of level 3 in relation A. For relation B, there will be three entries of either ‘0’ or ‘1’ for the partition 6 of level 3. The difference in the number of entries is to avoid matching entries twice. To identify the entries of a particular level file, level file name could be stored. Along with entries of ‘0’ and ‘1’, the z-order value that represents the entities in the partition is also stored, which is value 5 for the entities in partition 6.

Algorithm 4: SJBInsert

input: level files of joining relation DS2; Z-order value of the inserted entity IE, Z; level of the inserted entity IE, l.

Let lupd denote the file in which the entity is inserted using the level file identifier l and the Z-order value Z.

Locate the SJB indices for the partition from which the entity is inserted using the level l and the Z-order value Z.

for i ← 0 to l do
  if SJB[i] is ‘0’ then
    process entries in lupd with entries in DS2l−1(Z)
  else if lupd has intersecting entries then
    SJB[i] ← ‘1’
end

Write SJB in SJB indices file of DS1 at the appropriate location.

Let DS2l(Zx, Zy) denote the entities at level l of the joining relation DS2, having Z-order values in the range (Zx, Zy) that could possibly intersect with entries in lupd.

for i ← l + 1 to L − 1 do
  for j ← Zx to Zy do
    SJB ← SearchSJB(SJB indices file of DS2, j, i)
    if SJB[corresponding bit] is ‘0’ then
      process entries in DS2l(j) with entries in lupd
      if lupd has intersecting entries then
        SJB[corresponding bit] ← ‘1’
    end
  end
end

Algorithm 4 is given to understand the process of generating SJB indices when an entity is inserted. Algorithm 5 describes the process of updating the SJB indices when an entity is deleted. Algorithm 6 illustrates the process of searching the SJB indices file of respective relation.

Algorithm 5: SJBDelete

input: level files of joining relation DS2; Z-order value of the deleted entity Z; level of the deleted entity l.

Let lupd denote the file in which the entity descriptor of the entities which belongs to the partition of the deleted entity is stored. Let DS2l(Z) denote the entities at level l of the joining relation DS2, having Z-order value Z that could possibly intersect with entities in l upd.

begin
  Locate the SJB indices for the partition from which the entity is deleted using the level l and the Z-order value Z.
  for i ← 0 to l do
    if SJB[i] is ‘1’ then
      process entries in l upd with entries in DS2l−1(Z)
      if l upd has no intersecting entries then
        SJB[i] ← ‘0’
    end
  end
  Write SJB in SJB indices file of DS1 at the appropriate location.
  Let DS2l(Zx, Zy) denote the entities at level l of the joining relation DS2, having Z-order values in the range (Zx, Zy) that could possibly intersect with entries in l upd.
  for i ← l + 1 to L − 1 do
    for j ← Zx to Zy do
      SJB ← SearchSJB(SJB indices file of DS2, j, i)
      if SJB[corresponding bit] is ‘1’ then
        process entries in DS2l(j) with entries in l upd
        if l upd has intersecting entries then
          SJB[corresponding bit] ← ‘0’
    end
  end
end

Algorithm 6: SearchSJB

input: SJB indices file of the relation, Z-order value Z, level file l.

begin
  With the level file identifier l and Z-order value Z, locate the corresponding entries in the SJB indices file.
  Retrieve the entries of ‘0’ and ‘1’ stored for Z.
end

The join after generating the SJB indices is done by looking up the corresponding entries in SJB indices file for every partition. If the value is set to ‘1’, then the processing is done. The processing (join) of the partitions at every level is done in a sorted sequence given by the Z-order value. While the SJB indices are generated, the entries in the SJB indices file are maintained in the same sorted sequence. The level identifier and the Z-order value are used to perform the search operation. The search operation in the SJB indices file could be easily implemented with the efficient searching techniques commonly available in database systems.
3.2 Processing Join Queries with SJB Indices

Algorithm 7: JoinWithSJB

```
input: level files of relations DS1 and DS2.
Let SJB_{DS1} denote SJB indices file of DS1 and SJB_{DS2}
de note SJB indices file of DS2. Let DS1^{i}(Z_m, Z_n)
denote a page of the l-th level file of DS1 containing entities with Z-order
values in the range (Z_m, Z_n) and DS2^{i}(Z_m, Z_n)
denote a page of the l-th level file of DS2
containing entities with Z-order values in the range (Z_m, Z_n).
begin
for level files l ← 0 to L − 1 do
    for i ← 0 to l do
        process entries in DS1^{i}(Z_m, Z_n) with
        entries in DS2^{i−1}(Z_m, Z_n) if their
        corresponding entries in SJB_{DS1} is
        set to ‘1’
    for i ← 1 to l do
        process entries in DS2^{i}(Z_m, Z_n) with
        those in DS1^{i−1}(Z_m, Z_n) if their
        corresponding entries in SJB_{DS2} is
        set to ‘1’
    (Note: The ranges of i differ in the two
    ‘for’ loops in order to avoid matching
    DS1^{i}(Z_m, Z_n) and DS2^{i}(Z_m, Z_n) twice.)
end
```

Algorithm 7 illustrates the spatial join with SJB
indices. When join is performed based on the SJB
indices, data entries in partition i of level l in re-
lation DS1 will be processed with the corresponding
partitions in the relation DS2 only if the index entries
are set to ‘1’. Consider Figure 2, data entries in par-
tion 6 of level 3 in relation A will be compared with
the respective partitions in relation B only when their
 corresponding index is set to ‘1’. The SJB indices are
stored in an ordered sequence and it resembles the
order in which the join is performed.

3.3 Update of SJB Indices

In the real world scenario, although the extent of up-
dates is limited, efficiently handling updates becomes
a prime concern. Updates require a simple insertion/
deletion operation, however the SJB indices need to
be maintained. The number of pages for storing the
SJB indices is highly dependent on the number of level
files and it is possible for the SJB indices to be
kept in memory for efficient processing of updates.

Consider e as the inserted entity of relation A. The
spatial join (A \bowtie B) is performed considering entity
e as the only entity of relation A. The insertion of
entities falls in the following cases:

- Case 1: The inserted entity is the only entity in
  the partition that intersects with the correspond-
ing level of the other relation.
- Case 2: The inserted entity falls in the partition
  which does not have any entity.
- Case 3: The inserted entity falls in the partition
  which has intersecting entities; hence the bit is
  already set.

Case 1 leads to the change in the SJB index of the
corresponding partition to ‘1’. Case 2 leads to the in-
sertion of the new set of bits which represents the par-
tition. The SJB index needs to be updated for Cases
1 and 2. Consider the insertion of an entity in rela-
tion DS1, call InsertEntity to insert the entity and
call SJBInsert to generate/modify SJB indices for the
inserted entity. If new set of SJB indices is generated
as per Case 2, then insertion of the generated indices
at the appropriate location in SJB indices file of DS1
is done. Deletion of entries is handled with respect
to the partition that it belongs to. Consider f as the
deleted entity of relation A. The partition p to which
the entity f belongs to is calculated and all the en-
tities of p except the deleted entity is considered as
relation A. The spatial join (A \bowtie B) is performed
and the SJB index of the respective partition is ei-
ther changed to ‘0’ or deleted based on the result of
processing. Deletion of entities in the relation would
fall into the following cases:

- Case 1: The deleted entity is the only entity that
  intersects with the corresponding levels of the
  other relation.
- Case 2: The deleted entity is the only entity that
  occupies the partition.
- Case 3: The deleted entity is not the only enti-
ty in the partition that intersects; no action
  required.

If the deleted entity is the only entity that inter-
sects, then the bit index is changed to ‘0’ for the cor-
responding partition. If the deleted entity is the only
entity that occupies the partition, the bit entries that
represent the partition are deleted. Consider the dele-
tion of an entity in relation DS1, call DeleteEntity
to delete the entity and call SJBDelete to modify SJB
indices for the deleted entity. If Case 2 is valid, then
delete the respective SJB indices from SJB indices file
of DS1.

4 Analysis of SJB Indices

The number of pages read for generating the level
files and for the join process is illustrated in [Sevcik
and Koukas, 1996]. [Sevcik and Koukas, 1996] ex-
plains in detail the distribution of entities across level
files for data set of uniform distribution, the distri-
bution of level occupancy, the average level occupied
by squares of certain size and how the size separation
is achieved by Filter tree structure. As discussed,
for the join phase every page of the joining relations is
accessed only once. However, for data sets of low join
selectivity, the number of pages processed is signif-
ically higher than the number of pages that have
intersecting entities. This is observed in the data sets
of uniform distribution where the join selectivity is
low which leads to the decrease in the time required
for processing the pages.

Let P_X represent the number of pages processed
for method X, C represent the cost (time) for a page
to be processed and T_{perf} represent the difference
in performance with respect to time. C_{MI} gives the
cost (time) of computing the bit-mapping indices. The
difference in performance with respect to response time
is given by the following equation:

\[ (P_{SpatialJoin} - P_{SpatialJoinWithMI}) \times C - C_{MI} = T_{perf} \]
4.1 Space Requirement for SJB Indices

The number of pages required for storing SJB indices is calculated for relations A and B separately. For every level file, the number of partitions is given by $4^l$ and the page size $2^{12}$ for the purpose of implementation is considered to have $2^{12}$ bits, which is 4KB. For a given relation, the value of $l_{max}$ is set to the value $L - 1$ of that relation. For relation A the number of pages is given by:

$$\sum_{l=1}^{l_{max}} 4^l (l+1) \pi^{2l}$$

For relation B the number of pages is given by:

$$\sum_{l=1}^{l_{max}} 4^l (l) \pi^{2l}$$

4.2 Analysis of Updates

The update maintenance cost for SJB indices is higher than the update cost (direct insertion and deletion) when the SJB indices are not maintained. The reason is that spatial join with SJB indices, the updated entries are processed individually and their corresponding SJB indices are updated. Let the response time for updating SJB indices be $R_{UM,MI}$ and the response time for the processing of spatial join queries with SJB indices be $R_{SJ,MI}$. The response time of spatial join operation for the relations with updated entities be $R_{SJ}$ and the response time for updating entities in the relation be $R_{UM}$. The decrease in the response time when the spatial join is performed with SJB indices balances the increase in the update cost for maintaining the SJB indices. Hence, if the spatial join is performed $\alpha$ times after an update, this method is beneficial if the following equation holds true for $\alpha \geq 1$

$$R_{UM,MI} + \alpha \ast R_{SJ,MI} < R_{UM} + \alpha \ast R_{SJ}$$

5 Experimental Results

In order to assess the performance benefits of SJB indices, we implemented the Filter tree join algorithm. Filter tree join algorithm performs better than the best R-tree join algorithm proposed by [Brinkhoff et al., 1993], as described in [Sevcik and Koudas, 1996]. Filter trees can perform the join with almost 32% savings in response time with 2.1% buffer space, relative to the R-tree with 5% buffering [Sevcik and Koudas, 1996]. As per the experimental results shown in [Sevcik and Koudas, 1996], Filter trees outperform R-trees and hence we chose to compare SJB indices with Filter trees. In the following, we present the experimental results for the join performance of Filter trees and SJB indices.

5.1 Description of Data Sets

The experiments were conducted on an Intel® Core™2 Duo Processor at 2.00GHz which has the main memory capacity of 2.0GB. The windows experience rating index for the processor is 4.7 out of 5. We experimented with both real and synthetic data sets. For the purpose of implementation, we consider the z-order value is generated at the time of generating level files and will be stored as part of the entity descriptor. This would save the processor time when computing the join results. The real data sets consist of the road segments extracted from the TIGER/LINE relation [Tig, 2000]. The relations consist of long beach county roads containing 53,145 MBRs, railways containing 128,971 MBRs, tiger streams containing 194,971 MBRs. The synthetic data sets which are of uniform and zipfian distributions have 250,000, 100,000 and 50,000 entities, respectively. Table 1 presents the data sets used for the experiments.

5.2 Spatial Join using SJB Indices

Figures 3, 4, 5 and 6 show the result of the experiments conducted on data sets of various sizes and data distributions. The benefits of using SJB indices are clearly seen in real data sets as the join selectivity is low compared to the relations of uniform and zipfian distributions.

The data sets of real, uniform and zipfian are sampled to have 25,000, 50,000, 75,000, 100,000, 150,000 and 250,000 entities and the join is performed with data set having 50,000 entities of respective distribution. The use of SJB indices decreases the response time. The experiments conducted on the data sets of uniform distribution show that the percentage improvement in the response time is found to be approximately 95%. The improvement in the response time is due to the decrease in the number of pages processed. The intersecting entities are distributed randomly and not every partition has an intersecting entity, which leads to the decrease in the pages processed. Figure 3 presents the response time for the join of two uniformly distributed data sets, U1 and U2 containing 250,000 and 50,000 entities respectively. Data set U1 is sampled as described above and the join is performed with data set U2 for every sample.

The experiments conducted on the data set of zipfian distribution show that the percentage improvement in the response time is found to be approximately 25%. The decrease in the response time in the zipfian distribution is found to be less when compared to the uniform distribution as the relations of zipfian distribution show high join selectivity in our experiments. The performance of SJB indices is high when the join selectivity is low. The join on data sets of zipfian distribution shows a low percentage improvement in performance due to the high join selectivity. The percentage difference in the number of page processed is 15%.

Figure 4 presents the response time for the join of two data sets of zipfian distribution, Z1 and Z2 containing 250,000 and 50,000 entities respectively. Data set Z1 is sampled as described above and the join is performed with data set Z2 for every sample. Although the number of partitions which have the intersecting entities is less for data sets of zipfian distribution, the number of pages processed is high. The reason is that, the partitions that have the intersecting entities are the only partitions that have the highest percentage of entities. The cost (response time) is reduced by reducing the number of pages processing to compute the join. The difference in the number of pages processed for the Filter trees and SJB indices in case of data sets of zipfian distribution is low, which results in the low performance.

The improvement in performance for the real data sets is shown in Figures 5 and 6. The percentage improvement in response time is found to be 150%, which is due to the low join selectivity and the high
Name | Description | Data Set Size
--- | --- | ---
U1 | Uniformly distributed entities | 250,000
U2 | Uniformly distributed entities | 50,000
U3 | Uniformly distributed entities | 100,000
Z1 | Entities of Zipfian distribution | 250,000
Z2 | Entities of Zipfian distribution | 50,000
Z3 | Entities of Zipfian distribution | 100,000
SD | Tiger streams of Iowa, Kansas, Missouri and Nebraska | 194,971
LB | Long beach county roads | 53,145
RA | Tiger/Line LA railways | 128,971

Table 1: Relations Used

The percentage decrease in the number of pages processed while using SJB indices. The percentage decrease in the number of pages processed is approximately 110%. The increase in the performance is due to the size of the entities, which is invariably small in case of the real data sets and the low join selectivity. The entities in the real data sets (SD) are found to be scattered all over the space and the size of the entities is small, which results in the low join selectivity. The improvement in performance for the real data sets (RA and LB) is found to be similar to the results found in real data sets (SD and LB).

Table 2 summarizes all the experimental results in this subsection and presents the percentage of improvement in response time and pages processed for SJB indices over Filter trees. The experiments show that the improvement in performance is almost 150% for real data sets (TIGER/Line). The percentage of improvement while using SJB indices is computed with the following equation:

\[
\frac{V_{FILTER\_TREES} - V_{SJB\_INDICES}}{V_{SJB\_INDICES}} \times 100
\]

where \(V_x\) means response time for method \(x\).

5.3 Updates of SJB Indices

The cost for maintenance of the updates involves the cost of updating the SJB indices. The update maintenance cost for SJB indices is expensive when compared to Filter trees as the SJB indices need to be updated. The cost of processing a spatial join operation after updates is found to be lesser when the SJB indices are used and this could be used to compensate the extra cost involved in maintaining SJB indices for updates. The experiments are conducted on data sets of uniform distribution and Zipfian distribution containing 100,000 and 50,000 entities each. The data sets are sampled to have 200, 400, 600, 800 and 1000 entries each and Figure 7 shows the number of times the spatial join has to be performed to compensate the extra cost incurred during updates. The difference in response time while using SJB indices could be used effectively to maintain updates. We found that the following equation holds true and \(\alpha\) is < 1 when the updates extend up to 1000 entries for data sets of uniform distribution.

\[
R_{UM\_MI} + \alpha \times R_{SJ\_MI} < R_{UM} + \alpha \times R_{SJ}
\]
Table 2: Percentage of Improvement in Real, Uniform and Zipfian Data Sets

<table>
<thead>
<tr>
<th>Improvement</th>
<th>Real (TIGER/Line)</th>
<th>Uniform</th>
<th>Zipfian</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SD join LB</td>
<td>RA join LB</td>
<td>U1 join U2</td>
</tr>
<tr>
<td>Time</td>
<td>150%</td>
<td>148%</td>
<td>95%</td>
</tr>
<tr>
<td>Pages Processed</td>
<td>110%</td>
<td>105%</td>
<td>60%</td>
</tr>
</tbody>
</table>

For data sets of uniform distribution, the $\alpha$ value is less than 0.4 for updates of about 1000 entries and the method saves cost even when the spatial join is not performed involving updates up to a certain limit. However, in the real world scenario, the number of updates is quite less than the number of times the spatial join is performed on the data sets. The suggested method can be used for updates up to a certain limit and is highly based on the type of data distribution. For data sets of zipfian distribution, the limit to which the updates are handled efficiently by this method is quite limited as the difference in the response time of spatial join with Filter trees and the spatial join with SJB indices is considerably low.

5.4 Performance of SJB Indices Measured for Different Data Distributions

The percentage of bit set to ‘1’ for uniform, zipfian and real data sets is given in Figure 8. $L_1$ represents level 1, $L_2$ represents level 2, and so on. It shows that for real and zipfian data sets, the number of bits set is comparatively lesser than the bits set for uniform data sets. For zipfian data sets ($Z_1$ and $Z_2$), large number of entities concentrate on few partitions, while most of the partitions at the lower levels are empty. The percentage of bits set for zipfian data sets is less but this does not decrease the number of pages processed as the intersecting entities are found in those partitions which have large number of entities. The join selectivity is found to be high in zipfian data sets and hence, the number of pages processed does not decrease. For uniform data sets ($U_1$ and $U_2$), the percentage of bits set is comparatively high but reduces the number of pages processed. The entities are equally distributed across the partitions and the number of bits set to ‘0’ relates to the number of pages ignored during processing. Although the bits set to ‘0’ is comparatively less, it affects the number of pages processed considerably as the join selectivity is low. For real data sets (SD and LB), the join selectivity is low and hence the number of bits set is also less.

Figure 9 represents the response time for performing join with the uniform ($U_2$ and $U_3$) and real data sets ($RA$ and $LB$) in the uniform-real percentage. The real ($RA$ and $LB$) and uniform data sets ($U_2$ and $U_3$) are sampled to have 100,000 and 50,000 entities respectively. The 100-0 represents the join performance when the data sets are of uniform distribution. The response time considerably decreases as the data sets are mixed with real data sets as can be seen in Figure 9. The decrease in the response time is due to the considerable decrease in the join selectivity and the decrease in the number of pages processed. The 90-10 represents the join performance when the data sets are 90% uniform and 10% real. This shows that there is a decrease in the result set by 25% which results in the decrease of response time. The entities are evenly distributed in case of uniform data distribution and every partition has an entity at all levels. When the data sets are mixed with real data sets in noted percentages, the existence of entity in every partition is not seen and the existence of intersecting entities at almost every partition which is the case in uniform data sets is no longer seen.
The percentage of improvement in response time is high when the large zipfian data distribution, join selectivity is high and decrease in the number of pages processed. In case of uniform and real data sets and it is due to the lack of difference in the number of pages processed when the real data sets are compared with the data sets of zipfian distribution. The lack of improvement in response time for data sets of uniform distribution is due to lack of difference in the number of pages processed. In case of zipfian data distribution, join selectivity is high and it attributes to the increase in the response time. The high join selectivity is due to the fact that the large number of entities concentrate on a few partitions. The percentage of improvement in response time is shown in Figure 11.

![Graph](image)

**Figure 10:** Join Performance for Zipfian and Real Data Sets

The benefit of using SJB indices is clearly seen when the real data sets are used. The benefit of using SJB indices in real data sets is shown in the Figure 9 and Figure 10, which is almost 148% in case of response time and 100% in case of the number of page processed. The 0-100 represents the join performance for the real data sets. The improvement in the response time for the real data sets is due to the decrease in the pages processed and the low join selectivity.

Figure 10 represents the response time for performing join with the zipfian (Z2 and Z3) and real data sets (RA and LB) in the zipfian-real percentage. The response time and the number of pages processed decrease at the scale of 40% as the data sets are mixed with real data sets in noted percentages. The number of intersecting entities decreases substantially when the data sets are mixed with the real data sets and this leads to the decrease in the response time and number of pages processed.

The use of SJB indices does not decrease the response time of zipfian data sets when compared to the uniform and real data sets and it is due to the lack of decrease in the number of pages processed. In case of zipfian data distribution, join selectivity is high and it attributes to the increase in the response time. The high join selectivity is due to the fact that the large number of entities concentrate on a few partitions. The percentage of improvement in response time is shown in Figure 11.

![Graph](image)

**Figure 11:** Percentage of Improvement in Response Time while using SJB Indices

The lack of improvement in response time for data sets of uniform distribution is due to lack of difference in the number of pages processed and for the zipfian distribution it is due to the high join selectivity. Figure 11 shows that the SJB indices are beneficial in data sets of uniform distribution when compared to the data sets of zipfian distribution. The high join selectivity results in the decrease of improvement in performance for data sets of zipfian distribution. The SJB indices perform extremely well when the join selectivity is low and the data sets are randomly distributed. However, SJB indices perform better than Filter trees for data sets of various distributions. The SJB indices could be further optimized by choosing the value of \( L \). Beyond a certain level \( cl \), the number of entities found at the levels lower than \( cl \) is found to be extremely less (for instance, could be stored in less than one page). In which case, \( L \) could be fixed to \( cl \) and the entities that fall beyond \( cl \) could be restricted to level \( cl \).

6 Related Work

[Brinkhoff et al., 1996] is the most widely used algorithm because of its efficiency and the popularity of the R-trees. Several spatial join algorithms have been proposed for the cases that only one of the inputs is indexed by an R-tree [Lo and Ravishankar, 1994, 1995] or when both inputs are not indexed [Koudas and Sevcik, 1997, Lo and Ravishankar, 1996, Patel and DeWitt, 1996]. The join operation is an expensive operation in relational databases and several methods have been proposed for optimizing join operations. A join index facilitates in rapid query processing and for data sets that are updated infrequently, the join index can be very useful [Gunther, 1993, Koudas, 2000, Rotem, 1991, Valduriez, 1987]. [Shekhar et al., 2002] uses the method of clustering to compute the efficient order of page access when the memory buffer size is < 10% relative to the size of the relations. Given a fixed buffer size and a buffer size which is relatively small than the size of the relations, the join index helps in finding the order of page access which could substantially reduce the number of redundant page re-accesses.

[Chan and Ooi, 1997] examines the issue of scheduling page accesses in join processing, and proposes new heuristics for an optimal page access sequence for a join such that there are no page re-accesses using the minimum number of buffer pages, and an optimal page access sequence for a join such that the number of page re-accesses for a given number of buffer pages is minimum. [Brinkhoff et al., 1993] uses a buffer pinning technique to reduce the number of page re-accesses. Filter trees proposed by [Sevcik and Koudas, 1996] uses the concept of hierarchical representation to perform the join with the minimum number of page reads. Filter trees out-perform R-trees [Abel and Smith, 1983] by reading each page only once. [Antoine et al., 2009] illustrates the extension of this work by the authors to employ recursive partitioning for trajectories; which is the recorded instances of a moving object with respect to time. They have illustrated the methodology for indexing trajectories in the unrestricted space. This recursive partitioning method can be further studied on the problem of processing continuous intersection joins over moving objects [Zhang et al., 2008].

7 Conclusion

We have presented the methods for generating Spatial Join Bitmap (SJB) indices and have shown through experimental results that SJB indices improve the performance of spatial join queries substantially. We have presented algorithms to dynamically construct SJB indices and to effectively handle updates and have shown through experimental results that the extra cost incurred while maintaining SJB indices for updates is balanced with the substantial cost saved in processing subsequent joins. This method is highly beneficial in the real world scenario as the number of
times the data set is updated is fairly low when compared to the number of times the join processing is done on the data sets. We have presented the page requirement for storing SJB indices, which is considerably less in every case and can be kept in memory. We have shown that the SJB indices perform better than Filter trees through a series of experiments conducted on data sets of various distributions. We have also shown that SJB indices outperform Filter trees by 150% for real data sets.

Acknowledgements

This work is sponsored in part by National ICT Australia (NICTA).

References

Bureau of the census. TIGER/Line Census Files, 2000.


Chee Yong Chan and Beng Chin Ooi. Efficient scheduling of page access in index-based join processing. IEEE Transactions on Knowledge and Data Engineering, 9:1005–1011, November 1997.


Scheduling with Freshness and Performance Guarantees for Web Applications in the Cloud

Yingying Zhu  Mohamed A. Sharaf  Xiaofang Zhou
School of Information Technology and Electrical Engineering
The University of Queensland
Brisbane, Australia
Email: (y.zhu3, m.sharaf, zxf)@uq.edu.au

Abstract
Highly distributed data management platforms (e.g., PNUTS, Dynamo, Cassandra, and BigTable) are rapidly becoming the favorite choice for hosting modern web applications in the cloud. Among other features, these platforms rely on data partitioning, replication and relaxed consistency to achieve high levels of performance and scalability. However, these design choices often exhibit a trade-off between performance and data freshness. In this paper, in addition to performance SLAs, we also perceive an application tolerance to data staleness as another requirement determining the end-user satisfaction and our goal is to strike a fine balance between both the quality of service (QoS) and quality of data (QoD) perceived by a key, while no operation can span multiple replicas at each replica node so that to meet the conflicting requirements of user queries and replica updates. Our experimental results show that employing our scheduling strategies for resource allocation can provide significant improvements in the overall system utility when compared to the existing ones.

Keywords: Web Database, Distributed Database, Cloud Computing, Scheduling, Consistency, SLA, Quality of Data, Quality of Service.

1 Introduction
In modern web applications, user satisfaction or positive experience determines the applications’ success (and keeps the competitors “more than a click away” [17]). A fundamental requirement in such web applications is to consistently meet the user’s expectations for page load time as expressed by a Service Level Agreement (SLA). An example of a simple SLA is a web application guaranteeing that it will provide a response within 300ms for 99.9% of its requests for a peak client load of 500 requests per second [8].

Clearly, application SLAs place stringent response time requirements on data management platforms demanding a near real-time performance. Towards this, several data management techniques have been continuously improved in order to maximize the SLA satisfaction of web database transactions. Examples of such techniques include data caching [19], adaptive transaction scheduling [9, 17], etc. However, the continuous growth in database-driven web applications as well as the complexity of user requirements required re-thinking the traditional database solutions and resulted in a new generation of highly distributed database platforms especially designed to meet the ever stringent performance requirements expected by today’s end user. Examples of such platforms include PNUTS [7], Dynamo [8], Cassandra [15], and BigTable [5].

Such platforms are expected to meet strict operational requirements in terms of performance, reliability and efficiency, and to support continuous growth of the platform needs to be highly scalable [8]. However, the design choices for these platforms often exhibit a trade-off between perceived performance and data freshness, which is the focus of this paper. In particular, most of such modern platforms share the following key design choices:

- **Data Partitioning and Replication:** Dynamically partitioning data across the available storage nodes allows the system to incrementally scale out, where adding capacity becomes as simple as adding new servers [7]. Further, in a large-scale web application, users are scattered across the globe which makes it critical to have data replicas on multiple continents for low-latency access [7]. For instance, Dynamo uses a synthesis of well known techniques for data partitioning using consistent hashing as well as data replication.

- **Key-Value Data Model:** Presents a simplified data model to the user based on a key-value data store motivated by the observation that the big majority of web applications only need primary-key data access manipulating one record at a time. For example, Dynamo provides simple get() and put() operations for the read and write to a data item that is uniquely identified by a key, while no operation can span multiple data items.

Clearly, the above features directly contribute to improving the performance of web applications and meeting the pre-specified SLA requirements. However, achieving *serializability* for web transactions over a globally-replicated and distributed system is very expensive and often unnecessary [7]. In particular, web applications expect and tolerate weaker levels of consistency. For instance, Dynamo is designed to be an *eventually consistent* data store; that is all updates reach all replicas eventually. Similarly, PNUTS provides a consistency model that is between general serializability and eventual consistency. While providing weaker levels of consistency allow for high availability, this often comes at the expense of data freshness where user queries might access stale data. This is typically accepted by most
Web applications only if the perceived staleness is bounded within some pre-specified staleness tolerance [10]. Meanwhile, current platforms cannot guarantee such bounds on data staleness while still satisfying the stringent performance SLAs. For instance, in the Dynamo platform, replica synchronization tasks are executed in the background at low priority so that to allow enough resources for running the foreground queries and meeting their SLAs.

In this paper, in addition to a query SLA, we also perceive an application tolerance to data staleness as another requirement determining the end-user satisfaction, and in turn the success of a web application. Towards this, our goal is to balance the trade-off between the perceived Quality of Service (QoS) as expressed by performance SLA and the Quality of Data (QoD) as expressed by freshness. Our approach towards achieving that goal relies on efficiently allocating the resources at each node so that to meet the conflicting requirements of the foreground user transactions and the background replica updates. In particular, in this paper we propose new schemes for scheduling the execution of both user transactions and replica updates for database-driven web applications in the cloud.

To this end, several research efforts have leveraged scheduling as a method for balancing the trade-off between QoS and QoD in contexts such as web databases [16], real-time databases [1, 13], and real-time data warehouses [2, 20, 12]. Those approaches adopt various high-level mechanisms as well as different low-level scheduling policies.

For instance, the On-Demand (OD) [1] mechanism couples the execution of pending replica updates with the arriving user requests, where all the data items read by a certain query are refreshed on demand before the execution of that query. This strategy has been shown to be beneficial in saving system resources [11]. However, the OD mechanism employs simple low-level policies for scheduling queries (and in turn updates) which fall short in meeting the requirements of modern web applications.

On the other hand, the QUTS [16] approach considers both the QoS and QoD requirements of web applications under a unified Quality Contracts [14] model. However, QUTS decouples the execution of queries from that of replica updates, where it allocates to each a separate time quota of the system resources. This allows QUTS to target general database transactions where it is not easy to determine the data objects read by a query beforehand. But at the same time, this decoupling might waste significant system resources.

To the contrary, our scheduling approach presented in this paper leverages the specific characteristics of modern key-value data stores towards satisfying the QoS and QoD specifications of web applications. In particular, we propose high-level mechanisms together with low-level scheduling policies that consider the "cost" and "benefit" of executing background replica updates in conjunction with the foreground user requests so that to improve both performance and freshness.

Moreover, between the two extremes of coupling or decoupling the scheduling of user reads and replica updates, we propose a new hybrid mechanism called FIT, which integrates the advantages of both in order to maximize the system gains. Our experimental results show that employing our scheduling schemes for resource allocation can provide significant improvements in the overall system utility when compared to existing policies.

The rest part of this paper is organized as follows. Section 2 describes the system model. Our proposed scheduling policies are presented in section 3. Section 4 describes the evaluation environment. Section 5 presents our experimental results. Section 6 finalizes this paper with conclusions and future work.

2 System Model

We consider a distributed data management platform where data is partitioned and replicated across multiple nodes. The data replication model we adopt in this paper is very close to the one currently employed by the Yahoo! P2NUTS [7]. Specifically, we assume that each data record has one master copy and multiple replicas. All the write operations on a certain record are directed to its master copy, then later propagated to the replicas. This propagation takes place in a lazy or asynchronous fashion, where a write is installed at the master first and updates are propagated in the background. This model allows for timeline consistency where all replicas will go through the same sequence of updates such that they will eventually converge to the latest update made by the application. Hence, a record read by an application might be stale unless it is the master copy or it is a replica that has already applied the latest update.

Under the model described above, a node in our system receives 3 types of record operations: 1) write to a master, 2) read to a master or replica, or 3) update to a replica. Our proposed policies operate at the node-level where they are responsible for scheduling the execution of those operations so that to maximize both QoS and QoD. Next, we describe the details of our model together with our metrics for QoS and QoD.

2.1 Database Replica

Each node typically stores a set of master records as well as another set of replica records. As described above, a master record is accessed by foreground read and write requests and it is always fresh. Whereas, a replica record is accessed by either a foreground read request (i.e., query) or a background refresh request (i.e., update) and it might be stale. In the rest of this discussion, we will focus on the latter set of records (i.e., replicas) since accessing these records leads to a trade-off between performance and freshness. In particular, we assume that a database replica \( B \) consists of \( M \) data records (or objects) \( \{O_1, O_2, \ldots, O_M\} \) that are accessed simultaneously by both user queries and system updates.

As in [10], we use the term replica broadly to include saved data derived from some underlying source tables. As such, it could be a replica in the ordinary sense as in the distributed data management platforms described above. But also a replica could be a materialized view internal to the web application for efficient query processing. For instance, to support social networking applications, P2NUTS stores materialized views as regular tables that are asynchronously maintained by the system [18].

Finally, a replica could be a web database which represents a portal updated aperiodically by external sources. For example, in a stock information application, external databases such as the New York Stock Exchange store the history of updates, whereas the web database corresponds to a snapshot view reflecting the most recent stock information as propagated by that external database [16, 1].
2.2 Updates

Updates to a data replica are queued internally in the node’s update queue until they are scheduled for execution. Each update $U_i$ has a timestamp $t_i$ representing the time when it was generated at the master copy. Further, each update modifies the value of a single record at a time using an operation such as set() or put() to set the value of a replica record to the same value committed at the master copy at time $t_i$.

Finally, each update $U_i$ is characterized by a cost $C_{iu}$, which reflects the time required for processing the update and installing it onto the replica. This processing time incorporates both CPU and I/O costs and is typically determined by monitoring the processing of previous updates over a reasonable time window.

2.3 Queries

In our model, each query $Q_i$ represents a get() operation to the key-value data store. Rendering a web page in modern web applications typically fires a large number of such operations. For example, a page request to an e-commerce site typically requires the rendering engine to construct its response by sending over 150 get() operations [8].

Similar to updates, in a key-value data store, each query $Q_i$ will have a timestamp representing its arrival time $A_i$ and will access a single record or data object $O_i$. Meanwhile, operations that touch multiple records typically require a component that generates multiple requests to individually access each record. Finally, $C_{iq}$ denotes the cost for retrieving and processing that data record. Like update processing, query processing time incorporates both CPU and I/O costs and is statistically estimated over time.

2.4 QoS and QoD Metrics

There are several metrics for capturing the user perceived QoS as well as QoD. In this paper, we focus on QoS in terms of minimizing tardiness and QoD in terms of minimizing staleness and our goal is to strike a fine balance between both metrics across all the user queries submitted to the system.

Ideally, if a query $Q_i$ finishes execution at time $F_i$, then $F_i$ should be within the QoS tolerance of $Q_i$. Similarly, the staleness of a record $O_i$ accessed by the query should be within its QoD tolerance. However, in the presence of multiple queries and updates competing for the system resources, $Q_i$ might experience queuing delays or access stale data that fall beyond its tolerance.

The natural way to capture those deviations is to define for each query $Q_i$ two deadlines: 1) Tardiness Deadline ($D_i$), and 2) StalenessDeadline ($S_i$). In our model, these two deadlines represent the QoS and QoD requirements of a query and violating either incurs a penalty to be paid by the system. In particular, to specify the QoS and QoD requirements, each query is associated with the following parameters:

- Weight ($W_i$): the weight assigned to query $Q_i$, which represents its importance to the system.
- QoS Factor ($\alpha_{iq}$): the fraction of the weight assigned to QoS, which represents the QoS importance to the application.
- QoD Factor ($\alpha_{id}$): the fraction of the weight assigned to QoD ($= 1.0 - \alpha_{iq}$), which represents the QoD importance to the application.

2.4.1 Query Perceived Tardiness

The tardiness deadline ($D_i$) is defined as:

$$D_i = A_i + \gamma_{ia}$$

where $A_i$ is the arrival time of query $Q_i$ and $\gamma_{ia}$ is its tolerance as described above.

if $Q_i$ cannot meet its deadline, the system will still execute it but it will be “penalized” for the delay beyond the deadline $D_i$. This penalty per query is known as tardiness which is formally defined as:

**Definition 1 Tardiness, $T_i$, for query $Q_i$ is the total amount of time spent by $Q_i$ in the system beyond its deadline $D_i$. That is, $T_i = 0$ if $F_i \leq D_i$, and $T_i = F_i - D_i$ otherwise.**

2.4.2 Query Perceived Staleness

The staleness deadline ($S_i$) is defined as:

$$S_i = R_i + \gamma_{id}$$

where $R_i$ is the timestamp of the first unapplied update to data object $O_i$ and $\gamma_{id}$ is the tolerance of $Q_i$ to staleness as described above.

In particular, at time $R_i$, a replica record $O_i$ is rendered stale because of the generation of a new update at the master copy. If that update is still unapplied until the time $Q_i$ is scheduled for execution, then the system will still execute the query but it will be penalized for the staleness beyond the deadline $S_i$. This penalty per query is known as staleness (or age [3]) which is formally defined as:

**Definition 2 Staleness, $L_i$, for query $Q_i$ is the total amount of staleness accumulated by $Q_i$ beyond $Q_i$’s staleness deadline $S_i$. That is, $L_i = 0$ if $F_i \leq S_i$, and $L_i = F_i - S_i$ otherwise.**

2.5 Problem Definition

Given the above definitions of tardiness and staleness, our goal is to minimize the total combined penalty incurred by the system. This combined penalty per query $Q_i$ is simply the sum of weighted staleness and weighted tardiness (as shown in Figures 1, 2, and 3) and is computed as follows:

- QoS Tolerance ($\gamma_{ia}$): the tolerance of query $Q_i$ to tardiness in time units.
- QoD Tolerance ($\gamma_{id}$): the tolerance of query $Q_i$ to staleness in time units.

| Parameter | Symbol ||---|---|---|---|---|
| Database | $U$ | Objects in Database | $\{Q_1, ..., Q_M\}$ | Write-only Transaction: Update | $U_i$ | Timestamp of First Unapplied Updates | $R_i$ | Read-only Transaction: Query | $Q_i$ | Arrival Time of Query | $A_i$ | Finish Time of Query | $F_i$ | Weight of each query | $w_i$ | QoS Factor | $\alpha_{iq}$ | QoD Factor | $\alpha_{id}$ | QoS Tolerance | $\gamma_{ia}$ | QoD Tolerance | $\gamma_{id}$ | Staleness deadline | $S_i = R_i + \gamma_{id}$ |

Table 1: Model Parameters

```
The average penalty for deferring the trade-off between QoS and QoD (Section 2). Then, we propose our new low-level scheduling policies that further enable balancing the trade-off between QoS and QoD (Section 3.2). Then, we propose our new FIT mechanism together with low-level scheduling policies that further enable balancing the trade-off between QoS and QoD (Section 3.3).

3 Scheduling Strategies

In this section, we present several strategies for the scheduling of both pending queries and updates at each node in the system. For each strategy, we make the distinction between the high-level general mechanism specifying the dependency between queries and updates (e.g., coupled, decoupled, etc.) and the low-level scheduling policy used for ordering the execution of those queries and updates (e.g., FCFS, EDF, etc.). Specifically, we first propose low-level scheduling policies that extend the On-Demand mechanism [1] by considering query characteristics (Sections 3.1) as well as update characteristics (Section 3.2). Then, we propose our new FIT mechanism together with low-level scheduling policies that further enable balancing the trade-off between QoS and QoD (Section 3.3).

3.1 Query-aware On-Demand Scheduling

Recall that the On-Demand (OD) [1] mechanism couples the execution of the pending replica updates with the arriving user requests, where all the data items read by a certain query are refreshed on demand before the execution of that query. This mechanism is well suited for key-value data stores where each record is accessed by its key leading to a simple coupling between queries and updates.

Further, the OD mechanism also allows for minimizing the system resources needed to install replica updates in a timeline consistency system like PNUTS. To explain this, recall that in a key-value data store, updates are blind operations that do not require reading the current value of a record before updating it. Hence, in a single-master system like PNUTS, all replicas will go through the same sequence of blind updates such that they will eventually converge to the latest update made by the application. Accordingly, the arrival of a new update to a certain record will make any pending update to that same record worthless as in the Thomas Write Rule [21]. That is, a replica can converge simply by applying the newest update skipping any intermediate ones.

Given the advantages of the OD mechanism, we have decided to further investigate its underlying scheduling policies. In general, under OD, queries are always given precedence over updates. However, when a query $Q_i$ encounters a stale data object $O_i$, the update queue will first be checked if there is a pending update to $O_i$ (i.e., $U_i$). If an update is found, it is applied before executing the query. This provides an attractive property which is maximizing the freshness of data by applying any pending relevant updates first, which results in almost no penalty for data staleness in our system. However, in terms of QoS, the On-Demand mechanism suffers from a major drawback as it employs a basic First-Come-First-Served (FCFS) policy where the arrival time of query $Q_i$ determines its priority. FCFS has been shown to perform very poorly under deadline-based metrics such as tardiness [4], which leads to high QoS penalties for our system that are expected to overweight the gains from improving QoD provided by the OD mechanism. Hence, we propose extending OD with a set of priority-based scheduling policies that are well known for performing reasonably well under deadlines.

For all of those policies, for each pending query $Q_i$ we compute a priority $V_i$ based on some of the properties of $Q_i$. For the query with the highest priority, we first apply the pending update (if any) to the data item $O_i$, then execute the query as in the On-Demand mechanism. We first start with the FCFS-Q policy.

FCFS-Q: First-Come-First-Served (FCFS) has been proposed as the scheduling policy under the OD mechanism [1]. Under FCFS, each query $Q_i$ is assigned a priority $V_i = \frac{1}{A_i}$, where $A_i$ is the arrival time of query $Q_i$ as described in Section 2. FCFS is a fair scheduling policy since it bounds the waiting time of a query in the system queue. However, this is often at odds with minimizing system performance metrics such as response time or tardiness.

EDF-Q: Earliest Deadline First (EDF) is one clear alternative for replacing FCFS under the OD mech-
anism. Under EDF, each query \( Q_i \) is assigned a priority \( V_i = \frac{1}{D_i} \), where \( D_i \) is the tardiness deadline of query \( Q_i \) as described in Section 2. It has been shown that EDF provides a close to zero tardiness under low to medium system utilization which makes it attractive for web database during periods of high load.

**WSJF-Q:** Weighted Shortest Job First (WSJF) is another alternative under the OD mechanism as it considers both the query processing time and its weight. Here, we only need to consider the fraction of weight pertaining to QoS (i.e., \( \alpha_i W_i \)) since the QoD component of weight is already maximized under the OD mechanism. Hence, under WSJF-Q, each query \( Q_i \) is assigned a priority \( V_i = \frac{1}{\sqrt{\alpha_i W_i}} \), where \( \alpha_i W_i \) is the QoS weight component and \( C_{iq} \) is processing of query \( Q_i \), as described in Section 2. It has been shown that when \( C_{iq} \) is empty, the system starts executing updates until a high system utilization as opposed to EDF which might exhibit a "domino effect" [9]. This makes WSJF especially attractive for web database during periods of high workload which is expected to be the norm for the applications we are considering in this work.

**Density-Q:** The density policy is very similar to the WSJF-Q except that it considers the query benefit (or penalty) at the current time rather than its weight [12]. As in WSJF-Q, we only need to consider the fraction of penalty to pertaining QoS since the QoD component of penalty is already minimized under the OD mechanism. Hence, under Density-Q, each query \( Q_i \) is assigned a priority \( V_i = \frac{C_{iq}}{C_{iu}} \), where \( C_{iq} \) is the current time where a scheduling decision is to be made and \( \tau \) is the current time where a scheduling decision is to be made and \( \alpha_i W_i \) is the QoS weight component and \( C_{iq} \) is processing of query \( Q_i \), as described in Section 2. It has been shown that when \( C_{iq} \) is empty, the system starts executing updates until a high system utilization as opposed to EDF which might exhibit a "domino effect" [9]. This makes WSJF especially attractive for web database during periods of high workload which is expected to be the norm for the applications we are considering in this work.

In the previous section, we have applied two features of the On-Demand approach, namely:

1. Applied any pending update to a data object before it is accessed by a query, and
2. Employed scheduling policies that only consider the properties of pending queries.

The first feature above enforces the On-Demand mechanism where updates are applied when an object is accessed leading to fresh data. Meanwhile, the second feature simplifies the scheduling decision by restricting the priority functions to only the query parameters. However, exploiting only the query parameters in scheduling might have a serious negative impact on the system performance.

In particular, all the policies presented above are oblivious to the properties of updates which might be in conflict with the properties of the corresponding query. For instance, under the WSJF-Q, if a query \( Q_i \) has the lowest processing cost then it might be selected for execution first regardless of the cost for refreshing data object \( O_i \) (i.e., \( C_{iq} \)). If that cost of installing the update happened to be very high, then all pending queries will be delayed and accumulating tardiness resulting in a poor overall system performance.

To avoid such conflict, we propose an *Update-aware* strategy, which works like the original On-Demand but employs scheduling policies that consider the characteristics of updates in addition to those of queries. Before explaining those policies, note that the negative impact of an update on the system is restricted to the QoS perceived by other queries but not on the perceived QoD. In particular, processing a certain update \( U_i \) with cost \( C_{iu} \) leads to delaying the processing of other queries and might lead to an increase in tardiness if those queries are close to their deadlines. However, it has no impact on the QoD under the On-Demand mechanism since data objects are always refreshed before accessed by a query leading to maximum freshness.

Hence, under the Update-aware version, we only need to modify those scheduling policies that consider processing cost, to include the cost of processing an update in addition to that of processing a query. Specifically, the EDF and FCFS policies will remain the same under update-aware, whereas we need new versions of WSJF and Density. For those two policies, for each pending query \( Q_i \), we compute a priority \( V_i \) based on some of the properties of \( Q_i \) and its corresponding update \( U_i \) (if any). For the query-update pair with the highest priority, we first apply the pending update (if any) to the data item \( O_i \), then execute the query as in the On-Demand mechanism.

**WSJF-QU:** Under the update-aware WSJF-QU, each query \( Q_i \) is assigned a priority

\[
V_i = \frac{\alpha_i W_i}{C_{iq} + C_{iu}} \tag{4}
\]

where \( \alpha_i W_i \) is the QoS weight component, \( C_{iq} \) is the cost of processing query \( Q_i \) and \( C_{iu} \) is the cost of refreshing data object \( O_i \) by applying the pending update \( U_i \).

**Density-QU:** Under the update-aware Density-QU, each query \( Q_i \) is assigned a priority

\[
-\frac{W_i \alpha_i \times (\tau + C_{iq} + C_{iu} - D_i)^+}{C_{iq} + C_{iu}} \tag{5}
\]

where \( \tau \) is the current time where a scheduling decision is to be made.
Intuitively, the two policies above consider the negative impact of applying an update in terms of delaying other queries by an amount of time equal to the update cost $C_{iu}$. Further, the Density-QO policy also considers the negative impact of an update $U_i$ on its own query $Q_i$, since waiting until an update is installed might lead to $Q_i$ missing its tardiness deadline, which results in QoS penalty to the system.

### 3.3 FIT Scheduling

The On-Demand mechanism for scheduling asynchronous updates defers applying an update as much as possible (i.e., until a query request is about to access a stale data object). For “blind” updates, this allows for saving system resources that otherwise would have been unnecessarily wasted on installing intermediate updates. However, it is often the case that applying the most recent update is not that necessary. This occurs under different conditions such as when the staleness of a data object is within the query’s tolerance or in the extreme case when a query actually does not assign any weight to the QoD.

Even when the staleness violates the query requirement, applying an update might require high processing cost that will have a negative impact on the tardiness of that query and all the other pending queries on that node leading to an overall lower QoS. Towards this, we propose a new Freshness/Tardiness aware mechanism called FIT for the scheduling of queries and updates.

FIT, like OD, defers refreshing an object until it is requested by a query. However, under FIT, the scheduling policy reasons about the global impact of applying the update in terms of the utility of processing that update to the query under consideration as well as the other queries in the system. Before describing the details of our scheduling policies under FIT, we first introduce the general mechanism shared by all those policies. Specifically, under FIT, for each pending query $Q_i$, we compute two priorities:

1. $v^+_i$: The priority of $Q_i$ if it is executed together with the latest corresponding update $U_i$ (if any), and
2. $v^-_i$: The priority of $Q_i$ if it is executed while skipping $U_i$.

Finally, $Q_i$’s priority $V_i$ is computed as:

$$V_i = \max(v^-_i, v^+_i)$$

For the query with the highest priority $V_i$, if $v^+ > v^-$, then first apply the pending update (if any) to the data object $O_i$, then execute the query. Otherwise, the query will directly access the stale data object $O_i$ and the pending update $U_i$ will not be removed from the updates queue.

In order to understand the intuition underlying each of the next scheduling policies, recall that the priority $v^+_i$ corresponds to only a QoS penalty as represented in Figure 1, whereas a priority $v^-_i$ corresponds to a combined QoS and QoD penalty as represented in Figure 3. Hence, the $v^+_i$ priority under all scheduling policies is the same as their counterparts under the update-aware mechanisms, whereas the $v^-_i$ priority should reflect the impact of skipping an update in those cases where it is more beneficial than applying it (i.e., $v^-_i > v^+_i$).

Finally, note that measuring the impact of skipping or applying an update pertains only to those scheduling policies with priority functions that can capture that impact, namely, WSJF and Density, but not the EDF or FCFS policy.

In order to compute $v^-_i$, we need to consider the impact of an update on both QoS and QoD. From Figure 3, we notice that combined penalty is a function in time with two critical points: 1) tardiness deadline, and 2) staleness deadline. In particular, assume the case in Figure 3 where $D_i < S_i$, then the penalty is zero up to time $\tau = D_i$, then it increases linearly with slope $\alpha_i \times W_i$ reflecting the penalty incurred by the system for not meeting the tardiness deadline. This slope stays the same up until time $\tau = S_i$, where the slope increases to be $W_i$ reflecting the combined penalty for both staleness and tardiness. This slope remains constant until the query is eventually answered. In the opposite case where $D_i > S_i$, the penalty function will have the same general shape except that the first slope will be $(1 - \alpha_i) \times W_i$.

In general, we can represent the penalty as a function (Figure 4) with two deadlines: $D_{i1}$, and $D_{i2}$ and three segments with slopes that have the following values: (a) 0, if $\tau < D_{i1}$; (b) $W_{im}$, if $D_{i1} < \tau < D_{i2}$, and (c) $W_i$, if $\tau > D_{i2}$. $W_{im}$ is the intermediate weight when the query misses one of its deadlines but not both. As such, if $D_i < S_i$, then $W_{im} = \alpha_i \times W_i$ and if $D_i > S_i$, then $W_{im} = (1 - \alpha_i) \times W_i$, whereas $W_i$ is the weight when a query misses both its deadlines as described in Section 2.

**WSJF-FIT:** Under WSJF-FIT, $v^+_i$ is computed similar to the update-aware counterpart, whereas $v^-_i$ is computed based on Figure 4 as explained above.

$$v^+_i = \frac{\alpha_i W_i}{C_{iq} + C_{iu}}, \quad v^-_i = \begin{cases} \frac{W_{im}}{W_{im} + W_i} & \tau \leq D_{i1} \\ \frac{W_i}{W_i + W_{im}} & \tau > D_{i1} \end{cases}$$

By considering the query cost $C_{iq}$ in $v^+_i$, WSJF-FIT, Like WSJF-Q, also captures the negative impact of running a certain query $Q_i$ on the other pending queries on the node. Similarly, it also captures the system loss in QoS if $Q_i$ were to miss its tardiness deadline, which is expressed by the QoS portion of its weight. But in addition to that, it also captures the system loss in QoD if $Q_i$ were to access a stale data object, which is expressed by the QoD portion of its weight.

To reflect the loss in QoS and QoD in $v^-_i$, we simply extended the basic WSJF policy to consider...
two deadlines (Figure 4) instead of one deadline (Figure 1). In general, we can argue that WSJF sets the weights according to the slope of the next critical point. Hence, under WSJF-Q and WSJF-QU (Figure 1), at any time \( \tau \) the weight will have only one value because there is only one critical point. However, under WSJF-FIT (Figure 4), after crossing the first critical point (i.e., \( D_{1}^{+} \)) the weight is updated to reflect the future penalty incurred by the system if the query were to be delayed further, where that penalty is expressed by the slope at the next critical point (i.e., \( D_{2}^{+} \)).

**Density-FIT:** Under Density-FIT, \( v_i^{+} \) is computed similar to the update-aware counterpart, whereas \( v_i^{-} \) is computed to reflect the impact of having two deadlines as defined below:

\[
v_i^{+} = -W_i (\alpha_i \times (\tau + C_{iq} + C_{iu} - D_i)^{+})/C_{iq} + C_{iu},
\]

\[
v_i^{-} = -W_i (\tau + C_{iq} + C_{iu} - D_i)^{+} - (W_i - W_{im}) (\tau + C_{iq} - C_{iu})^{+}/C_{iq}.
\]

By balancing the trade-off between the “cost” and “benefit” of applying a replica update, FIT is able to strike a fine balance between QoS and QoD as we show in the next sections.

4 Experimental Evaluation

**Testbed:** We have created a simulator that implements the different mechanisms and policies discussed in this paper. The simulator takes as an input the system parameters, and generates the queries and updates based on these parameters such as deadlines, processing cost, etc. We have varied the parameters settings and conducted several experiments to test the performance of our proposed mechanisms and policies and compared them to other existing approaches.

**Queries:** For each simulated point, we generated 5000 queries where the data object accessed by each query is generated according to uniform distribution over the range \([1, 100]\). The processing cost \( C_{iq} \) for each query \( Q_i \) depends on the accessed data object and is generated according to a uniform distribution over the range \([10, 50]\) mSec. Each query \( Q_i \) is assigned a tardiness tolerance \( \tau_{is} = k_i \times C_{iq} \), where \( k_i \) is generated uniformly over the range \([1, \bar{k}_{max}]\). Hence, the tardiness deadline \( D_i = A_i + k_i \times C_{iq} \), where we set \( \bar{k}_{max} = 5 \) in our experiments.

Each query \( Q_i \) is also assigned a staleness deadline \( S_i \), which is related to the tardiness deadline of the query (i.e., \( D_i \)). This enables us to control the distance between the 2 deadlines for staleness and tardiness. In our experiments, \( S_i \) is generated using uniform distribution in the range \([D_i + \lambda_i, D_i + \lambda_i] \). In the default setting, \( \lambda_i = -50 \) and \( \lambda_i = +50 \). Note that if \( S_i < R_i \), then we set \( S_i = R_i \). That is, the staleness deadline has to be at least equal to the arrival time of the last unapplied update but not less so that to reflect only positive values of tolerance.

To specify the QoS and QoD requirements, each query is assigned a weight \( W_i \) uniformly distributed over the range \([1,10]\) which represents the importance of that query. The QoS fraction of the weight (i.e., \( \alpha \)) is set in the range \([0.1, \alpha_{max}]\) where in the default setting \( \alpha_{max} = 1.0 \) and the skewness for \( \alpha \)’s zipf distribution is 0.0 (i.e., uniform). The arrival of queries is modeled as a poisson process, where we vary the arrival rate of queries between 10 to 50 queries/sec. Given our distribution for processing costs, an arrival rate of 50 queries/second is equivalent to \( \geq 100\% \) utilization of the replica node.

**Updates:** The processing cost \( C_{iu} \) of each update \( U_i \) is also generated according to a Zipf distribution over the range \([10, C_{max}]\) mSec. Varying the values of \( C_{max} \) and the skewness Zipf allows us to control the impact of updates on the system load. In the default setting, \( C_{max} \) is set to 100 with the default Zipf parameter for skewness \( \theta_i \) set to 0.5 and skewed towards the high-end of the cost range. The arrival of updates is modeled as a poisson process, where we set the arrival rate to 50 updates/sec. Table 2 summarizes our simulation parameters and their default values.

Table 2: Simulation Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Data Objects</td>
<td>100</td>
</tr>
<tr>
<td>Number of Queries</td>
<td>5000</td>
</tr>
<tr>
<td>Data Object Access Cost</td>
<td>Uniform over ([10, 50])</td>
</tr>
<tr>
<td>Update Cost</td>
<td>Zipf over ([10, C_{max}])</td>
</tr>
<tr>
<td>Query Arrival Rate</td>
<td>(50)</td>
</tr>
<tr>
<td>Update Arrival Rate</td>
<td>(50)</td>
</tr>
<tr>
<td>Query Deadline Parameter</td>
<td>(k_{max} = 5)</td>
</tr>
<tr>
<td>Importance Weight</td>
<td>Uniform over ([1, 10])</td>
</tr>
<tr>
<td>QoS fraction (\alpha)</td>
<td>Uniform over ([0.1, 1.0])</td>
</tr>
</tbody>
</table>

Table 3: Mechanisms and Policies

<table>
<thead>
<tr>
<th>Query-aware</th>
<th>Update-aware</th>
<th>FIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCFS</td>
<td>FCFS-Q</td>
<td>FCFS-Q</td>
</tr>
<tr>
<td>EDF</td>
<td>EDF-Q</td>
<td>EDF-Q</td>
</tr>
<tr>
<td>WSJF</td>
<td>WSJF-QU</td>
<td>WSJF-QU</td>
</tr>
<tr>
<td>Density-FIT</td>
<td>Density-QU</td>
<td>Density-FIT</td>
</tr>
</tbody>
</table>

**Algorithms:** Table 3 summarizes the mechanisms and policies discussed in this paper and simulated in our experiments. A blank entry in Table 3 entails that the corresponding policy is not applicable under the mechanism.

Additionally, we have also included the QUTS policy [16]. QUTS prioritizes the scheduling of updates and queries using a two-level scheduling scheme that dynamically allocates CPU resources to updates and queries according to user preferences on QoS (tardiness) and QoD (staleness). To implement QUTS under our model, we have changed the hard deadlines to be soft deadlines and changed the QoS and QoD functions to our tardiness and staleness metric. We have also replaced the user’s preference on QoS and QoD by \(\alpha_{is}\) and \(\alpha_{id}\). Finally, we have set the the atom time \(\tau\) to 0.01 and the adaptation period to 1 time unit.

5 Experimental Results

5.1 Impact of Query Arrival Rate

In this experiment, we set all the parameters to the default values mentioned in the previous section. We varied the query arrival rate from 5 queries/second to 50 queries/second. Under our setting for query processing costs, an arrival rate of 50 queries/second will bring the node up to a utilization around 100%.

Figure 5 shows the penalty incurred by the system when applying different strategies for scheduling queries and updates. The schedulers included in this
cause of the mentioned domino effect.

penalty increased significantly at high utilization be-
vided the lowest penalty at low utilization but that
performance of EDF came as expected where it pro-
OD) exhibits the highest penalty. Meanwhile, the
increasing utilization). However, FCFS (employed by
5 shows that in general, for all schedulers the penalty
figure are the query-aware ones presented in Section 3
in addition to QUTS as described in Section 4. Figure
5 that in general, for all schedulers the penalty
increases with increasing the query arrival rate (i.e.,
increasing utilization). However, FCFS (employed by
OD) exhibits the highest penalty. Meanwhile, the
increasing utilization). However, FCFS (employed by
increases with increasing the query arrival rate (i.e.,
5 shows that in general, for all schedulers the penalty

Additionally, Figure 5 also shows that at high
query arrival rate, the performance of QUTS is very
similar to that of Density, whereas WSJF outper-
forms them both. Specifically, at the arrival rate of 50
queries/sec, WSJF-Q reduces the system penalty by
35% compared to QUTS, 33% compared to Density-
for WSFJ-QU. These gains are further de-
penalty by 67% compared to WSFJ-Q and by 22%
compared to WSFJ-QU. These gains are further de-
Figure 7 shows the tardiness penalties (i.e., loss
in QoS) for the WSJF policies. WSJF-FIT exhibits
the lowest loss in tardiness since it might selectively
decide to skip some updates if the benefit of an up-
date does not justify its cost, thus saving resources
that might be needed by other queries and updates. Figure 7 shows that at 50 queries/sec, WSJF-FIT re-
duces the tardiness penalty by 37% vs. WSJF-Q and
18% vs. WSFJ-QU.

As expected, the gains provided by WSJF-FIT in
reducing the tardiness penalty come at the expense of
an increase in the staleness penalty as shown in
Figure 8. The figure shows that by skipping some
updates, WSJF-FIT increased the staleness penalty
compared to both WSFJ-Q and WSFJ-QU. However,
these losses are countered by higher gains in terms
of reducing the tardiness penalty leading to striking
a fine balance between QoS and QoD.

5.2 Impact of different QoS and QoD prefer-
ences

To further illustrate the trade-off between QoS (i.e.,
tardiness) and QoD (i.e., staleness), in this experi-
ment we keep the same default values as in the pre-
However, under this setting the reduction in penalty provided by WSJF-FIT vs. WSJF-QU is only 31% (in comparison to 53% in Figure 9). The reason for that closer gap in performance is that under this setting, WSJF-QU (being update-aware) will also recognize those updates with high processing costs and give them lower priority to favor queries and updates with lower costs. Similarly, WSJF-FIT will recognize those expensive updates and will either skip them (if they have low benefit) or give them low priority (if they have high enough benefit to balance the high cost). This trade-off is further illustrated in Figures 13 and 14, where we break down the penalty incurred by the system into its two components of tardiness and staleness.

### 6 Conclusions and Future Work

Motivated by the need for providing guarantees on both query performance and data currency in highly distributed data management platforms, we addressed the problem of scheduling queries and updates to strike a fine balance between QoS and QoD. Towards this, we presented three mechanisms for the scheduler implementation together with scheduling policies that work in conjunction with those mechanisms. Our experimental results show the the FIT mechanism introduced in this paper, together with the WSJF-FIT policy can efficiently allocate the available resources across queries and updates to maximize the system utility.

While our proposed scheduler is designed to operate at the node-level, in the future we plan to investigate global solutions that work at the system-level for achieving further improvements in performance. We are also planning to investigate advanced query/update scheduling policies that dynamically adapt to the workload and provide the best performance under both low and high utilizations.
Figure 13: Average Tardiness of WSJF

Figure 14: Average Staleness of WSJF

Acknowledgements: We would like to thank the anonymous reviewers for their thoughtful and constructive comments. We also highly appreciate the feedback we have received from Panos K. Chrysanthis and the students of the CS 3551 seminar course at the Univ. of Pittsburgh.

References


Discovering Conditional Functional Dependencies in XML Data

Loan T.H Vo, Jinli Cao, Wenny Rahayu
Department of Computer Science Engineering
La Trobe University
Melbourne, Australia
t7vo@students.latrobe.edu.au, (J.Cao, W.Rahayu)@latrobe.edu.au

Abstract
XML data inconsistency has become a serious problem since XML was widely adopted as a standard for data representation on the web. XML-based standards such as OASIS, xCBL and xBRL have been used to report and exchange business and financial information. Such standards focus on technical rather than semantic aspects. XML Functional Dependencies (XFDs) have been introduced to improve XML semantic expressiveness. However, existing approaches to XFD discovery that have been proposed mainly for enhancing schema design are not capable of dealing with data inconsistency. They cannot find a proper set of semantic constraints from the data, and thus are insufficient for capturing data inconsistency. In this paper we propose an approach, called XDIscover, to discover a set of minimal XML Conditional Functional Dependencies (XCFDs) from a given XML instance to improve data consistency. The XCFD notion is extended from XFDs by incorporating conditions into XFD specifications. XCFDs can be used to constrain data process and also to detect and correct non-compliant data. XDIscover incorporates pruning rules into discovering process to improve searching performance. We present several case studies to demonstrate the effectiveness of our approach.

Keywords: Discovering data rules, data inconsistencies, data quality, XML data.

1 Introduction
Extensible Markup Language (XML) has become a standard for representing data on the web. XML-based standards, such as OASIS, xCBL and xBRL have been introduced for reporting and exchanging business and financial information (Lampathaki et al. 2008). However, such standards only provide schema document frameworks for preparing reports and exchanging data. Most XML-based standards do not address the semantics of underlying business information. Constraints on the underlying data from different organizations are often violated.

In other words, these constraints satisfied by an individual data source may not be applicable in the federated data. Under such circumstances, deriving a complete set of constraints from a given data instance to constrain the heterogeneous data sources is necessary for improving data consistency. Although XML functional dependency (XFD) is one type of semantic constraint, existing notions of XFD (Vincent et al. 2004, Fan 2005, Arenas 2006) are not sufficient in capturing data inconsistency. This is because XFDs globally express data constraints over the whole document; and thus, they are unable to capture conditional semantics partially expressed in some fragments of the document.

Figure 1: An example of Booking Schema Tree

![Diagram](image)

Figure 1: An example of Booking Schema Tree

Copyright (c)2011, Australian Computer Society, Inc. This paper appeared at the 22nd Australasian Database Conference (ADC 2011), Perth, Australia. Conferences in Research and Practice in Information Technology (CRPIT), Vol. 115. H. T. Shen and Y. Zhang, Eds. Reproduction for academic, not-for profit purposes permitted provided this text is included.

Type 1: Constraints without conditions
Constraints without conditions contain only variables. They are data constraints hold over the whole document and are commonly known as Functional Dependencies (FD). For example,

Constraint 1: Any Booking with the same Flight (including Departure and Arrival) should have the same Tax.
Constraint 1 is an example of a FD holding for all Bookings in D.

**Type 2: Constraints with conditions**

This type includes constraints which either contain constants only or both constants and variables. Such constraints hold conditionally on the document. They are not standard FDs. For example,

*Constraint 2a:* Any Booking with Carrier of *Qantas* having the same Price should have the same Tax.

*Constraint 2b:* Any Booking with Carrier of *Virgin* and Arrival of *BNE* has a Tax of 20.

Constraints 2a and 2b are supposed to hold for Bookings with Carrier of *Qantas* or for Bookings with Carrier of *Virgin* and Arrival of *BNE* respectively. They refine Constraint 1 by binding particular values to elements in the constraints e.g. *Qantas* or *Virgin*, *BNE* and 20 for Carrier, Arrival and Tax respectively. Constraints of type 2 are very common in real data, especially for data from multiple sources that use XML-based standards. Each constraint holds only on a subset of a document containing data from one particular source.

To capture data inconsistency, we need to enforce constraints of type 2. This is because when constraints with conditional semantics are not enforced explicitly, the data inconsistency in some parts of document cannot be detected. By *data inconsistency*, we mean XML data violates certain constraints. For example, Bookings data in D (Figure 2) do not satisfy all above constraints. The Bookings of nodes (12, 1) and (22, 1) contain the same values of Flight including Departure (DRW) and Arrival (BNE) and have the same value of Tax (30). They satisfy constraint 1 but violate either constraint 2a or constraint 2b. For constraint 2a, if Carrier is *Qantas*, the Price determines the Tax. Node (12, 1) and node (2, 1) have the same Price (200) but they contain different values of Tax (30 and 40 respectively) that violate constraint 2a. According to constraint 2b, if a Booking with Carrier of *Virgin* and Departure of *BNE*, Tax should be 20 but node (22, 1) contains the Tax of 30 that violates constraint 2b. We can see that if constraint 2a and 2b are not enforced, the inconsistency of node (12, 1) and node (22, 1) cannot be identified.

In this paper, we propose a novel approach to improve XML data consistency. Our contributions are twofold. Our first contribution is to introduce a new notion of XCFDs as constraints of type 2 which revise XFDs by incorporating conditions into XFD specifications. This overcomes limitations of the previous work in two aspects: (i) XCFDs can express constraints in the hierarchical structure in XML data, as opposed to Conditional Functional Dependencies in relational databases; (ii) XCFDs are more powerful than XFDs in terms of capturing data inconsistency. This is because XCFDs allow binding specific constants to particular elements. They cover more situations of dependencies under some conditions. Additionally, we propose an approach, named XDiscover, to search for a set of minimal XCFDs from a given instance. XDiscover also incorporates a set of pruning rules in discovery process. Our purpose is to reduce the searching lattice and the number of XCFD candidates to be checked on the dataset to improve searching performance.

Discovered XCFDs can be embedded as an integral part in an enterprise’s systems to constrain the data process to minimize the data inconsistency. Our approach also can be used within the process of data quality management to detect and correct non-compliant data.

The rest of paper is organized as follows. In section 2, we review related work and discuss the limitations of existing approaches including CFDs in relational databases and XFDs. Section 3 presents preliminary definitions which are necessary for introducing XCFDs in section 4. The detailed approach is described in section 5. The case studies and comparative evaluation are presented in section 6, and conclusions of our work are discussed in section 7.

## 2 Related Work

The problem of data inconsistency has been extensively studied for relational databases. The notion of Conditional Functional Dependencies (CFDs) (Bohannon et al. 2007) has been widely used as a technique to
detect and correct data inconsistency. Many approaches (Chiang and J. Miller 2008, Fan et al. 2009, Golab et al. 2008) have been proposed to automatically discover CFDs from data instances. Despite facing similar problems with relational counterparts, the existing approaches of CFDs cannot be applied directly into XML data for several reasons.

Firstly, there are significant differences in the data structures and the nature of constraints. For relational databases, each object is defined by a single row. Discovering CFDs from data stored in tables has a clearly defined structure. By contrast, XML data has a hierarchical structure and constraints often involve elements from multiple hierarchical levels. Identifying XML data constraints faces challenges that are not encountered in discovering CFDs.

Secondly, different notions of equality are used for constraints. Whereas relational equality simply is the equality of values, the equality of two objects in XML has to be compared according to both structure and data (Yu and Jagadish 2006). Finally, relational CFD discovery algorithms cannot scale well when the XML data structure is complex. This is because applying these algorithms to XML data requires an XML document be transformed into a single relational table. When the structure of schema is complex, the number of attributes in transformed relation is large. The number of tuples also increases multiplicatively when the XML document contains data with complex data type (e.g. maxOccurs in XML Schema). For example, if each Booking contains two Flights, then the number of tuples in transformed relation would double.

Most existing work in XML data (Grahne and Zhu 2002; Hartmann and Link 2003, Yu and Jagadish 2008, Lv and Yan 2006, Trinh 2008) focuses on introducing XFDs to improve schema design. Such approaches are insufficient to capture data inconsistency. This is because XFDs are semantic constraints on the whole document whereas XML data is often obtained by integrating data from different sources with conditional semantics. Discovery algorithms to XFDs cannot find a proper set of constraints. Therefore, it is necessary to introduce new notions of XML data constraints with conditional semantics.

Some previous work (Flesca et al. 2003, Tan et al. 2007, Flesca et al. 2005) has addressed the XML data inconsistency problems. They assume that the set of constraints is known and they only focus on finding consistent parts from inconsistent XML documents w.r.t predefined constraints. In fact, discovering data constraints from data instances manually is a tedious process which requires extensive searching and is time consuming. As XML data becomes more common and more complex in the data structure, it is necessary to develop a formal approach that automatically discovers semantic constraints. To the best of our knowledge, no existing work addresses such problems yet.

XFDs are formally defined in two perspectives (Vincent et al. 2004, Arenas 2006, Hartmann and Link 2003) which are path-based and tree tuple-based approaches. These approaches cannot express the semantics of constraints with a set of complex elements as in our cases. Only the notion of Generalized tree-based XFDs (Yu and Jagadish 2008) is closest to ours. However, this approach adopts tuple-like semantics which requires XML data to be stored in a set of tables and cannot express constraints on the XML data tree directly. No existing XFD definitions can be used to fully extend to our XCFD notion.

To facilitate our approach, we first introduce necessary preliminaries in the next section.

3 Preliminaries

In this section, we present the background and some definitions such as XML schema tree, data tree, data–schema conformance and node-value equality.

We use XPath expression to form a relative path: “..” (self); select the context node. “../”: select the descendants of the context node. For example, //Carrier: select Carrier descendants of the context node Booking; //Flight/Departure: select all Departure elements which are children of Flight.

In this paper, we consider an XML schema or an instance as rooted-unordered-labelled trees, referred to as a schema tree or a data tree, respectively. Each element node is followed by a set of element nodes or a set of attribute nodes. For the instance, the element node can be terminated by a text node.

We give formal definitions for an XML schema tree and an XML data tree as follows:

**Definition 1.** (XML schema tree)

An XML schema tree is defined as $S = (E, A, T, root)$, where:

- $E = E_1 \cup E_2$: a finite set of element nodes in $S$ in which each node is associated with frequency label of $?, +, *, 1$; For every node $e_i$ in $E$, the number of nodes from an instance mapped to $e_i$ is at most one if node $e_i$ has frequency label $?$, exactly one if $e_i$ has a frequency either label 1 or no label at all; at least one if node $e_i$ has frequency label $*$; and undefined occurrences if $e_i$ has a frequency label $+$. $E_1$ is a set of complex nodes; $E_2$ is the set of simple nodes.

- $A$ is a finite set of attribute nodes; attribute nodes only appear as leaf nodes.

- $T$ is a finite set of node types; for each node $v \in E_1 \cup E_2 \cup A$ $v$ is associated with a data type $t \in T$; $t$ can be a simple data type (e.g. string, int, float) or a complex data type (e.g., the data type represents for the maxOccurs, “choice” and “all” model groups) in XML Schema Language (W3C 2004). An element node is called a simple element node if it is defined with a simple data type. Otherwise it is called a complex node. An attribute node is considered as a simple element node.

- $root$ is the root the schema tree.

For example, the schema tree in Figure 1 is defined as $S = (E, A, T, root)$ where:

- $E = E_1 \cup E_2$; $E_1 = \text{feature}$

- $E_2 = \{\text{Carrier}, \text{Departure}, \text{Arrival}, \text{Price}, \text{Tax}\}$

- $A = \{\emptyset\}$; root$= Bookings$; $T = \{\text{String}, \text{int}, \text{Booking}, \text{Flight}\}$; Booking and Flight are complex data types.
We assign a path-ID to each node in the XML schema tree as shown in Figure 1 in a preorder traversal. Each path-ID is a pair \((order, depth)\); where order is an increasing integer (e.g., 1, 2, 3, ...) which is used as a key to identify the path from the root to a particular node and depth label is the number of edges traversing from the root to the node in the schema tree. The depth of the root is 0; e.g., assigning 0 for /Bookings; 1 for /Books/

**Definition 2.** (XML data tree)

An XML data tree constrained by an XML schema tree \(S = (E, A, T, \text{root})\) is defined as \(D = (V, \text{lab}, \text{ele}, \text{att}, \text{val}, r)\), where:

- \(V\) is a set of nodes in \(D\); each \(v \in V\) consists of a label \(e\) and a node-ID that uniquely identify node \(v\) in \(D\).
- \(\text{lab}\) is a labelling function which maps the set \(V\) to the set \(E \cup A\). Each \(v \in V\), \(v\) is called an element node if \(\text{lab}(v) \in E\); \(v\) is called an attribute node if \(\text{lab}(v) \in A\).
- \(\text{ele}\) is a partial function from \(V\) to a sequence of \(V\) nodes; for each complex element node \(v \in V\), the function \(\text{ele}(v)\) maps \(v\) to a list of element nodes \([v_1, v_2, ..., v_n]\) in \(V\); \(\text{att}(v)\) maps \(v\) to a list of attribute nodes \([v_1', v_2', ..., v_m']\) in \(V\) with distinct labels.
- \(\text{val}\) is a function that assigns values to simple element nodes and attribute nodes. Each node \(v \in V\); \(\text{val}(v)\) is the content of attribute if \(\text{lab}(v) \in A\) or the content of simple node if \(\text{lab}(v) \in E\); \(\text{val}(v) = v\) if \(\text{lab}(v) \in E\).
- \(r \in V\), \(\text{lab}(r) = \text{root}\) is that the unique root node and is labelled with complex data types.

The node-ID in the XML data tree is assigned the same ordering as the path-ID in the XML schema tree. Each node-ID(order, depth) contains values uniquely identifying its position in the data tree.

For example, from Figure 2, we have \(V\) a set of nodes from node(1, 0) through node(38, 2).

\[
\begin{align*}
\text{lab}(\text{node}(1,0)) &= \text{Bookings}; \\
\text{lab}(\text{node}(3,2)) &= \text{Carrier}; \quad \text{val}(\text{node}(2,1)) = \text{Book}; \\
\text{lab}(\text{node}(3,2)) &= \text{Qantas}; \quad \text{ele}(\text{node}(2,1)) = \{\text{Carrier}, \text{Flight}, \text{Price}, \text{Tax}\}.
\end{align*}
\]

From Definition 2, we have the following properties:

1. If \(v_1, v_2 \in \text{ele}(v)\) then \(v_1, v_2\) is called a child node of \(v\).
2. \(\{v(P)\}\) is a set of direct nodes that can be reached following path \(P\) from \(v\), where \(v\) is the path from the root to the node \(v\). The path \(P\) can be single node, e.g. \(\text{root}[\text{root}] = \{\text{all direct children nodes of \text{root}}\}\). If there is only one node in \(\{v(P)\}\), we write \(v(P)\).

In this paper, we assume that the XML data tree is required to conform to the associated XML schema tree. The conformance is defined as follows:

**Definition 3.** (XML data – Schema tree conformance)

An XML data tree \(D = (V, \text{lab}, \text{ele}, \text{att}, \text{val}, r)\) is said to conform to a schema tree \(S = (E, A, T, \text{root})\) denoted as \(D \models S\) if and only if (iff):

- \(\text{lab}(r) = \text{root};\)
- Every node \(v \in V\), \(\text{lab}(v) \in E \cup A\). There is a homomorphism from \(V\) to \(E \cup A\) such that for every pair of mapping nodes \((v, e)\), the node name and the data type are preserved. Figure 2 is an example of the Bookings data tree which conforms to the Bookings schema tree in Figure 1.

Now we introduce a notion of node-value equality which is an essential feature in the definition of XFDS. Two nodes are called node-value equality if two corresponding sub-trees rooted at the two nodes are identical.

**Definition 4.** (Node-value equality) Two nodes \(v_i\) and \(v_j\) in an XML data tree \(D = (V, \text{lab}, \text{ele}, \text{att}, \text{val}, r)\) are node-value equality, denoted \(v_i \equiv v_j\) iff:

- \(v_i\) and \(v_j\) have the same label \(\text{lab}(v_i) = \text{lab}(v_j)\)
- If \(v_i\) and \(v_j\) are both simple element nodes or attribute nodes, then \(\text{val}(v_i) = \text{val}(v_j)\)
- If \(v_i\) and \(v_j\) are both complex element nodes and ele(\(v_i\)) = [\(v_{i1}, ..., v_{in}\)] then ele(\(v_j\)) = [\(v_{j1}, ..., v_{jn}\)] and \(v_{ik} \equiv v_{jk}\) for all \(k\); \(1 \leq k \leq n\) and vice versa.

For example, node(14, 2) and node(24, 2) (in Figure 2) are node-value equality. Because we have:

\[
\begin{align*}
\text{lab}(\text{node}(14,2)) &= \text{lab}(\text{node}(24,2)) = \{\text{Flight}\}; \\
\text{ele}(\text{node}(14,2)) &= \{\text{node}(15,3), \text{node}(16,3)\}; \\
\text{ele}(\text{node}(24,2)) &= \{\text{node}(25,3), \text{node}(26,3)\}; \\
\text{node}(15, 3) &= \text{node}(25, 3) = \text{“DRW”} \text{ and node}(16, 3) = \text{node}(26, 3) = \text{“BNE”}. \\
\end{align*}
\]

The definitions above are basic concepts used in our proposed notions in the next section.

**4 XML Conditional Functional Dependency**

In this section, we present the new notion of XML Conditional Functional Dependency (XCFD). The most important features of XCFDs are path and value-based constraints.

An XCFD is defined as an extension of XFD. Therefore we need to describe a precise XFD definition before introducing XCFDs.

**Definition 5.** (XML Functional Dependency)

Given an XML data tree \(D = (V, \text{lab}, \text{ele}, \text{att}, \text{val}, r)\) conforming to an XML schema tree \(S = (E, A, T, \text{root})\), an XML Functional Dependency over \(D\) is defined as:

\[\phi = P : X \rightarrow Y;\]

where:

- \(P\) is a downward context path starting from the root to a considered node \(v\). The scope of \(\phi\) is the sub-tree rooted at node \(v(P)\).
- \(X, Y\) are non-empty set of paths rooted at \(v(P)\).
- \(\langle X \rightarrow Y \rangle\) indicates a relationship between the nodes in \(X\) and \(Y\), such that for two sub-trees that share the same values for \(X\) must share the same values for \(Y\). That is, the values of nodes following the \(X\) path uniquely identify the values of the nodes following the \(Y\) path. We refer to \(X\) as the antecedent and \(Y\) as the consequent.

A document \(D = (V, \text{lab}, \text{ele}, \text{att}, \text{val}, r)\) conforming to \(S, D \models S\) is said to satisfy \(\phi = P : X \rightarrow Y\) denoted \(D \models \phi \models S\) iff any two nodes \(n_i\) and \(n_j\) in \(D\), if \(\{n_i[X] = n_j[X]\}\) then \(\{n_i[Y] = n_j[Y]\}\).
Let us consider an example, where we suppose $P_{Booking}$ is the context path from the root to the $Booking$ nodes in the $Booking$ data tree (in Figure 2).

$$X = (//Flight/Departure, //Flight/Arrival) \text{ and } Y = (//Price, //Tax)$$

then we have an XFD:

$$\varphi = P_{Booking}; (//Flight/Departure, //Flight/Arrival) \rightarrow (//Price, //Tax).$$

After defining the XFD notion, we propose a novel notion of XCFDs by revising the XFD definition.

**Definition 6.** (XML Conditional Functional Dependency)

Given an XML data tree $D=(V, lab, ele, att, val, r)$ conforming to a schema tree $S=(E, A, T, root)$; an XML Conditional Functional Dependency holding on $D$ is defined as:

$$\psi = P : \{\mathcal{E}, [X] \rightarrow [Y]\},$$

where $\mathcal{E}$ is a condition for the XFD $[X] \rightarrow [Y]$ holds on a subset of $D$. The condition $\mathcal{E}$ has the form: $\mathcal{E} = ex_{1}\theta ex_{2}\theta ... \theta ex_{n}$, where $ex_{i}$ is a Boolean expression associated to a particular data node. “$\theta$” is an operator either $\wedge$ or $\vee$.

For example, suppose we assume that $P_{Booking}$ is the context path from the root to the $Booking$ nodes in the $Bookings$ data tree (Figure 2); if there exists an XFD $([Price] \rightarrow [Tax])$ holding on the Bookings data tree under condition $\mathcal{E} = (//Carrier = "Qantas"$), then we have an XCFD:

$$\psi = P_{Booking}; (//Carrier = "Qantas", //Price) \rightarrow (//Tax).$$

XDiscover only returns minimal XCFDs. Before proposing the XDiscover algorithm, we define the minimal XCFDs as follows.

**Definition 7.** (Minimal XCFDs)

Assume that an XML data tree $D=(V, lab, ele, att, val, r)$ conforms to the XML schema $S=(E, A, T, root)$. An XCFD $\psi = P : \{\mathcal{E}, [X] \rightarrow [Y]\}$ on $D$ is minimal if $\forall \mathcal{E}' \subset \mathcal{E}, \psi' = P : \{\mathcal{E}', [X] \rightarrow [Y]\}$ does not hold on $D$.

Based on aforementioned definitions, we formally define the problem of discovering XCFDs as follows:

**Problem Statement:** Given an XML data tree $D=(V, lab, ele, att, val, r)$ conforming to a schema $S=(E, A, T, root)$; the goal of XDiscover is to discover a set of non-redundant XCFDs in the form $\psi = P : \{\mathcal{E}, [X] \rightarrow [Y]\}$; where each XCFD is minimal and contains only a single path in the consequence $Y$.

The approach of discovering XCFDs, called XDiscover, includes a number of functionalities. We describe the XDiscover algorithm in detail in the next section.

5 XDiscover: Discovering XCFDs

![Figure 3: A set of containment lattice of Carrier (C), Departure (D), Arrival (A), Price (P) and Tax (T)](image-url)
The XDiscover algorithm includes four main functions to discover XCFDs. The first function is to generate partition identifiers and identify candidate XCFDs. The second function is to generate partitions of partition identifiers associated to each candidate XCFD. Such generated partitions are used in the third function to validate for a satisfied XCFD. The last function performs a set of pruning rules to search for minimal XCFDs and remove redundant candidate XCFDs in the next level in the searching lattice.

5.1 Candidate XCFD Identification
To search for all XCFDs, the first function generates a searching lattice which contains all possible combinations of node labels. The process starts from nodes with a single label (level l=1). Node labels in level l with l >= 2 will be obtained from the node labels in level (l-1). Figure 3 is an example of a searching lattice of node labels: C, D, A, P and T (represent for Carrier, Departure, Arrival, Price, and Tax respectively). The node label (CP) in level 2 is generated from nodes C and P in level 1.

Assume that W & Z are two nodes directly linked in the searching lattice; and Z=W \cup \{Y\}. Each edge(W, Z) represents a candidate XCFD $\psi = \{P, C, \{E\}, \{X\} \rightarrow \{Y\}\}$; where $W=\{X\}\cup \{E\}; X$ is a set of variable nodes; $E$ is a set of conditional nodes. For example, for edge(W, Z) = (CP, CPT) in Figure 3, we assume the conditional data node is C, then we have $\psi = \{P_{bookings}, C, P \rightarrow T\}$. If condition $E$ is empty then $\psi$ become a data constraint on the whole document as an XFD. That means an XFD is a special case of an XCFD.

Node labels (e.g., W and Z) in the searching lattice associated with a candidate XCFD are called partition identifiers.

Once the searching lattice has been established, partitions of partition identifiers in each candidate XCFD are generated. The results of partition generation are the input of discovering XCFDs function.

5.2 Partition Generation
The partition generation for each partition identifier classifies a considered node in the data tree into classes based on the node’s values in the partition identifier. Each class contains all elements which have the same values on the partition identifier. Partitions play a vital role in validating a satisfied XCFD on the data tree. In the following, we present a formal definition of partition.

Definition 8. (Partition) A partition $W_{w}$ of W on D under the sub-tree rooted at v is a set of disjoint equivalence classes $w_{i}$. Each class $w_{i}$ in $W_{w}$ contains all nodes with label v having the same values on partition identifier W.

The number of classes in a partition is called cardinality of the partition, denoted $|W_{w}|$; $|w|$ is the number of nodes in class $w_{i}$.

For example, from schema tree Bookings in Figure 1, we have: $E=\{(1, Booking),(2, 1)Booking,(3, 2)Carrier\}, [(4, 2)Flight], [(5, 3)Departure], [(6, 3)Arrival], [(7, 2)Price], [(8, 2)Tax]$. From the searching lattice (Figure 3), suppose we consider a partition identifier $W_{v} \equiv \{\text{Carrier}\}$ which corresponds to the node $\{\text{3, 2 Carrier}\}$ in the schema tree S. Traversing data tree Bookings D in Figure 4 to find all data nodes which have the node name as Carrier and depth of 2.

The found nodes are grouped into two classes:

$\text{Class}_{1} = \{\{(23, 2)\text{ Carrier} = \text{"Qantas"}\}, \{(33, 2)\text{ Carrier} = \text{"Qantas"}\}, \{(53, 2)\text{Carrier} = \text{"Qantas"}\}$

$\text{Class}_{2} = \{\{(43, 2)\text{carrier} = \text{"Virgin"}\}, \{(73, 2)\text{carrier} = \text{"Virgin"}\}$

The partition $W_{\text{Carrier}}$ Booking to the value of node Carrier w.r.t sub-tree rooted at Booking is represented as $W_{\text{Carrier}}$ Booking = $\{W_{w}, W_{v}\}$

$w_{v} = \{(22, 1)\text{Booking}, (32, 1)\text{Booking}, (52, 1)\text{Booking}\}$

$w_{v} = \{(22, 1)\text{Booking}, (32, 1)\text{Booking}, (52, 1)\text{Booking}\}$

![Figure 4: A simplified instance of Bookings data tree: each Booking contains only one Flight](image-url)
Booking], [(62, 1 Booking)]
w_2 = [[(42, 1 Booking), [(72, 1 Booking)]
|PCarrier| Booking| = 2; \(|w_1| = 4; |w_2| = 2.

To simplify the presentation, we omit the node-ID and path-ID associated with each node in following sections to avoid cluttering.

After finding partitions of partition identifiers in a candidate XCFD, a validation function is performed to check whether or not this candidate holds on the data tree.
The detail of this function is in the next subsection.

5.3 XCFD Validation
We still use the same assumptions of \( W \) & \( Z \) in Section 5.1. Validating for a satisfied XCFD is based on the concept of partition refinement defined by Huhuala et al. (Huhuala et al. 1999):

**Definition 9.** (Partition refinement): A partition \( P_W \) refines \( P_Z \) if any nodes in a class \( w_i \) of \( P_W \) are also in a class \( z_j \) of \( P_Z \).

A functional dependency holds on document \( D \) if \( P_W \) refines \( P_Z \) (Yu and Jagadish 2008). Because an XCFD \( \psi = P_{\mathcal{C}, \{X\}} \rightarrow \{Y\} \) holds only on the subset of data tree \( D \), the satisfied XCFD does not require every class \( w_i \) in \( P_W \) be the subset of a class \( z_j \) in \( P_Z \). This means if there exists at least one equivalence pair \( (w_i, z_j) \) between \( P_W \) and \( P_Z \) then \( \psi \) holds on document \( D \).

For general cases, let \( P_W \) be a set of all classes in \( P_W \), where each \( w_i \) in \( P_W \) has a corresponding equivalence class \( z_j \) in \( P_Z \). If there exists a class \( e \) in \( P_E \) which contains exactly all elements of \( P_W \), the class \( e \) will be the condition for an XCFD: \( \psi = P_{\mathcal{E}, \{X\}} \rightarrow \{Y\} \) holds on data tree \( D \).

The number of candidate XCFDs and the searching lattice are very large. Our algorithm uses a set of pruning rules, introduced in the next section, to reduce the number of XCFDs to be checked on the dataset.

5.4 Pruning rules
We introduce rules which are used to (i) skip searching for those XCFDs that are logically implied by the already found XCFDs and (ii) prune redundant XCFD candidates from the searching lattice.

**Rule 1:** Let \( E \) be a set of nodes in the schema \( S \), and \( \psi \) be an XCFD: \( \psi = P_{\mathcal{C}, \{X\}} \rightarrow \{Y\} \). For each \( X \) we only check XCFDs which have consequence \( \{Y\} \) in the remaining set of elements in \( E \); \( Y \in E \setminus \{X\} \cup \{\mathcal{C}\} \).

For example, if \( E = \{D, A, P, T\}, X = \{D\} \), and \( \mathcal{E} = \{A\} \) then only XCFDs contain \( Y \in \{P, T\} \):

\[ \psi = P_{\mathcal{E}, \{A\}}; \{D\} \rightarrow \{P\}; \]
\[ \psi = P_{\mathcal{E}, \{A\}}; \{D\} \rightarrow \{T\} \] to be checked on the data.

**Rule 2:** We Prune supersets of nodes associated with the already discovered XCFDs. Given partition identifiers \( W = \mathcal{X}, \mathcal{E} \& Z = \mathcal{W} \setminus \{Y\} \), and there exists a discovered XCFD \( \psi = P_{\mathcal{C}, \{X\}} \rightarrow \{Y\} \) corresponding to edge \( (W, Z) \) then we remove the supersets of nodes in the edge \((W, Z)\) from the searching space. The reason is that if \( \psi \) holds on data tree \( D \) then any extra label adding to the antecedence of \( \psi \) (e.g. \( Z \)) will produce implied XCFD in the form of \( \psi' = P_{\mathcal{C}, \{X, Y\}} \rightarrow \{Y\}; \psi' \) is redundant and is not minimal.

**Rule 3:** If \( \psi = P_{\mathcal{C}, \{X\}} \rightarrow \{Y\} \) is a minimal XCFD then we do not consider \( \psi' = P_{\mathcal{C}, \{X\}} \rightarrow \{Y\} \) where \( \mathcal{C} \subseteq \mathcal{E} \). This guarantees that the discovered XCFD is minimal. For example, if class \( w_{ij} \) = \{22, 32, 52, 62\} satisfies an XCFD \( \psi = P_{\text{Booking}}, (\text{/Carri} \rightarrow \{\text{Carri} \quad \text{"Qantas"}, \text{/Price} \rightarrow \{/\text{Tax}} \)), \( \mathcal{E} = \{\text{/Carri} = \{\text{"Qanta"} \} \) the condition then we do not consider class \( w_{ij} \) in subsequent candidates \( \psi' \) with condition \( \mathcal{C}' \) = \{\text{/Carri} = \{\text{"Qants"} \} \) on the dataset because we can infer from the discovered XCFD \( \psi \). Therefore, if \( X \) and \( Y \) are equivalent sets, no further XCFDs with antecedent containing \( Y \) need to be considered.

**Rule 5:** Given a threshold \( \tau \), we only consider nodes associated with class \( w_{ij} \) such that \(|w_{ij}| > \tau \), e.g. for \( \tau = 1 \), we do not consider classes containing only one element. The constraint associated with such classes is trivial.

In the following, we present the algorithm of XDiscover.

5.5 Algorithm XDiscover
Listing 1 presents our proposed XDiscover algorithm to find XCFDs from a given data tree \( D \). Our algorithm traverses the searching lattice following a breath-first search manner combining with pruning rules described in section 5.4.

The searching process starts from level 1 \((l=1)\); all nodes from \( E \) are stored in Partition Identifier \( Pl_{l=1} = \{v_1, v_2, \ldots, v_{|E|}\} \) (line 3). Each node in \( E \) is a partition identifier with a single label associated with some candidate XCFDs. Partitions of Partition identifiers are generated and stored in \( GP_{l=1} \) - Generated Partition (line 4). At level \( l > 1 \), node labels are generated from \( Pl_{l-1} \) and stored in \( Pl_l \) (line 7) in the form \( v_{i,l} \); where \( v_{i,l} \neq v_{i-1} \); \( v_{i,l} \in Pl_{l-1}; Pl_{l-1} \) contains node labels at level \((l-1)\); all partitions of \( v_{i,l} \) nodes at level \( l \) are generated and stored in \( GP_{l=1} \). All candidates in the form \( c_{i,l}, c_{j,l} \rightarrow z_j \) are checked; where \( v_{i,l}, v_{j,l} \in c_{i,l}, c_{j,l} \in Pl_l \setminus (\{w_i \cup c\}). The validation for a satisfied XCFD follows the approach described in Section 5.3 (line 9: function DiscoverXCFD in Listing 2). The found XCFDs are stored in Discovered set of XCFDs \( DF \). Then the Prune function including the pruning rules (in Section 5.4) is performed to prune redundant nodes
and edges from the searching lattice for the next level (line 10). The searching process is repeated until no more partition identifiers are considered (line 5). The output of XDiscover is a set of minimal XCFDs.

Function: XDiscover

Input: XML data tree D=(V, lab, ele, att, val, r) schema tree S=(E, A, T, root)

Output: a minimal set of XCFDs
1. DF:= {Ø};
2. Level l:=1;
3. Plj:= E;
4. GPj := GeneratePartition(D, Plj);
5. While [Plj | {Ø}] do
6. l++;
7. Plj := GeneratePartitionIdentifier(GPj);
8. GPj := GeneratePartition(D, Plj);
9. DF := DF \ DiscoverXCFD(GPj, GPj);
10. Prune(GPj);
11. Return(DF).

Listing 1: The XDiscover Function

The function of DiscoverXCFD depicted in Listing 2 searches for XCFDs at each level l. If there still exists classes in Pj,l which do not belong to any discovered XCFD then we continue to consider such classes with additional condition nodes. DiscoverXCFD calls the GenerateAdditionPartition function to calculate partitions with additional condition nodes. The DiscoverXCFD returns XCFDs to XDiscover.

Function: DiscoverXCFD

Input: GPj, GPj+1/partitions at level l and l+1
Output: satisfied XCFDs
1. DF:= {Ø};
2. For each partition of W ∈ GPj do
3. For each partition of Z ∈ GPj do
4. If Z = (W ∪ {Y}) then
5. Ωw := superset w;
6. While [Ωw | {Ø}] do
7. For each class w ∈ Ωw do
8. For each class zj ∈ Hj do
9. If ([w] ∈ T) and ([w] = [zj]) then
10. DF := DF \ ({ω, X→Y});
11. Ωw := Ωw \ (w ∈ (ω, X→Y));
12. If not found XCFD then
13. GenerateAdditionPartition;
14. For each cj ∈ E do
15. If c, contains values only from Ωw then
16. DF := DF \ ({ω, X→Y});
17. Ωw := Ωw \ (w ∈ (ω, X→Y));
18. Return(DF).

Listing 2: The DiscoverXCFD Function

6 Case studies and comparative evaluation

To evaluate and demonstrate the effectiveness of the proposed approach, we use the Flight Booking XML data for our case studies.

From schema Bookings S in Figure 1, we have E={Bookings, Booking, Carrier, Flight, Departure, Arrival, Price, Tax}. We assume τ = 1, which means we only consider XCFDs holding for more than one data node.

Case 1: XCFDs contain only constants.

Suppose the data tree D in Figure 4 conforms to schema Bookings S and each Booking only contains one Flight as shown in D.

Consider edge(W, Z)= edge(CA, CAT) = (Carrier-Arrival, Carrier-Arrival-Tax) in Figure 3. Following the process described in section 5.2 to generate two partitions of Carrier-Arrival and Carrier-Arrival-Tax w.r.t sub-tree rooted at Booking. To simplify the presentation, we omit the node label (e.g. Booking) associated to each node in classes.

\( \Pi_{\text{Carrier}}(\text{Flight}/\text{Arrival}|\text{Booking} = \{\{W_1, W_2, W_4\}\}
\)

\( =\{\{(22,1), (32,1), (42,1), (72,1)\}, \{(62,1)\}\}\)

\( \Pi_{\text{Carrier}}(\text{Flight}/\text{Arrival}|\text{Tax}|\text{Booking} = \{\{Z_1, Z_2, Z_3, Z_4\}\}
\)

\( =\{\{(22,1), (32,1), (42,1), (72,1)\}, \{(52,1), (62,1)\}\}\)

We can see that Wj in \( \Pi_{\text{Carrier}}(\text{Flight}/\text{Arrival}|\text{Booking} \)

is equivalent to Zj in \( \Pi_{\text{Carrier}}(\text{Flight}/\text{Arrival}|\text{Tax}|\text{Booking} - That is, wj = zj = \{(42,1), (72,1)\}\.

Nodes in wj have the same value of Carrier = "Virgin" and Arrival = "BNE". Nodes in zj share the same value of Tax = "20". An XCFD is discovered:

\( W := \text{Pruning}(.//\text{Carrier} = \text{Virgin} ^ { .//\text{Flight}/\text{Arrival} = \text{"BNE"}}) \Rightarrow ()//\text{Tax} = \text{"20"} \).

This case demonstrates the XCFD contains only constants. For each XFD, there might exist a number of conditional dependencies XCFDs which refine this XFD by binding particular values to elements in its specification. Such constraints cannot be expressed by using the XFD notion.

Case 2: XCFDs contain both variables and constants.

Using the same assumption in case 1, considering edge (W, Z)= edge(CP, PT) = (Price, Price-Tax) in Figure 3, two partitions of Price and Price-Tax w.r.t the sub-tree rooted at Booking:

\( \Pi_{\text{Price}}(\text{Booking} = \{\{W_2, W_3\}\} = \{\{(22,1), (32,1), (72,1)\}, \{(42,1), (52,1), (62,1)\}\}
\)

\( \Pi_{\text{Price, Tax}|\text{Booking} = \{\{Z_1, Z_2, Z_3, Z_4\}\} = \{\{(22,1), (32,1)\}, \{(42,1), (52,1), (62,1)\}\}, \{(72,1)\}\}\)

There does not exist any equivalent pair between two partitions \( \Pi_{\text{Price}}(\text{Booking} \)

and \( \Pi_{\text{Price, Tax}|\text{Booking} \). We need to add more data nodes from the remaining set of E\{W \cup Z\}. For example, the node of /Carrieron can be added to edge(P, PT) as a conditional data node. We now consider edge(W, Z)= edge(CP, CPT) = (Carrier-Price, Carrier-Price-Tax). Partitions of Carrier-Price and Carrier-Price-Tax w.r.t sub-tree rooted at Booking are as follows:
\[ H_{\text{Carrier}, \text{Price}, \text{Booking}} = \{ w', z', w'_e \mid \langle 22, 1 \rangle, (32, 1), \langle 42, 1 \rangle, (52, 1), (62, 1) \} \]
\[ H_{\text{Carrier}, \text{Price}, \text{Taxi Booking}} = \{ z', z'_e \mid \langle 22, 1 \rangle, (32, 1), \langle 42, 1 \rangle, (52, 1), (62, 1) \} \]

The partition of the condition node \( \text{/Carrier} \):
\[ H_{\text{Carrier} | \text{Booking}} = \{ e_{\text{p}}, e_{\text{c}}, e_{\text{j}} \} = \{ (22, 1), (32, 1), (52, 1), (62, 1) \} \]

We have two equivalent pairs \((w'_e, z'_e)\) and \((w'_c, z'_c)\) between \(H_{\text{Carrier}, \text{Price}, \text{Booking}}\) & \(H_{\text{Carrier}, \text{Price}, \text{Taxi Booking}}\) with \(|w'_e|=2\) and \(|w'_c|=2 > \tau\). Furthermore, there exists a class \(e_{\text{c}}\) in \(H_{\text{Carrier, Booking}}\) containing exactly all elements in \(\Omega_{\text{c}} = w'_c \cup w'_e\). All elements in class \(e_{\text{c}}\) have the same value at \(\text{Carrier} = \text{"Qantas"}\). This means nodes in classes \(w'_e\) and \(w'_c\) share the same condition \((/\text{Carrier} = \text{"Qantas"})\). Therefore, an XCFD \(\psi = P_{\text{Booking}}(./\text{Carrier} = \text{"Qantas"}, ./\text{Price}) \rightarrow ./\text{Tax} \) is discovered.

Case 2 illustrates techniques to find an XCFD with extra data nodes which are referred to as the condition of the XCFD. Such XCFDs contain both variables and constants.

Case 3: Partition identifiers contain a set of complex nodes.

Suppose data tree \(D\) in Figure 5 conforms to schema Bookings \(S\) in Figure 1. Each Booking contains multiple complex nodes Flight.

For partition identifiers containing a set of complex data nodes, the calculating partitions are processed in a bottom-up fashion. We first consider the sub-tree rooted at the bottom level in the data tree (e.g., Flight) to calculate partitions. Then we convert all classes in each generated partition into the corresponding parent of this complex node (i.e., the parent of Flight is Booking) to find the refinement. We repeat converting the found partition to obtain its refinement until reaching the sub-tree rooted at the considered nodes (i.e., Booking). The validation for a satisfied XCFD is similar to the cases which deal with the partition identifier which contains single data nodes.

Consider edge \((W, Z) = (\text{Flight, Tax})\) w.r.t. sub-tree rooted at Booking, we start generating partitions under the sub-tree rooted at Flight. Following the process described in section 5.2, we partition the nodes according to each Flight (including Departure and Arrival) under the sub-tree rooted at Flight:
\[ H_{\text{Flight/Departure, Flight/Arrival}} = \{ (104, 2), (124, 2), (107, 2), (127, 2) \} \]

Then converting these classes into the Booking sub-tree, we have a refinement: \(H_{\text{Flight} | \text{Booking}} = \{ (102, 1), (122, 1) \}\). Validating for a satisfied XCFD is done similarly to case which partition identifiers contain only single data nodes. The discovered XCFD is represented in the form:
\[ \psi' = P_{\text{Booking}}(./\text{Carrier} = \text{"Virgin"}, ./\text{Flight}) \rightarrow ./\text{Tax} \]

where \((./\text{Flight})\) represents a set of complex data nodes Flight (including Departure and Arrival). In case there is only one Flight node in the constraint, the XCFD can be represented as:
\[ \psi'' = P_{\text{Booking}}(./\text{Carrier} = \text{"Virgin"}, ./\text{Flight}) \rightarrow ./\text{Tax} \]

\(\psi''\) is a special case of \(\psi'\). Generally, a partition identifier containing simple nodes is a special case of the partition identifier containing complex nodes. Therefore, we apply the same process to deal with the partition identifiers which contain complex nodes for both cases.

Comparative Evaluation: There does not exist any approach to discover Conditional Functional Dependencies in XML data. The only algorithm for discovering XFDs (Yu and Jagadish 2006) is close to ours. However, the existing algorithm discovers XFDs containing only variables and can not detect for dependencies which hold partially on documents with conditions. Our approach discovers constraints containing both variables and constants, or either variables or constants that allows the detection of more interesting semantic constraints than algorithms to discover XFDs.

Figure 5: An example of Bookings data tree; each Booking contains a set of complex element Flight.
7 Conclusion

This paper addressed the issues of XML data inconsistency. We pointed out the limitations of existing work in handling such problems, and introduced the notion of XML Conditional Functional Dependency which incorporates conditions into data constraints. We proposed the XD discover algorithm to detect a set of possible XCFDs on a given XML data instance. Our approach employs the set of pruning rules to reduce the searching space and the number of XCFDs to be checked on the dataset. Our approach can be used to enhance data quality management by suggesting possible rules and identifying non-compliant data. Discovered XCFDs also can also be embedded into an enterprise’s systems as an integral part to support manipulating data. XML data changes very often which may lead to a corresponding change in the semantics of data constraints. Therefore, our work can be further extended to address the problem of data evolution.

References


Yu, C. and Jagadish, H. V. (2006): Efficient Discovery of XML Data Redundancies. VLDB.
20 Years of Data Quality Research: Themes, Trends and Synergies

Shazia Sadiq Naiem Khodabandehloo Yeganeh Marta Indulska

School of Information Technology and Electrical Engineering
1 Business School,
The University of Queensland,
Email: {shazia,naiem}@itee.uq.edu.au, mindulska@business.uq.edu.au

Abstract

Data Quality is a cross-disciplinary and often domain specific problem due to the importance of fitness for use in the definition of data quality metrics. It has been the target of research and development for over 4 decades by business analysts, solution architects, database experts and statisticians to name a few. However, the changing landscape of data quality challenges indicate the need for holistic solutions. As a first step towards bridging any gaps between the various research communities, we undertook a comprehensive literature study of data quality research published in the last two decades in this study we considered a broad range of Information System (IS) and Computer Science (CS) publication (conference and journal) outlets. The main aims of the study were to understand the current landscape of data quality research, to create better awareness of (lack of) synergies between various research communities, and, subsequently, to direct attention towards holistic solutions. In this paper, we present a summary of the findings from the study, that include a taxonomy of data quality problems, identification of the top themes, outlets and main trends in data quality research, as well as a detailed thematic analysis that outlines the overlaps and distinctions between the focus of IS and CS publications.

Keywords: Data Quality, Literature Survey, Research Framework

1 Introduction

The issue of data quality is as old as data itself. The impact of poor data quality on decision making, organizational trust and customer satisfaction is well known. However, the changing nature and increasing volume of data has exacerbated the problem to a strategic level and, thus, increased manifold the stakes involved. Data quality management is complicated by several emerging factors. First, there are clear implications that relate to the sheer volume of data produced by organisations today. Second, recent years have seen an increase in the diversity of data. Such diversity refers to structured, unstructured, semi-structured data, and multi-media data such as video, maps, images, etc. Data also has an increasing number of sources. The use of various technologies, for example, sensor devices, medical instrumentation, RFID readers, etc, further increases the amount and diversity of data being collected. More subtle factors also exist - such as the lack of clear alignment between the intention of data creation and its subsequent usage. A prime example of such lack of alignment is the vast amount of data collected from social networks that can then be used, without assessment of quality, as a basis for marketing decisions. Accordingly, a related factor exists that relates to difficulties in defining appropriate data quality metrics.

As these changes occur, traditional approaches and solutions to data management in general, and data quality control specifically, are challenged. There is an evident need to incorporate data quality considerations into the whole data cycle encompassing managerial/governance as well as technical aspects. Currently it can be observed that contributions from research and industry into data quality has been from three distinct contributing communities:

• Business Analysts, who focus on organizational solutions. That is, the development of data quality objectives for the organization, as well as the development of strategies to establish roles, processes, policies, and standards required to manage and ensure the data quality objectives are met.

• Solution Architects, who work on architectural solutions. That is, the technology landscape required to deploy developed data quality management processes, standards and policies.

• Database Experts and statisticians, who contribute to computational solutions. That is, effective and efficient IT tools, and computational techniques, required to meet data quality objectives. Techniques in this regard can include record linkage, lineage and provenance, data uncertainty, semantic integrity constraints, as well as information trust and credibility.

For the research community to adequately respond to the changing landscape of data quality challenges, a unified framework for data quality research is needed. Such a framework should acknowledge the central role of data quality in future systems development initiatives and motivate the exploitation of synergies across diverse research communities.

It is unclear if synergies across the three contributing communities have been fully exploited. We argue that a unified framework for data quality management should bring together organizational, architectural and computational approaches proposed from the three communities respectively. As a first step towards bridging any gaps between the various research communities, we undertook a comprehensive
literature study of data quality research published in the last two decades. In this study we considered a broad range of Information System (IS) and Computer Science (CS) publication (conference and journal) outlets so as to ensure adequate coverage of organizational, architectural and computational contributions. The main aims of the study were to understand the current landscape of data quality research, to create better awareness of (lack of) synergies between various research communities, and, subsequently, to direct attention towards holistic solutions that span across the organizational, architectural and computational aspects (thus requiring collaboration from the relevant research communities).

In this paper, we present a summary of the findings from the study. In the following section we first discuss some related studies. We then present the methodology employed for the study, followed by a discussion of the key results. We also provide a taxonomy of data quality problems that emerged from the analysis of current research, and identify the top themes, outlets and main trends in data quality research.

2 Related Studies

This is not the first study outlining the contributions on data and information quality research. A number of significant works have addressed this issue in the past. Owing to the cross-disciplinary needs of this area, identifying the central themes and topics and correspondingly the associated methodologies has been a challenge. Recent work by [Madnick et al., 2009] has presented a framework that characterizes data quality research along the two dimensions of topics and methods thereby providing a means to classify various research works. Previous works have also assisted by developing frameworks through which data quality research could be characterized, including a predecessor framework by the above group [Wang et al., 1995] that analogized data quality processes with product manufacturing processes. Some key research aspects such as data quality standardization, metrics/measurements and policy management emerged from these earlier works.

Other more recent studies have also provided valuable means of classification for data quality research. [Ge and Helfert, 1996] have structured their review of the literature as IQ Assessment, IQ Management and Contextual IQ. [Lima et al., 2006] classify the literature between theoretical (conceptual, applied, illustrative) and practical (qualitative, experimental, survey, simulation) aspects. Further [Neely and Cook, 2008] present their classification as a cross-tabulation of Wang’s framework [Wang et al., 1995] and Juran’s original fitness for use factors [Juran, 1962]. The above studies provide various angles through which the body of knowledge can be classified and thus provide an essential means of understanding the core topics of data quality. However, understanding the intellectual corpus of a discipline requires not only an understanding of its core, but also its boundaries [Benbasat and Zmud, 2003]. As the realm of data quality has grown, so has the scope of its reference disciplines. With these factors in mind, we focused our study on understanding the interconnections and synergies across the various communities that contribute to data quality, rather than an identification of its central themes. We argue that addressing the current challenges in data quality warrants such an understanding so synergies would be better exploited and holistic solutions may be developed.

Towards this end, we have identified a corpus of publication outlets that span across the Information Systems (IS) and Computer Science (CS) disciplines. In the next section we present details of our selection and the methodology employed to conduct the analysis.

3 Methodology

Our study broadly follows a conceptual analysis approach [Smith and Humphreys, 2006], in which material is examined for the presence, and frequency of concepts. These concepts can be words or phrases and may be implied or explicit. To ensure broad coverage of data quality research, we select well regarded Information Systems and Computer Science academic publication outlets. The selection is based on journal and conference rankings (See www.aisnet.org and www.core.edu.au) that are now common in many disciplines [Fisher et al., 2008] as well as our perception of these outlets. We acknowledge that this is an area of much debate and may vary between researchers. However, we have attempted to minimize any bearing on the outcome through the selection by an expanded scope and as far as possible identifying a well-balanced set of publications for the analysis. We further broaden our perspective through the consideration of both conference and journal publications, to provide a different perspective to the relatively common journal-only literature and citation studies [Chen et al., 2007].

Table 1 details the list of considered Information Systems and Computer Science publication outlets, and the respective volume of papers, that has been considered in this study. In particular, we have focused on almost the last two decades of conference and journal publications (1990-2009) 2. Collections vary depending on the span of the conference/journal, and was sometimes prohibited by the unavailability of high quality digital copies.

Our data set consists of 31,701 articles. Given the large volume of papers considered, we set out to develop a consistent and reproducible full text search strategy prior to commencing analysis. As a first step and where required, each article was inspected and prepared for a full text search (OCR). The articles were organized in a structured folder system based on the discipline area, then on the publication type (conference or journal), then the relevant outlet, followed by folders for each year of publication within that structure. The entire folder system was then subjected to the creation of a full-text index. The created index was then used to identify contributions relevant to data quality research. This step was performed through a series of searches using the high level keywords of “data quality”, “quality of data”, “information quality”, and “quality of information” as the first seed for the analysis. An article was selected as potentially relevant to our study if the keyword(s) occurred 3 times or more within the body of the text. This process resulted in four (one per each keyword) large, overlapping, sets of articles, which were then cleaned by removing duplicates to arrive at a total of 961 unique documents. Once this set of unique articles was obtained, we started the process of elimination of irrelevant papers. The removal of such papers was necessary for the following reasons:

1. The theme of the article was irrelevant to data quality research; or
2. The keywords only appeared in the bibliographic references of the article, authors bio, or footnote; or

2This span varies for publications that were incepted after 1990 and in such cases included all of the publications since inception.
3. The keyword was a part of another phrase irrelevant to data quality e.g. “quality of data modelling”, “quality of data transmission”, “quality of data flow diagram”, to name a few.

The application of the above criteria reduced the relevant data set to 764. This narrowing indicates that 2.4% of the above listed publication outlets between the years 1990-2009 were directly and explicitly discussing concepts relevant to data or information quality. However, it was evident that the data set may also contain articles in which the chosen broad keywords may not necessarily explicitly appear, but the articles could still be implicitly related to the area and contain valuable outcomes. For example, papers within the database/computer science community that focus on record linkage may not contain any of the aforementioned four keywords but are still relevant to data quality research.

Accordingly, as a next step following the identification of the reduced dataset of 764 relevant papers, we identified a set of ‘second level’ keywords to further review the literature. To obtain an objective and relevant list, two researchers independently reviewed a sample (5%) of the initial set of articles to obtain further relevant concepts/keywords. The researchers identified the high level main theme(s) of the papers and associated these with terms and/or phrases that are representative of the theme e.g. terms such as entity resolution, record linkage, data profiling, provenance and lineage etc. Through this resource intensive activity, a large number of second level keywords was identified. The results of the two independent researchers were then compared, followed by a discussion to resolve any keyword conflicts. The agreed set of keywords was then later reduced as several did not return search results that were meaningful for data quality research.

A review of the second level keywords identified that several had synonyms. For example, record linkage had several related techniques such as approximate join, similarity join, fuzzy matching etc. Thus our identification of the second level keywords resulted in the development of a keyword taxonomy (see Figure 1). Finally, the identified keywords were also compared with a number of existing studies that have contributed to developing concept maps [Lima et al., 2006] and various taxonomies for data quality, see e.g. [Lima et al., 2006], [Ge and Helfert, 1996], [Madnick et al., 2009]. A number of augmentations were made to the list, including some further categories of the so-called second (and sometimes further) level keywords in order to ensure wider and more complete coverage.

It is important to point out that the purpose of this exercise was not to produce a classification of data quality research, but to be able to identify a sufficiently wide number of topics, so that the subsequent searches would return relevant results and produce a comprehensive coverage of the data quality research landscape. Accordingly, these new keywords were then used to search the data set again. The same strategy was used to prune the returned results as for the general keywords. After this second phase of analysis, a total of 1364 relevant publications were identified. Where there was a large group of publications (>50 papers) within a given keyword, an attempt was made to find sub keywords if possible e.g. edit distance, q-gram etc. for approximate matching.

Finally the bibliographic data of the selected papers (as found from the general keywords as well as the second level keywords) were recorded in a database together with the keyword(s) through which the paper was identified. Most publication were assigned more than one keyword. In general a publication with a lower level keyword (see Figure 1) was also associated with the higher level keywords. For example, a paper assigned the keyword edit distance will also have the keywords approximate matching and linkage. Further, some keywords were jointly assigned, e.g. data quality metrics and data quality assessment co-exist due to the clear overlap in the topics. These steps were taken in order to ensure an effective search of the papers later through the database, especially as the intention was to make this database publicly available. Access details of the online database are presented at the end of this paper.

4 Summary of Results

In this section we present key results of the study. We first present the taxonomy developed as a result of keyword identification during the data analysis (as discussed in previous section). The taxonomy consists of a total of 39 keywords across three levels of the hierarchy. The taxonomy is presented in Figure 1, and demonstrates the variability of themes in data quality research ranging from highly technical problems such as similarity joins for data linkage to very general such as data reliability.

In the sections below we present an analysis of the data based on above taxonomy from two aspects, namely analysis of document (publication) frequencies with respect to keyword and outlet, and a thematic analysis.

4.1 Keywords and Outlet Analysis

The Keyword and outlet analysis has been conducted using the bibliographic database that was constructed as part of the study. Table 2 and Figure 2 present a summary of the most widely published data quality topics from a variety of aspects.

The top 3 topics of data quality assessment, dimensions and metrics depict a common set of papers as the three keywords were jointly assigned due to the high topical overlap and also to promote effective search in the database. Similarly Constraints and Data Consistency also represent a common set. Although the number of papers is too large to discuss individually, we studied further a selection of the papers in the first three sets, i.e. the combined Assessment,
Dimensions and Metrics set (617), set of papers for Linkage (301) and for Content Quality (255). In order to make a meaningful selection from each set we utilized their citation count$^3$ and picked the top 8-10 papers that were most highly cited within the set.

The first set, as expected is dominated by papers that focus on organizational aspects, particularly on measurement of Information System Success e.g [Delone and McLean, 2003], [Rai et al., 2002], [Wixom and Watson, 2001]. Classical papers such as [Wang et al., 1995], [Lee et al., 2002b], [Ballou et al., 1998] also appear here. However, some works from computational and architectural perspective also appear, predominantly on Data Integration related issues such as [Lenzerini, 2002], [Spaccapietra et al., 1992]. The second set on Linkage primarily consists of computationally focused papers and span both Schema Matching [Do and Rahm, 2002] as well as data matching [Hernández and Stolfo, 1995] techniques. It also includes papers on data streams [Babcock et al., 2002], [Dobra et al., 2002], semantics/ontologies [Rodríguez and Egenhofer, 2003], dominance and preference modeling [Kossmann et al., 2002], [Papadias et al., 2003], rule based [Pirahesh et al., 1992] and probabilistic [Dalvi and Suciu, 2007] approaches.

The third set on Content Quality contains papers from all three aspects, i.e organizational (mostly from IS outlets), architectural and computational. In addition publications specific to particular data types can be found such as XML [Lee et al., 2002a], RDF [Jeffery et al., 2006], sensor data [Sharaf et al., 2004], spatial data [Koudas and Sevcik, 1997], web data [Katrattanakul and Siau, 1999], [Aladwani and Palvia, 2002], etc.

From the above, there is a clear indication that DQ themes are spread between IS and CS outlets. The overall distribution of papers between IS and CS outlets is summarized in figure 2. In the section on thematic analysis, we will further elaborate on the synergies (or lack of) between the two streams of contributions.

In Table 3, we present top document frequencies with respect to publication outlets. Table 3 provides an alternative view for observing main themes of research activity (as presented in Table 2) in the prominent IS and CS outlets. Obviously the International Conference on Information Quality (ICIQ) has the highest number of publications that span across a large number of keywords, with data quality assessment, metrics, dimensions and management being the dominant ones. For AMCIS, in addition to the above keywords, information usefulness and content quality can also be observed. Similarly, for VLDB as well as DKE journal, linkage is the dominant keyword, closely followed by data consistency and data uncertainty.

Finally, in Figure 3, we present a summary of the trends in the four publication outlets considered, namely IS conferences and journals, and CS conferences and journals. There is a clear upward trend post late 1990s in number of publications and signs of continued growth in publications from CS conferences (which primarily consist of computational approaches). Note that the last 2 years of publications

---

$^3$The citation analysis has been conducted through google scholar, using a web parser which opens a google scholar search for a title (publication) and extracts the number of citations for that title. All entries in the bibliographic database were searched in google scholar and their citations extracted and included into the database.
have not been fully uploaded in the database and hence the decreasing trend is not entirely accurate. Other than that, the numbers do not indicate any specific trends.

4.2 Thematic Analysis

Finally, we conducted a thematic analysis of the papers through a text-mining tool called Leximancer4. Leximancer performs a full text analysis both systematically and graphically by creating a map of the concepts and themes re-appearing in the text. The tool uses a machine-learning technique based on the Bayesian approach to prediction. The procedure used is a self-ordering optimization technique (unlike neural networks). Once the optimal weighted set of evidence words is found for each concept, it is used to predict the concepts present in fragments of related text. In other words, each concept has other concepts that it attracts (or is highly associated with contextually) as well as concepts that it repels (or is highly disassociated with contextually). The relationships are measured by the weighted sum of the number of times two concepts are found in the same “chunk” of text (typically 3 sentences long). An algorithm is used to weight them and determine the confidence and relevance of the terms to others in a specific chunk and across chunks. A concept map displays the themes and concepts in a manner that links to related subtext and may subsequently be explored. Each of the identified concepts is placed on the map in proximity of other concepts in the map through a derived combination of the direct and indirect relationships between those concepts (e.g. see Figure 4). In brief, a concept map has the following main characteristics:

• Concepts are represented by labeled and color coded dots. The labels are the abbreviated single word descriptors for a collection of evidence words that make up the concept (e.g. refer to the “group” concept in Figure 4).

• The size and the brightness of a concept dot on the map is indicative of the concept’s strength within the body of analyzed text (i.e. the brighter, bigger the concept, the more often it appears in the text; e.g. refer to the “query” concept in Figure 4, relative to the “group” concept);

• The thickness and the brightness of connections between concepts is indicative of the frequency of co-occurrence of the two concepts. Two concepts that occur together frequently (through their underlying evidence words) will be connected by a thicker and brighter link than two concepts that co-occur less frequently.

• The relative distance of concepts on the map is indicative of similar conceptual contexts (i.e. the concepts co-occur more frequently with concepts closer on the map, and less frequently with concepts that are placed further away) – e.g. “relational” and “query” are closely related relative to a relationship between “query” and “probabilistic”)

• Thematic clusters of concepts are indicated through colored circles, called themes. Themes are formed only around concepts that are highly connected. Furthermore, the color used to represent the theme is indicative of the degree of connectedness of the concepts within the theme, with colors towards the red color spectrum end representing more connected thematic clusters. In other words, concepts in a red-colored theme are more connected than concepts in a blue-colored theme), thus signifying a strong theme, e.g. it is clear in Figure 5 that the “information” theme is the strongest theme in the data analysed, with no strong focus (nor existing major theme respectively) on raw data.

To explore the synergies and differences between data quality research in the Computer Science and Information Systems disciplines we conducted a series of Leximancer analyses separately for each of the top 10 keywords listed in Table 2. For each of the keywords, data was analysed considering the Computer Science publications in isolation, then considering the Information Systems publications in isolation, followed by a joint analysis of both data sets to gain a better understanding of the relative distance and common focus of the two disciplines within the same topical area5. In each case, after one Leximancer pass, the list of automatically generated concepts was edited to remove concepts that do not add to the understanding of the content of the paper (given the already narrowed data set of papers) and would only clutter and dominate the generated theme map. For example, concepts such as “paper”, “study”, “follows”

---

4https://www.leximancer.com/

5Due to lack of space, we omit these results here and present a few exemplary analyses and related discussion.
**Table 3: Top Document Frequencies with respect to Publication Outlet**

<table>
<thead>
<tr>
<th>Publication outlet</th>
<th>#Pubs</th>
</tr>
</thead>
<tbody>
<tr>
<td>International Conference on Information Quality (ICIQ)</td>
<td>241</td>
</tr>
<tr>
<td>Americas Conference on Information Systems (AMICIS)</td>
<td>152</td>
</tr>
<tr>
<td>International Conference on Very Large Databases (VLDB)</td>
<td>148</td>
</tr>
<tr>
<td>IEEE Transactions on Knowledge and Data Engineering (DKE)</td>
<td>120</td>
</tr>
<tr>
<td>ACM SIGMOD International Conference on Management of Data (SIGMOD)</td>
<td>116</td>
</tr>
<tr>
<td>ACM Transactions on Information Systems (TOIS)</td>
<td>51</td>
</tr>
<tr>
<td>Communication of the ACM (CACM)</td>
<td>49</td>
</tr>
<tr>
<td>Pacific Asia Conference on Information Systems (PACIS)</td>
<td>45</td>
</tr>
<tr>
<td>Hawaii International Conference on System Sciences (HICSS)</td>
<td>44</td>
</tr>
<tr>
<td>Symposium on Principles of Database Systems (PODS)</td>
<td>36</td>
</tr>
<tr>
<td>ACM Transactions on Database Systems (TODS)</td>
<td>35</td>
</tr>
<tr>
<td>International conference on Information systems (ICIS)</td>
<td>34</td>
</tr>
<tr>
<td>European Conference on Information Systems (ECIS)</td>
<td>33</td>
</tr>
<tr>
<td>Australasian conference on Information Systems (ACIS)</td>
<td>33</td>
</tr>
<tr>
<td>Journal of Information &amp; Management (IM)</td>
<td>27</td>
</tr>
<tr>
<td>ACM Special Interest Group on Information Retrieval (SIGIR)</td>
<td>27</td>
</tr>
<tr>
<td>International Conference on Extending Database Technology (EDBT)</td>
<td>22</td>
</tr>
<tr>
<td>International Conference on Database Systems for Advanced Applications (DASFAA)</td>
<td>20</td>
</tr>
<tr>
<td>Journal of Management Information Systems (MIS)</td>
<td>19</td>
</tr>
<tr>
<td>International Workshop on Information Quality in Information Systems (IQIS)</td>
<td>18</td>
</tr>
<tr>
<td>Journal of Information Systems Research (ISR)</td>
<td>12</td>
</tr>
<tr>
<td>Management Information Systems Quarterly (MISQ)</td>
<td>12</td>
</tr>
<tr>
<td>International Conference on Advanced Information Systems Engineering (CAISE)</td>
<td>10</td>
</tr>
</tbody>
</table>

were removed in addition to the automatic removal of stop words inbuilt into Leximancer (e.g. “a”, “the”, “therefore”, etc) before running the analysis a second time, to make sure that the analysis is not biased by concepts and their underlying evidence words that have little relevance to the topic at hand.

In Figures 4 and 5 we present a comparison of the concept maps of the most frequently discussed topic area – that of Data Quality Metrics – in the Computer Science (CS) publications outlets and the Information Systems (IS) publication outlets respectively. The analysis clearly, but perhaps not surprisingly, shows that CS publications have a strong focus on data and the quality of the relational model, together with queries, approaches for quality measurement, as well as data constraints (see Figure 4). In the Information Systems field, however, information has the stronger focus of the research community, although there is a strong relationship between the “information” and “data” themes. The set of publications, however, highlights that the IS researchers focus on the outcome of contextualised data, i.e. information, and the quality of information provided to the organisation/users. Indeed, an interesting observation from the two concept maps in Figures 4 and 5 is that the IS community is focused on the use of information for decision making and the business value of high levels of quality of information extracted from Information Systems. While “information” appears as a concept in the CS publication set, it has a much stronger presence (i.e. one that ensures that it is in fact the central theme of the publication set) in the IS publication set.

We conclude from the analysis of Figure 4 and Figure 5 that there is a lack of synergy between the data quality metrics related research in the two disciplines. While the CS publications have a strong focus on technical solutions, there appears to be a lack of strong motivation (a business need) that triggers this research. On the other hand, the IS publications relevant to data quality metrics show that the strongest foci lie with information quality in general, the various application areas (e.g. “health”) but don’t have a strong focus on solutions or approaches to measuring data quality (or information quality). We note also that the IS publications have a tendency to be more focused on the user perspective and user/client satisfaction than the CS publications.

Despite the clear results from the thematic analyses of the two disciplines, as exemplified through the analysis of the Data Quality Metrics topic, we note a limitation in the isolated analysis of the two sets of publications. In particular, as Leximancer is designed to identify common themes and build a context of a concept, two concepts with the same name can have a slightly different meaning as they are created out of a selected set of text. This limitation is common to our approach where the publication sets from different disciplines are analyzed in Leximancer in isolation and the relationships between concepts in these disparate data sets is thus not explored and not taken into consideration during the concept learning phase of the Leximancer algorithm. It is possible, however, to supplement such analyses with a second
Leximancer analysis performed on the two data sets jointly. Leximancer, in particular, allows for such an analysis by means of file concepts, which can be used to explore the relationships of concepts within each of the two datasets within a single combined analysis. To exemplify the outcome of such an analysis within our study, we select the set of “Content Quality” related publications in the CS and IS publication outlets. We deliberately select a different topical area, one that is more uniformly discussed across the publication sets (refer to Figure 2).

Figure 5 shows the outcome of the analysis, wherein the collective Information Systems dataset related to the “Content Quality” topic is indicated by a “FOLDER-136-IS” concept. Likewise, the Computer Science dataset is represented by “FOLDER-136-CS” concept. Specifically, it shows the relationships of concepts related to Content Quality across all considered publication years and how the data set relates to concepts that were identified to be the strongest common concepts across the two data sets. Our analysis indicates that, while there are concepts that are common to both data sets, the strength of the connection is weak. Indeed, the analysis uncovers strong evidence that the Information Systems set of papers is very strongly focused on information quality, issues relating to satisfaction (as was also the case in the paper set relevant to Data Quality Metrics) and business value in general, yet it is not as strongly focused (as indicated by the relative distance of the themes from each other and the relative closeness of the themes to each of the two publication sets) on approaches for ensuring content quality. While this is not surprising in itself, given that Information Systems is a less technically oriented community of researchers, we see a weakness in a situation where the communities that should be collaborating together, appear to lack a strong collaboration and common focus.

The analysis also indicates that themes and concepts that are more commonly referred to in both sets of publications are those that relate to attribute values, the context of data/information use, and performance related issues.

Overall, based on the above analysis excerpt, we see a need for a stronger connection of technical solutions published by the Computer Science publication outlets and the application (and evaluation) of those technical solutions in the Information Systems discipline. After all, while the two publication sets appear to have a focal difference on data vs information quality, ‘good quality data’ is the necessary foundation for ‘good quality information’. Organisations rely strongly on information to support their operational and strategic decision making, thus any quality problems related to information that resides in organizational systems will have a negative impact on the overall operations of the organization in the long term. Therefore, we see the practice of developing data quality solutions in isolation from the real-world application of those solutions, and the practice of developing solutions without a strong business motivation for them, as counter-productive to reducing data quality problems in organizations. Accordingly, we call for the two communities to increase their awareness of the relevant leading research that is often published in publication outlets that are not on the immediate radar of the research community. This paper is the initial step to increase the connection and highlight synergies and differences between the foci of the two communities in data quality related research.

5 Concluding Summary

We have presented summary findings relating to the main trends, themes and synergies across data quality research communities. The study is by no means complete, and we hope as a first step in our future work to extend the study to include citation and social network analysis so as to better understand the relationships between the IS and CS research contributions. The deeper analysis is expected to be complemented by an industry study on data quality tools, issues and challenges that will provide a comparison platform against the research (solution) contributions and help identify true gaps and an industry relevant research agenda for this area.

The dataset used to present the analysis in this paper has been recorded in a bibliographic database. The database is publicly available (excluding full text of publications) from the Data Quality Management Community Portal (dqm.cloud.itee.uq.edu.au). Access to the databases is restricted to members of the
community portal, but membership is free. The bibliographic database has advanced search functionality i.e it can be searched on various fields within the bibliographic entry including the keywords as presented in the taxonomy in Figure 1. Additionally, it is our intention to utilize the portal to establish an Australian wide community of practice on data quality, which will be open to user communities, solution providers and vendors as well as research and academia. Readers are welcome to join the community through its free membership registration at dqm.cloud.itee.uq.edu.au.

References


Appendix 1 - full names for the publication outlets

<table>
<thead>
<tr>
<th>Publication Outlet</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Computer Science Conferences</strong></td>
</tr>
<tr>
<td>International Conference on Business Process Management</td>
</tr>
<tr>
<td>Conference on Information and Knowledge Management</td>
</tr>
<tr>
<td>International Conference on Database Systems for Advanced Applications</td>
</tr>
<tr>
<td>European Conference on Object-Oriented Programming</td>
</tr>
<tr>
<td>International Conference on Extending Database Technology</td>
</tr>
<tr>
<td>International Conference on Data Engineering</td>
</tr>
<tr>
<td>Symposium on Principles of Database Systems</td>
</tr>
<tr>
<td>ACM Special Interest Group on Information Retrieval</td>
</tr>
<tr>
<td>ACM SIGMOD International Conference on Management of Data</td>
</tr>
<tr>
<td>International Conference on Very Large Databases</td>
</tr>
<tr>
<td>Web Information and Data Management</td>
</tr>
<tr>
<td>Web Information Systems Engineering</td>
</tr>
<tr>
<td><strong>Computer Science Journals</strong></td>
</tr>
<tr>
<td>ACM Transactions on Database Systems</td>
</tr>
<tr>
<td>ACM Transactions on Information Systems</td>
</tr>
<tr>
<td>ACM Communication Transaction ' Communications of the ACM</td>
</tr>
<tr>
<td>IEEE Transactions on Knowledge and Data Engineering</td>
</tr>
<tr>
<td>Decision Support Systems</td>
</tr>
<tr>
<td>Information Systems Journal (Elsevier)</td>
</tr>
<tr>
<td>Journal of Database Management</td>
</tr>
<tr>
<td>IEEE Transactions on Knowledge and Data Engineering</td>
</tr>
<tr>
<td>Journal of Very Large Databases</td>
</tr>
<tr>
<td>ACM Journal of Data and Information Quality</td>
</tr>
<tr>
<td><strong>Information Systems Conferences</strong></td>
</tr>
<tr>
<td>Australasian conference on Information Systems</td>
</tr>
<tr>
<td>Americas Conference on Information Systems</td>
</tr>
<tr>
<td>Conference on Advanced Information Systems Engineering</td>
</tr>
<tr>
<td>European Conference on Information Systems</td>
</tr>
<tr>
<td>International Conference on Conceptual Modeling</td>
</tr>
<tr>
<td>Hawaii International Conference on System Sciences</td>
</tr>
<tr>
<td>International Conference on Information Quality</td>
</tr>
<tr>
<td>International conference on Information systems</td>
</tr>
<tr>
<td>International Federation for Information Processing Working Groups</td>
</tr>
<tr>
<td>Information Resources Management Association International Conference</td>
</tr>
<tr>
<td>Information Systems Foundation</td>
</tr>
<tr>
<td>Pacific Asia Conference on Information Systems</td>
</tr>
<tr>
<td><strong>Information Systems Journals</strong></td>
</tr>
<tr>
<td>Business Process Management Journal</td>
</tr>
<tr>
<td>Communications of the Association for Information Systems</td>
</tr>
<tr>
<td>European Journal of Information Systems</td>
</tr>
<tr>
<td>Journal of Information &amp; Management</td>
</tr>
<tr>
<td>Information System Frontiers</td>
</tr>
<tr>
<td>Information Systems Journal (Blackwell)</td>
</tr>
<tr>
<td>Information Systems Journal (Sarasota)</td>
</tr>
<tr>
<td>Journal of Association for Information Systems</td>
</tr>
<tr>
<td>Journal of Information Systems Research</td>
</tr>
<tr>
<td>Management Information Systems Quarterly</td>
</tr>
<tr>
<td>Management Information Systems Quarterly Executive</td>
</tr>
</tbody>
</table>
Author Index

Částková, Zuzana, 17
Antoine, Elizabeth, 123
Atkinson, Ian, 65, 75
Cao, Jinli, 143
Chao, Wen Han, 57
Christensen, Chris, 65
Endres, Markus, 7
Franke, Hubertus, 27
Ghodosi, Hossein, 75
Hu, Biyun, 57
Hu, Xia, 57
Indulska, Marta, 153
Iyer, Bala, 27
Kießling, Werner, 7
Kitsuregawa, Masaru, 37
Li, Yuxuan, 85
Li, Zhixu, 103
Li, Zhoujun, 57
Marks, Gerard, 47
Min, Hong, 27
Nakano, Miyuki, 37
Nishikawa, Norifumi, 37
Pokorny, Jaroslav, 17
Qin, Yongrui, 95
Rahayu, Wenny, 143
Ramamohanarao, Kotagiri, 123
Read, Wayne, 65, 75
Roantree, Mark, 47
Sadiq, Shazia, 153
Shao, Jie, 123
Sharaf, Mohamed, 133
Shen, Heng Tao, iii
Sim, Nigel, 65
Sitbon, Laurianne, 103
Smyth, Dominick, 47
Stantic, Bela, 113
Trevathan, Jarrod, 65, 75
Villafuerte, Frances, 27
Vo, Loan T.H, 143
Wang Jun, 57
Wang, Hua, 95
Watts, Julie, 27
Xiao Jitian, 95
Yeganeh, Naiem, 153
Zhang, Rui, 3, 123
Zhang, Xiuzhen, 85
Zhang, Yanchun, iii
Zhou Xiaofang, 103
Zhou, Xiaofang, 133
Zhu, Yingying, 133
Listed below are some of the latest volumes published in the ACS Series Conferences in Research and Practice in Information Technology. The full text of most papers (in either PDF or Postscript format) is available at the series website http://crpit.com.

Volume 91 - Computer Science 2009
Contains the proceedings of the Thirty-Second Australasian Computer Science Conference (ACSC2009), Wellington, New Zealand, January 2009.

Volume 92 - Database Technologies 2009
Contains the proceedings of the Twentieth Australasian Database Conference (ADC2009), Wellington, New Zealand, January 2009.

Volume 93 - User Interfaces 2009
Contains the proceedings of the Ninth Australasian User Interface Conference (AUIC2009), Wellington, New Zealand, January 2009.

Volume 94 - Theory of Computing 2009
Edited by Prabhakar Manjunath, University of Ballarat and Rod Downey, Victoria University of Wellington. January, 2009. 978-1-920682-75-0.

Volume 95 - Computing Education 2009
Contains the proceedings of the Eleventh Australasian Computing Education Conference (ACE2009), Wellington, New Zealand, January 2009.

Volume 96 - Conceptual Modelling 2009
Contains the proceedings of the Fifth Asia-Pacific Conference on Conceptual Modelling (APCCS2008), Wollongong, NSW, Australia, January 2008.

Volume 97 - Health Data and Knowledge Management 2009
Contains the proceedings of the Third Australasian Workshop on Health Data and Knowledge Management (HDKM 2009), Wellington, New Zealand, January 2009.

Volume 98 - Information Security 2009
Contains the proceedings of the Australian Information Security Conference (AISC 2009), Wellington, New Zealand, January 2009.

Volume 99 - Grid Computing and e-Research 2009

Volume 100 - Safety Critical Systems and Software 2009

Volume 101 - Data Mining and Analytics 2009
Contains the proceedings of the 8th Australasian Data Mining Conference (AusDM 2009), Melbourne, Victoria, Australia, November, 2009.

Volume 102 - Computer Science 2010
Contains the proceedings of the Thirty-Third Australasian Computer Science Conference (ACSC 2010), Brisbane, Queensland, Australia, January 2010.

Volume 103 - Computing Education 2010
Contains the proceedings of the Twelfth Australasian Computing Education Conference (ACE2010), Brisbane, Queensland, Australia, January 2010.

Volume 104 - Database Technologies 2010
Contains the proceedings of the Twenty-First Australasian Database Conference (ADC 2010), Brisbane, Queensland, Australia, January 2010.

Volume 105 - Information Security 2010
Contains the proceedings of the Eight Australasian Information Security Conference (AISC 2010), Brisbane, Queensland, Australia, January 2010.

Volume 106 - User Interfaces 2010
Contains the proceedings of the Eleventh Australasian User Interface Conference (AUIC2010), Brisbane, Queensland, Australia, January 2010.

Volume 107 - Parallel and Distributed Computing 2010
Contains the proceedings of the Eighth Australasian Symposium on Parallel and Distributed Computing (AusPDC 2010), Brisbane, Queensland, Australia, January 2010.

Volume 108 - Health Informatics and Knowledge Management 2010
Contains the proceedings of the Fourth Australasian Workshop on Health Informatics and Knowledge Management (HIKM 2010), Brisbane, Queensland, Australia, January 2010.
Volume 109 - Theory of Computing 2010
Contains the proceedings of the Sixteenth Computing: The Australasian Theory Symposium (CATS 2010), Brisbane, Queensland, Australia, January 2010.

Volume 110 - Conceptual Modelling 2010
Contains the proceedings of the Seventh Asia-Pacific Conference on Conceptual Modelling (APCCM2010), Brisbane, Queensland, Australia, January 2010.

Volume 112 - Advances in Ontologies 2009
Contains the proceedings of the Australasian Ontology Workshop 2009 (AOW 2009), Melbourne, Australia, December, 2009.