Database Server Workload Characterization in an E-commerce Environment

A Thesis Submitted to the College of

Graduate Studies and Research

in Partial Fulfillment of the Requirements

For the Degree of Master of Science

in the Department of Computer Science

University of Saskatchewan

Saskatoon, Saskatchewan

by

Fujian Liu

Permission To Use

In presenting this thesis in partial fulfillment of the requirements for a Post-graduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science
University of Saskatchewan
Saskatoon, Saskatchewan, Canada
S7N 5C9

Abstract

A typical E-commerce system that is deployed on the Internet has multiple layers that include Web users, Web servers, application servers, and a database server. As the system use and user request frequency increase, Web/application servers can be scaled up by replication. A load balancing proxy can be used to route user requests to individual machines that perform the same functionality.

To address the increasing workload while avoiding replicating the database server, various dynamic caching policies have been proposed to reduce the database workload in E-commerce systems. However, the nature of the changes seen by the database server as a result of dynamic caching remains unknown. A good understanding of this change is fundamental for tuning a database server to get better performance.

In this study, the TPC-W (a transactional Web E-commerce benchmark) workloads on a database server are characterized under two different dynamic caching mechanisms, which are generalized and implemented as query-result cache and table cache. The characterization focuses on response time, CPU computation, buffer pool references, disk I/O references, and workload classification.

This thesis combines a variety of analysis techniques: simulation, real time measurement and data mining. The experimental results in this thesis reveal some interesting effects that the dynamic caching has on the database server workload characteristics. The main observations include: (a) dynamic cache can considerably reduce the CPU usage of the database server and the number of database page references when it is heavily loaded; (b) dynamic cache can also reduce the database reference locality, but to a smaller degree than that reported in file servers. The data classification results in this thesis show that with dynamic cache, the database server sees TPC-W profiles more like on-line transaction processing workloads.

Acknowledgements

I would like to extend my sincere gratitude to all those who gave me the possibility to complete this thesis. First, I am deeply indebted to my supervisor, Dr. Dwight Makaroff, for his constant encouragement and valuable suggestions during all the time of research and writing of this thesis. In particular, I highly appreciate the great amount of time he spent for my program and his high availability to students. I have learned a lot from his overly dedication and enthusiasm to research. Besides of being an excellent supervisor, Dwight is as close as a great friend to me. I am really glad that I have come to get know Dwight in my life.

My thanks also go to Dr. David Klymyshyn (external), Dr. Nadeem Jamali, and Dr. Derek Eager for serving as members of my thesis defense committee. Their insightful comments and suggestions have gone a long way to improve the quality of the thesis.

Thanks to Brian Gallaway (System Administrator) for providing technical support whenever I needed it. Thanks to Jan Thompson (Graduate Correspondent) for her effective administrative helps. Thanks to my fellow graduate students for making my time in the program memorable.

This thesis could not have been accomplished without Huan Liang, my wife who is always with me no matter how dubious my decision were. She always gives me incredible support and love in every situation. I believe I owe deepest thanks to all people in my entire family who have supported me since I came to Canada.

Table Of Contents

List of Tables

List of Figures

List of Acronyms

2Q – Two Queue (buffer pool replacement algorithm)

ACID – Atomicity, Consistency, Isolation, and Durability

ARC – Adaptive Replacement Cache

CLOCK – Clock (buffer pool replacement algorithm)

CPU – Central Processing Unit

DBMS – DataBase Management System

DCA – Dynamic Content Accelerator

DSS – Decision Support System

EB – Emulated Browser

E-commerce – Electronic commerce

FBR – Frequency Based Replacement

FIFO – First-In First-Out

HTML – HyperText Mark-up Language

HTTP – HyperText Transfer Protocol

IRR – Inter-Reference Recency (buffer pool replacement algorithm)

JDBC – Java DataBase Connectivity

KB – Kilo Bytes (2^{10} bytes)

LFU – Least Frequently Used

LIRS – Low Inter-reference Recency Set

LRU – Least Recently Used

MP – Multiple Processors

MRU – Most Recently Used

OLAP – On-Line Analytical Processing

OLTP – On-Line Transaction Processing

PGE – Payment Gateway Emulator

QoS – Quality of Service

RAID – Redundant Arrays of Independent Disks

RAM – Random Access Memory

RPM – Revolutions Per Minute

SCSI – Small Computer Systems Interface

SQL – Structured Query Language

SSL – Secure Sockets Layer

TPC – Transaction Processing Performance Council

TPC-C – TPC benchmark C

TPC-D – TPC benchmark D

TPC-H – TPC benchmark H

TPC-W – TPC benchmark W

UDB – Universal DataBase

UID – Update/Insert/Delete

URL – Uniform Resource Locator

WIPS – Web Interactions Per Second

Chapter 1

Introduction

1.1 E-commerce System and Their Challenges

The use of computer networks to conduct commercial activities has shown a steady increase and is an important component of the retail industry. The term E-commerce is used to describe many forms of such activity, from business to consumer, business to business and consumer to consumer. Different researchers and business people use the term in different contexts. In one report published by US Census Bureau[1], the E-commerce sales reached 21 billion dollars in the third quarter of 2005, which accounted for 2.2% of total retail sales; compared with the same quarter in the previous year, the E-commerce sales increased by 26.5% at an annual rate on average for every quarter from the first quarter of 2001 to the third quarter of 2005.

Also, Statistics Canada[2] reports that the E-commerce sales in Canada increased by a high annual rate of 51.4% on average from the year of 2000 to the year of 2004. More interestingly, it shows that in Canada, private sector firms accounted for an overwhelming majority of the E-commerce sales, and that business to business sales played an important role in E-commerce activities. For example in 2004, private companies contributed 93% of the total E-commerce sales, and business to business sales represented 75% of total E-commerce sales by private firms.

E-commerce provides Web users the convenience of shopping online. If Web users experience long response time however, the results to an E-commerce business

[1] http://www.census.gov/mrts/www/data/html/05Q3table4.html
[2] http://www.statcan.ca/Daily/English/050420/d050420b.htm

provider can be detrimental. Previous study [25] has shown that users abandon Web sites at the rate of 2% when the response times are less than 7 seconds, while this rate goes dramatically up to 30% with response times around 8 seconds, and jumps to 70% if response times are more than 12 seconds. Slow processing of the online transactions caused about 420 million USD in lost revenues in 1999 [42]. A response time is composed of client processing time, network delay and server processing time, and it has been found that server-side latency accounts for 40% of the Web page delay experienced by the Web users [21].

The database server in E-commerce systems can become the performance bottleneck when the request frequency increases due to the increase of the number of Web users. Zhang et al. [53] have shown that the database server response time increases dramatically when the number of concurrent users increases. All the database response times are less than 5 seconds when there are 128 concurrent users. However, 5% of the database response times are more than 5 seconds when there are 256 concurrent users, and the percentage goes up to 20% when there are 512 concurrent users.

To reduce the delay at the server side, a database server in an E-commerce system can be upgraded to a machine with a faster CPU, more memory, and faster disks. This is an expensive solution, however, and such a server will become the bottleneck again due to the continuous increase of database size and the request frequency.

Fortunately, there are two alternative solutions addressing the challenge [26]. The first one is to extend the traditional database to a parallel database or distributed database to achieve the scalability. The second solution is to use caching to store frequently accessed items in higher speed storage locations either near clients or near the server.

Database workloads in E-commerce tend to be split into 2 separate activities/categories. Some of the data is mainly read-only data, for example the address of a customer. Other data is updated very often, such as prices and quantities in stock. As well, the order table is updated on every order transaction. Parallel/distributed

databases do not explicitly consider this processing characteristic and instead try to optimize performance for a general database workload [37]. Therefore, the approach in this thesis will focus on caching techniques which permit the separation of data items according to their usage characteristics. In such techniques, managing read-only data at a cache can reduce the load on a database server, while updates are still served by the database server.

Caching can affect the characteristics of the E-commerce workloads seen by the database server. Understanding the effects that caching has on E-commerce workloads is crucial for the design and tuning of database servers in E-commerce systems. This thesis studies the effects that caching has on the database workloads in an E-commerce environment.

1.2 Definitions

Some words/terms used in this thesis have their word context in general use, but they are restricted for the purpose of this thesis. These terms include "E-commerce", "response time", and "transaction".

E-commerce: E-commerce can be generally defined as electronically conducting any form of business, such as buying, selling, marketing, and negotiating over computer networks [5]. In this thesis, E-commerce is more narrowly referred to as the business model of purchasing and selling products and/or services electronically over the Internet. Particularly, the focus is on a *Web commerce* model. In a Web commerce system, a consumer treats a company's Web site as a virtual store where the consumer can browse around or buy products/services and pay with a credit card [36].

Response time: In this thesis, response time is used in two different contexts. The response time of a Web user request is a significant metric which has been used to measure the performance of an E-commerce system [41]. It refers to the time elapsed from when a user requests a Web interaction until when the user receives the resultant Web page. The response time of a database server refers to the time

elapsed from an application requesting a database transaction until the transaction is completed and the feedback is sent to the application.

Transaction: This term is generally used in this thesis to refer to a complete data exchange process between a Web user and the related servers of an enterprise computer system[3]. The enterprise servers usually include Web servers, application servers, and a database server. From a database point of view, a transaction is also defined as an atomic database management system (DBMS) interaction [41]. It is guaranteed either to complete successfully or not at all. If a transaction completes successfully, database changes are said to be committed; if not, changes are rolled back. The meaning of "transaction" in this thesis depends on the context. It refers to the first process if associated with an entire enterprise server system, while it refers to the latter activity if used with respect to a database server only.

1.3 A Typical Three-tier E-commerce System

There are at least three types of E-commerce applications: business to business, business to consumer, and consumer to consumer [9]. The focus of this study is the business to consumer model. A representative business to consumer enterprise E-commerce system includes three tiers (as shown in Figure 1.1): Web servers, application servers, and a database server that host the functions of presentation logic, business logic, and data management respectively [31].

In the three-tier E-commerce system architecture shown in Figure 1.1, each Web/application server can run on the same machine or separate ones. Running on separate machines introduces network latency between the Web server and the application server. Using one machine removes the network latency, but adds some requirements on the hardware, such as memory, due to the resource competition between the Web server and the application server programs. In a typical system, instead of sharing one machine with Web/application servers, a dedicated machine with a fast CPU running a traditional centralized database server is generally de-

[3]http://www.csgnetwork.com/glossary.html

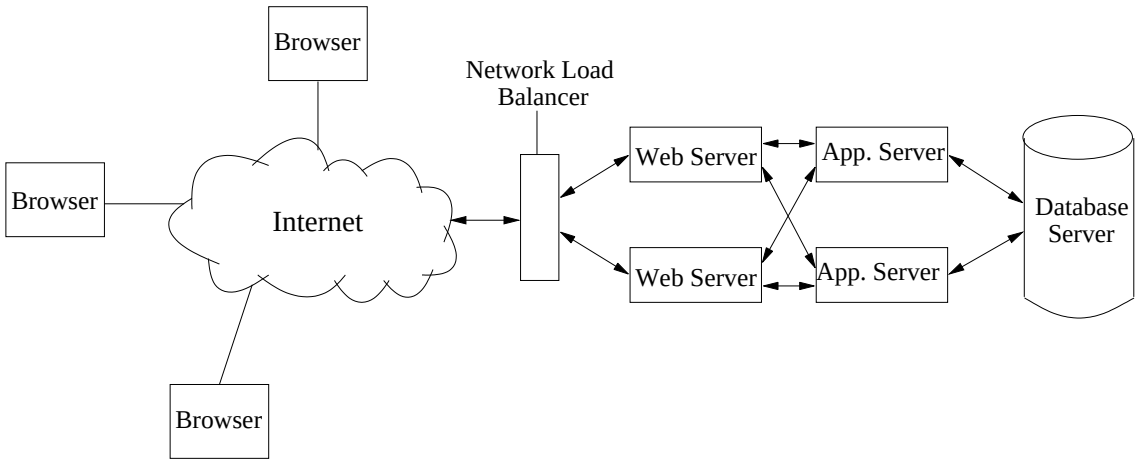


Figure 1.1: Three-tiered E-commerce Architecture

ployed [18, 25] to get better system performance, such as good response time.

Users in this type of a system (as in Figure 1.1) interact with the system through a Web interface and the system executes *business logic* at application servers. The business logic typically responds to requests from the Web servers, and may access the database server to complete the requests, and build response pages in order to do so. Application servers also keep track of user sessions to manage the shopping cart and provide page personalization. The database server stores business information, for example the price of a product. The database server also processes queries from the application server, for example selecting the top ten best sellers from the product table.

When a user accesses an E-commerce Web site, an outgoing request from the user is received by a Web server. An example of such a request is "*get product details*". Next, the Web server takes out the associated parameters from the request and routes the parameters to an application server. The application server identifies the requested data and sends queries to the database server. The database server processes the queries and sends the results back to the application server. Then, the application server uses the results to assemble a page and sends it to the Web server for final transmission to the user. Finally, the response Web page is displayed in the user's Web browser.

In a typical Web E-commerce system, the data objects stored at Web/application servers, such as images and scripts, are static. To achieve a high degree of scalability, Web servers and application servers can be scaled up by adding more machines using current network load balancing technology [26]. It is cost-effective to use many low-end commodity machines as Web/application servers. The system administrators can use multiple similarly configured machines to execute the functionality and implement a proxy machine to route the initial requests to individual Web/application servers. On the other hand, the database server stores some data, such as product quantities, that is updated frequently by transactions. It is very difficult and costly to replicate the database server while maintaining data consistency efficiently in E-commerce systems [25, 40].

Suppose several replicated databases are grouped together to serve queries via a standalone load-balancing machine. Assume a dedicated master database broadcasts any update, such as a new on sale price, to all other databases when it comes to the database group. An update is completed only after a transaction is committed across all the databases to ensure data consistency [40]. The relevant data items have to be locked until the final commit is performed on all the databases, before the fresh data is available to the incoming queries that depend on the updated data. This synchronization would be very costly, since the transaction data at an E-commerce site is usually updated with a high frequency, sometimes in a burst mode.

The single database server can use up its computing resources however to process queries from application servers with a high user request frequency. It then becomes the bottleneck of E-commerce systems, and the time spent at the database server will dominate the response time perceived by the Web users [26, 29, 53].

Caching is used in E-commerce systems for the following reasons. First, a parallel DBMS or distributed DBMS is still expensive, while cache servers can use cheap machines. The database server is often the most expensive component in the system. For example in the xSeries 440 system, the price of the database server is about the same as that of the sixty-two front-end servers (27 Web servers, 21 image servers, and 14 Web caches). Second, by holding the cached content on a separate server and

moving it closer to users, caching can achieve additional gains on the challenges of availability and performance via reduced network latency and other related factors as well as reducing the cost of the database server. These factors are not explicitly considered in this thesis, but are part of future work.

1.4 General Caching

Caching is a fundamental technique used to improve the performance of a computer system. It is widely used in operating systems, database systems, and Web servers/proxies. The objective of caches is to reduce the relative number of accesses to storage devices which have longer access times. Caching mechanisms store some portion of the data from the device with longer access time in the device that has shorter access time [35]. The resources saved could be memory access (for CPU cache), disk I/O (for file client cache and server cache), and network bandwidth (for Web browser cache and proxy cache).

Caching is often used at different levels of a system. Furthermore, it is common to have multi-level caches. A storage system has I/O controller cache and disk cache. A file system has client cache, server cache, in addition to the underlying storage caches. A Web site has browser cache, one or more proxy caches, Web server cache, in addition to the cache hierarchies of the underlying file and storage system. A database system has buffer pool cache in addition to the cache hierarchies of the underlying storage system.

1.5 Dynamic Caching in E-commerce Systems

To reduce the load on the database server and reduce the user response time, a variety of *dynamic caching* approaches have been proposed for E-commerce systems. The dynamic cache can be placed in the infrastructure of an enterprise E-commerce system or over the Internet between users and an enterprise E-commerce system.

The idea of dynamic caching is to cache some query results and/or tables, so

some queries can be filtered out before they reach the database server. A query result cache stores query results and serves subsequent identical queries using these results. A table-cache server runs on a separate machine and replicates infrequently-updated tables. The table cache server processes queries that involve the replicated tables so as to reduce the load on the database server.

In this thesis, a single cache is used. For query result cache, a timeout value (30 seconds) is set for all the cache results. After the timeout, a cached result becomes invalid, and the corresponding query must be forwarded to the database server. Before a cached result becomes invalid, the related data at the database server might change. For example the "best seller" might change during the timeout period. This possible misrepresentation during the 30 seconds of timeout period will not cause an inconsistency at the database server since the query result cache does not store any result from "update" queries, for example "customer registration". Therefore, data maybe obsolete, but it is never inconsistent.

For table cache, the changes on the cached table are propagated from the database server to the cache periodically. Some cached tables rarely change, for example the table of the names of countries. Some cached tables are not frequently updated, for example the table of the addresses of customers. There could be a misrepresentation at the table cache before changes on the cached tables are propagated from the database server to the table cache. This possible misrepresentation at the table cache will not cause data conflicts at the database server since all the updating queries are forwarded to the database server and the periodical data updating only happens in one direction: from the database server to the table cache.

In this thesis, the single query result cache or table cache is assumed being placed in the infrastructure of an E-commerce system instead of over the Internet. Further implications of these assumptions are discussed in Chapter 4. Multiple query result caches or table caches can be duplicated over the Internet to further reduce the user response time. Multiple query result caches or table caches can also be deployed as a content distribution network to serve Web user requests. However, the duplication or distribution of caches does not introduce data contention between the cache since

all the updating queries are forwarded to the database server. This thesis focuses on the effect that caching has on the database server workload. Duplication or distribution of caches over the Internet can reduce user response time and network bandwidth utilization more than a single cache sitting in the infrastructure of an E-commerce system, but it does not make much difference at the workload seen by the database server. A detailed description of cache duplication and content distribution network is beyond the scope of this thesis.

1.6 Thesis Statement

The presence of dynamic caching mechanisms may significantly affect the workload characteristics of the database server. Understanding the workload characteristics of the database server is a prerequisite for studying its performance. As new dynamic caching techniques emerge, understanding the effect of the dynamic caching on the database server workload is of paramount importance to the design, configuration, and tuning of the database server. Unfortunately, this issue has not been well addressed by previous research.

Although different dynamic caching policies have been proposed, the details of their effect on database workloads remains unknown. The design and tuning of a database server depends on the changes of workload characteristics after deploying dynamic caching. In this thesis, *the database server workloads are characterized in an E-commerce environment with the presence of various dynamic caching mechanisms by using a benchmark workload specifically designed to provide performance results for E-commerce systems.*

In this thesis, two generalized dynamic caching policies, query result cache and table cache, are emulated. The TPC-W[4] benchmark is configured to drive a real database server under the three different dynamic caching policies. TPC-W is a Web E-commerce benchmark from transaction processing performance council. It simulates a breadth of activities of a retail store, specifically an on-line bookstore.

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

A trace tool in the DBMS is implemented and traces of database page references to the buffer pool (an in-memory buffer that caches database pages) are collected. Traces are then analyzed in terms of a variety of system characteristics. For example, the sequentiality of the requests is studied to examine how the database server can benefit from prefetching. Three TPC-W workload profiles are also examined under caching to investigate if caching changes the classification of these profiles.

1.7 Contributions

This thesis combines a variety of analysis techniques: simulation, real time measurement and data mining. This thesis also reveals some interesting effects that the dynamic caching has on the database server workload characteristics. The main observations include: (a) dynamic cache can considerably reduce the response time of the database server and the number of database page references when it is heavily loaded; (b) dynamic cache can also reduce the database reference locality; (c) table cache does a better job than query result cache in reducing the database server load. The data classification in this thesis shows that with dynamic cache, the database server sees different categories of workload profiles from those under no-cache configuration.

1.8 Thesis Overview

The remainder of this thesis is organized as follows. Chapter 2 gives the background of this study. Chapter 3 presents the related work. Chapter 4 describes the experimental parameters and methodology used to conduct this study. Chapter 5 and Chapter 6 present the experimental results concerning different dynamic caching approaches and their influence on database server workloads with respect to general workload characterization and workload classification. The conclusions and future work are presented in Chapter 7.

Chapter 2

Background

Several components of background material are presented in this chapter. An outline of DBMS components, buffer pool management, and workload characterization methodologies is presented in Section 2.1. A discussion of caching in E-commerce systems is presented in Section 2.2. An overview of the TPC-W benchmark used in this study is presented in Section 2.3.

2.1 DBMS

The database server machine in E-commerce systems is a specially designed computer that holds the actual data and runs only the database management system (DBMS) and related software. A DBMS [37] is designed to facilitate the storage, access, and management of the data, such as achieving efficient data access, data integrity, concurrent access, and recovery from crashes. The most typical DBMS is a relational DBMS. In a relational database system, the data is placed on disk(s) in the form of tables. A table consists of a set of fields and any number of rows. Certain fields are defined as keys to facilitate linking records of related tables. Applications create, update, extract, and administer the data via the DBMS. The Structured Query Language (SQL) is generally used as a standard programming language for the communication between applications and the DBMS.

2.1.1 DBMS Components and Buffer Pool Management

A relational DBMS usually consists of several core components: application interface, query processing, storage management, concurrency control, and recovery management [37]. The application interface passes the requests from normal users or the database administrator to the query processing component, which parses the requests, optimizes and converts them to logical read/write requests upon the table/index files stored on disks. The storage management component receives the logical read/write requests, and translates them into physical read/write operations if necessary. When a logical/physical read/write is performed, concurrency control mechanisms and recovery management techniques are used to achieve data sharing and transaction processing properties of atomicity, consistency, isolation, and durability (ACID) [37].

Storage management is essential to the performance of a DBMS, and it is the component of interest in this study. A DBMS manages the reading and writing of data pages to and from disk, called disk input/output (I/O), and a special memory area, named buffer pool. Although its function is similar to that of the file system cache in operating systems, the database buffer pool is maintained as a separate cache by the DBMS to reduce the performance overhead of the file system cache and facilitate crash recovery in a database environment [39]. Data pages from disks are held, read, and modified in the buffer pool to improve the database performance by allowing data to be accessed from memory instead of from disks, since disk access is of several magnitude slower than memory access.

Upon a data page reference, if the page is already in the buffer pool (called a *buffer hit*), no disk access is required; otherwise a *buffer miss* occurs and the disk device must be accessed. The buffer *hit ratio* is the ratio of the number of buffer hits to the number of total references (# of buffer hits + # of buffer misses), and the buffer *miss ratio* is the ratio of the number of buffer misses to the number of total references ($1 - $ *hit ratio*). A buffered page is accessed much faster than if it had to be read from disk.

It is ideal to keep all the data pages in the buffer pool to get a buffer hit. In many database systems however, the buffer size is much smaller than the database size. A small buffer size is cost effective since memory is more expensive than disks. Moreover, a much smaller buffer pool than the database size can still keep a very low buffer miss ratio due to the locality of data references [35]. In such systems, the buffer pool management policy maintains the buffer pool to minimize buffer misses. The buffer pool management policy depends on three fundamental techniques: page replacement, page cleaning, and page prefetching, to score more buffer hits.

Page Replacement. If the buffer pool is already full and a new page needs to be buffered, the page replacement algorithm kicks a target page out of the buffer pool to make room for the new page based on its policy. The DBMS page replacement algorithms are similar to the general buffer replacement approaches that have been used in virtual memory, file system, and CPU cache management. The goal of buffer pool replacement algorithms is to maximize the buffer hit ratio so as to minimize the physical I/Os for a given buffer size.

Three basic buffer page replacement algorithms are the following: first-in first-out (FIFO), least recently used (LRU), and least frequently used (LFU). When a new page requests buffer space and the buffer pool is already full, FIFO evicts the page that has stayed in the buffer for the longest time, while LRU selects the page that has not been referenced for the longest time, and LFU kicks out the one that has lowest reference frequency.

The FIFO algorithm is easy to implement and has low overhead, but it does not perform well since it cannot recognize any locality or frequency characteristics of the workload references. Although the LRU algorithm is simple and shows good performance for many workloads, it has many disadvantages. It is unable to capture the frequency feature of a workload and its performance can degenerate dramatically upon a big scan, which is a sequence of successive page requests with only one-time use. On every page reference, the LRU algorithm needs to update its most recently used position. This updating work may cause unacceptably high contention in a multiple threads environment [10]. The LFU algorithm may never replace some

buffered pages with high reference frequency even though those pages will not be referenced in the future.

The DBMS uses a special FIX-UNFIX mechanism to prevent unexpected replacement [14]. If any page in the buffer pool is requested, a FIX operation is performed on the page, and an UNFIX is performed on the page after the request. During the FIX-UNFIX interval, that page cannot be replaced from the buffer pool. This mechanism guarantees the page is addressable for the duration of a query.

Page Cleaning. The data pages that have been modified after being fetched into the buffer pool are called *dirty pages*. If a dirty page is the target replacement page, the buffer pool manager first writes that dirty page back to the disk (called *page cleaning*) before it is evicted from the buffer pool. An asynchronous page cleaning mechanism is generally deployed to reduce the waiting time of writing back the dirty pages. It writes back the dirty pages before they become target replacement pages.

Page Prefetching. To further reduce the time spent waiting for disk I/Os to complete, the buffer pool manager also prefetches some pages into the buffer pool before they are requested. For example, if a scan through large volumes of data pages is predicted, the page prefetching policy will prefetch the corresponding pages to the buffer pool before the actual references arrive. An effective page prefetching policy depends on accurately anticipating the application's data referencing patterns.

2.1.2 Parallel DBMS and Distributed DBMS

As the amount of data and the number of concurrent users handled by a DBMS increase, the scalability of traditional DBMS that resides on a single powerful machine becomes a challenge. It is hard for a traditional DBMS to meet the response time requirement while searching terabytes of data and processing queries from tens of thousands of simultaneous users. A traditional DBMS can be replaced with two types of extended DBMSs to address the scalability challenge: parallel DBMS and distributed DBMS. Although a detailed investigation of parallel DBMS and distributed DBMS is beyond the scope of this study, a brief description of these two

techniques is presented in this section to make this thesis self-contained.

Parallel DBMS. A parallel DBMS is a DBMS implemented on a tightly coupled multiprocessor [34]. The system architecture for a parallel DBMS can range among shared-memory, shared-disk, and shared-nothing architectures, based on how a processor shares the memory and disks among its peers. Multiple processors share memory and disks via fast interconnects in the shared-memory architecture. This architecture is used in this study for experiments. In the shared-disk architecture, each processor shares disks, but has its own protected memory. In the shared-nothing architecture, each processor has its own memory and disks, which makes this architecture highly scalable. For a shared-nothing architecture model, a parallel DBMS is quite similar to a distributed DBMS. The only possible distinction is that a distributed DBMS supposes each processor runs its own operating system while all the processors run on a single operating system in a parallel DBMS environment [34].

Distributed DBMS. A distributed DBMS is a DBMS implementation that manages a collection of multiple, logically interrelated databases distributed over a computer network [34]. For a distributed DBMS, the data is partitioned horizontally and/or vertically to fragments across geographically distributed data sites. Such systems usually involve heterogeneous hardware/software. Data can also be replicated at a number of sites to make it close to applications.

To provide applications with transparent functionalities as a centralized DBMS does, parallel and distributed DBMSs face many challenges, such as query optimization and concurrency control [34]. Due to the parallel processing and/or the distribution of the data, these challenges are more difficult to attack than in a centralized DBMS. Further discussions on the open issues about parallel DBMS and distributed DBMS are beyond the scope of this study.

2.1.3 DBMS Workload Characterization

A DBMS needs to be tuned to handle its workloads with a good performance [15, 19, 37]. Knowing what kind of workload a DBMS must support can provide insights

for database administrators and guide their decisions on configuring a DBMS to get good performance. For example, workload characterization can help to determine how much computing power is required for a DBMS and identify the relationship between the workload and the Quality of Service (QoS) a DBMS can provide [31].

2.2 Caching in E-commerce System

For E-commerce systems, two general caching mechanisms exist: traditional static caching and dynamic caching. These two caching techniques are introduced in this section.

Static Caching. The traditional static caching technique stores one entire Web page as a data object, including the query results from the database server [43]. The query results are generated from the database server when a user performs a Web interaction by sending out a request to the Web server, such as requesting the homepage of a Web site. If a subsequent request is exactly the same as the previous one, the stored page will be served instead of repeatedly sending the request to the server.

Static caching is not a practical caching method for an E-commerce system, since it has many drawbacks. Static caching identifies cached content (static objects and dynamic generated data from database server) by the corresponding URL, but even for the same URL, the dynamically generated Web pages may not be the same because of different HTTP request headers. Static caching may also produce much redundant data since the contents associated with different URLs can share many components. In addition, to make the cached content consistent, static caching needs to refresh one entire document at the highest update frequency among all of its components. This incurs much overhead. Consequently, static caching is not further discussed in this thesis.

Dynamic Caching. Dynamic caching, on the other hand, is widely used in E-commerce systems, aiming at reducing the load on the database server and the Web user response time. It stores the query results from the database server or duplicates

some database tables at the cache for later use [32]. Dynamic caching policies can be roughly divided into two categories from the point of view of the database server: *query result caching* and *table caching*[1]. The following overview explores how these two dynamic caching mechanisms work in an E-commerce system. It describes what a certain dynamic caching mechanism will cache while a common user performs Web interactions, and what are the advantages and disadvantages of this dynamic caching mechanism.

For query result caching, the query results from the database server are cached. These cached results can be reused if identical queries (URL and corresponding query parameters) are sent to the dynamic cache again. These cached results become invalid after a certain time or they can be invalidated explicitly when the data generating these results change. The query result cache can be very simple since it does not involve the DBMS. The query result cache can save much computation at the database server on a cache hit when the queries are expensive. Only exactly repeated queries can be found and served in the cache.

A table cache is a DBMS with replicated tables from the database server. The database server periodically propagates changes to these tables to the table cache. Only infrequently changed tables are suitable cache candidates so that the update propagation costs do not dwarf the benefits of caching. Since the table cache has its own DBMS, it can answer arbitrary queries involving its cached tables. Queries partially involving the cached data can also be rewritten so that part of the query is performed at the table cache. Compared with query result cache, however, table cache cannot save expensive computation, and the query processing capacity of table cache requires more computing power than query result cache.

[1] To make it clear, *caching* is used to refer to the policies in this study, while *cache* is used to refer to the software/hardware components that implement/hold the data stored by caching policies.

2.3 TPC-W: An E-commerce Benchmark

TPC-W [41] is a widely used benchmark to measure and compare the performance of a transactional E-commerce Web site. It provides various system performance metrics under different loads and hardware/software deployment platforms. The metrics are examined to determine how the configuration of a system should change to correspond to changes in system workload.

The TPC-W workload simulates a breadth of activities of a retail store, specifically an on-line bookstore. Customers visit the bookstore Web site, browsing through pages, looking for some books, placing an order, checking the status of an existing order, or canceling an order. Administrations of the Web site (for example updating a product price) are also integrated into the workload model.

2.3.1 TPC-W Database

The TPC-W database includes eight tables listed below:

- CUSTOMER: Customer personal and session data.
- ADDRESS: Customer shipping address information.
- ORDERS: Order information, including total amount and shipping data.
- ORDER_LINE: One order line data per order.
- CC_XACTS: Credit card transaction data.
- ITEM: Description of each item (book) in the inventory.
- AUTHOR: Author data, including first name and last name.
- COUNTRY: Country name and exchange rate.

According to the TPC-W specification, additional tables can be included in the implementation to facilitate the execution of TPC-W workloads. A typical example is the *SHOPPING_CART* for the shopping cart Web interaction, which is generally

implemented in the database to help to meet the TPC-W durability requirement [41], so the system can preserve the effects of any committed database transaction after recovery from any single point of failure. A customer can add a new item to his/her shopping cart or update existing items in his/her shopping cart.

Table 2.1: TPC-W Database Scaling Rules with Example Table Sizes

Table Name	**Cardinality (in rows)**	**Typical Row Length (bytes)**	**Typical Table Size (bytes)**
CUSTOMER	2880*(number of EB)	760	2,188,888k
ADDRESS	2*(CUSTOMER)	154	887,040k
ORDERS	0.9*(CUSTOMER)	220	570,240k
ORDER_LINE	3*(ORDERS)	132	1,026,432k
CC_XACTS	1*(ORDERS)	80	207,360k
ITEM	1k, 10k, 100k, 1M, 10M	80	207,360k
AUTHOR	0.25*ITEM	630	1,575k
COUNTRY	92	70	6.44k

The store size is expressed by the number of items in the *ITEM* table and the number of emulated browsers (EBs), which emulate users using Web browsers to interact with an E-commerce Web site. The *database scaling* for TPC-W is defined by the size of the supported customers and the cardinality (number of rows) of the ITEM table [41]. The scaling rules are illustrated in Table 2.1, along with typical row lengths and table size estimates for 10,000 items and 1,000 EBs.

2.3.2 TPC-W Workload Profiles

Based on the user traversal probabilities defined in TPC-W [41], an emulated browser goes through one or more of 14 Web interactions to communicate with the E-commerce Web site. A Web interaction is a complete cycle of communication between the EB and the E-commerce system. It starts when the EB requests a Web page and finishes when the last byte of data from the response page has been received by the EB. The 14 Web interactions defined in TPC-W are: *Home, New Products, Product Detail, Best Sellers, Search Request, Search Results, Customer Registration, Shopping Cart, Order Inquiry, Order Display, Buy Request, Buy Confirm, Admin*

Request, and *Admin Confirm*.

Table 2.2: TPC-W Workload Profiles

Web Interaction	Browsing Mix	Shopping Mix	Ordering Mix
Browse	**95%**	**80%**	**50%**
Best Sellers	11.00%	5.00%	0.46%
Home	29.00%	16.00%	9.12%
New Products	11.00%	5.00%	0.46%
Product Detail	21.00%	17.00%	12.35%
Search Request	12.00%	20.00%	14.53%
Search Results	11.00%	17.00%	13.08%
Order	**5%**	**20%**	**50%**
Admin Confirm	0.09%	0.09%	0.11%
Admin Request	0.10%	0.10%	0.12%
Buy Confirm	0.69%	1.20%	10.18%
Buy Request	0.75%	2.60%	12.73%
Customer Registration	0.82%	3.00%	12.86%
Order Display	0.25%	0.66%	0.22%
Order Inquiry	0.30%	0.75%	0.25%
Shopping Cart	2.00%	11.60%	13.53%

TPC-W classifies the 14 Web interactions into two groups: *browse* and *order*. The *browse* Web interactions involve browsing the Web site and searching the database (e.g., querying new products, best sellers, and product details). The *order* Web interactions update the database, e.g., loading shopping carts, and registering customers. By varying the ratio of the *browse* to *order* Web interactions, TPC-W simulates three kinds of workloads: *browsing*, *shopping*, and *ordering*, which have been shown to be common Web use profiles [44]. Table 2.2 summarizes the exact Web interactions frequencies in each type of workload. The performance metric reported by TPC-W is the number of Web interactions per second (WIPS).

2.4 Summary

In this chapter, some DBMS related issues are discussed: DBMS components, parallel DBMS and distributed DBMS, and DBMS workload characterization. A general description of static caching and dynamic caching in E-commerce systems is also

presented, and dynamic caching is shown as a suitable solution to E-commerce systems. The widely used TPC-W benchmark is introduced in the last section from both database point of view and workload point of view.

Chapter 3

Related Work

This chapter covers the related work of this study. First, caching effects in the context of file systems, Web systems, and storage systems are presented in Section 3.1. The previous research in dynamic caching mechanisms in E-commerce systems is then presented in Section 3.2. Third, the related studies in DBMS buffer pool and workload are given in in Section 3.3. Finally, the questions that remain unanswered with the previous research are listed.

3.1 Caching Effects

It has been shown that caching is an effective technique to reduce client access delays and network load as well as CPU utilization at the server side in a distributed file system [33]. Similarly, traditional static Web caching is designed to reduce Web latency, network traffic, and Web server load [43]. Previous studies have found that Web caching mechanisms can alter the Web workload characteristics, such as request arrival rate [4], object concentration [28], and object popularity [48]. This is because only the requests that cannot be served by a Web cache are forwarded to its server-side systems.

Further research [13, 16] has found that Web caching mechanisms reduce the temporal locality of workloads. Similar effects of reducing the temporal locality of workloads have also been found in the context of distributed file systems [17] and CPU caching systems [47]. Some studies [49, 54] have found that a higher level

cache makes the LRU cache replacement algorithm inappropriate for the lower level cache system. As a result, the designers of lower level caching mechanisms for Web systems [7] and storage systems [8, 51, 55] must keep the caching effects of the higher level caching in mind.

This study is similar to the above research with respect to investigating the effects of caching. It examines how various caching mechanisms change the workload. It also analyzes what degree the workload should influence the design and tuning of the lower system. However, it differs from the above research in two ways. First, the caching under investigation in this study is *dynamic caching*, instead of traditional static caching. Second, the underlying system under study is a database server, instead of file servers, Web servers, or storage systems.

3.2 Dynamic Caching Mechanisms

Various dynamic caching mechanisms have been proposed to reduce Web user response time, network bandwidth load, and Web/application/database server load [32]. This section gives a brief overview of the recently proposed dynamic caching policies and classifies them into two general categories: *query result caching* and *table caching*. In this study, these two types of dynamic caching approaches are emulated based on their respective characteristics.

A *query result* cache stores Web page layout fragments and content components from database server independently [3, 11, 12, 27]. The cached content components can be shared among different layout fragments. Upon receiving a request, the related layout fragments and content components are built together to form a full Web page. Query result caches require simple computation and avoid re-computation for the repeated queries at the database server. A materialized view can also be cached to speed up query processing [52]. A view is a virtual table that consists of data that is pulled out of one or more existing tables by a query. If stored as a file, a view is called a materialized view.

Caching a materialized view is classified as a query result caching method in

this thesis because it works in a similar way by caching content components [52]. Upon receiving a request, the table cache mechanism checks whether the request refers to the tables it has cached. The query result cache mechanism checks whether the corresponding object has been cached. If the request cannot be served by the caching mechanism, it will be forwarded to the database server.

Query result caching mechanisms can significantly reduce the user response time by 20% to 50% [11, 27, 52]. If the cache is located near the Web users, query result caching can save network bandwidth by about 30% when the cache hit ratio is about 40% [12]. Also, query result caching can save up to 90% of the computation at the E-commerce site [3].

Table caching mechanisms usually store infrequently updated but frequently used tables or even subsets of the tables from the database server [1, 40]. A table cache demands powerful computation since it needs to analyze and answer a query in the way as an actual DBMS does. A table cache can be implemented as either a simplified in-memory database [40] or a full-fledged commercial DBMS [1]. Table caching is shown effective at reducing user response time up to 100% and increase the Web interaction throughput of the E-commerce site up to 200% [1, 2, 32].

A dynamic cache can be deployed at a proxy outside the E-commerce site infrastructure [3, 11, 12, 27] to move the content much closer to Web users; it can also be implemented in the E-commerce site infrastructure [1, 40, 52] to ease the work of maintaining the cached content. Cache location makes no difference however, on how a dynamic caching mechanism reduces the load seen by the database server.

The concern of this thesis is the impact that dynamic caching mechanisms have on the database server instead of the network load. In this thesis, the locations of dynamic caches are not distinguished in the simulation of dynamic caching mechanisms.

3.3 DBMS Buffer Pool and Workload Studies

Dynamic caching mechanisms can affect the workload on the database server and the tuning/design of the database server in many ways. In this study, the locality of the requests, the I/O activities of the database server, the buffer pool replacement algorithms, and the classification of the workload are considered. This section briefly outlines the related studies on these topics.

3.3.1 DBMS Buffer Pool Replacement Algorithms

In addition to the simple buffer pool replacement algorithms such as LRU, LFU, and FIFO, more complex algorithms have been proposed in recent work in this area. Two representative algorithms are low inter-reference recency set (LIRS) [23] and adaptive replacement cache (ARC) [30]. These algorithms are briefly explained in this section.

The LIRS strategy [23] keeps track of both the reference recency and the inter-reference recency (IRR) for each buffered page. The IRR is defined as the *number of references* to other pages between the last two *consecutive* references to the page. The recency is the difference between the time of the last reference and the current time. In LIRS strategy, the buffer is divided to a large buffer known as L_{lirs} and a small buffer known as L_{hirs}. Pages with low IRR are registered in L_{lirs}, and those with high IRR are registered in L_{hirs}. The registration status of a buffered page is switched when a page reference changes the IRR of the buffered pages to a certain threshold level. The pages registered in L_{hirs} are evicted from the buffer based on the FIFO policy.

The ARC algorithm [30] automatically balances the recency and frequency of the page references by maintaining two LRU lists: L_1 and L_2. L_1 contains the pages that have been referenced only once recently, while L_2 contains the pages that have been referenced more than once recently. The p most recent pages at the top of L_1 and c-p most recent pages at the top of L_2 are kept in the buffer, where c is the buffer size and p is a tunable parameter in the algorithm. The victim page is picked

from either L_1 or L_2, which is based on the tunable parameter p, the number of pages in the buffer from L_1, and the number of pages in the buffer from L_2. The pages at the bottom of L_1 and L_2 are maintained in the buffer directory but not in the buffer.

The performance of the proposed buffer pool replacement algorithms for some traditional workloads is already studied in the literature. However, how they perform for the E-commerce workloads in the presence of dynamic caching mechanisms is not addressed. This study aims at answering the question.

3.3.2 Database Buffer Pool Simulation

Wang and Bunt [45] built a DB2 buffer pool management simulator to study buffer pool management algorithm performance under a variety of system parameters. The simulator includes four components: client agent, buffer pool manager, page cleaner, and disk I/O handler. Based on the trace files it reads, the client agent sends out page requests to the buffer pool. The buffer pool manager is responsible for placing and replacing the buffer pool pages and requesting disk I/O. Dirty pages are cleaned by the page cleaner, and disk I/O requests are processed by the disk I/O handler.

The TPC-C benchmark was used as the input to DB2. The page requests of FIX/UNFIX at the database buffer pool were recorded. Comparisons of performance metrics between the trace-driven simulation results and the real system experimental results verified the buffer pool simulator. The results show that the simulator provides a good match with the real system. One of the example evaluations was the buffer pool hit ratio difference between the measured result and the simulated result. The difference reported by Wang and Bunt [45] is about 0.1%.

This simulator provides a good tool for the study of database server workload characteristics under caching. An extended version of the simulator obtains the information about the number of page requests, working set size, buffer pool miss ratio, and the number of disk I/Os from the traces collected in this study.

3.3.3 DBMS Workload Studies

Hsu *et al.* [19] implemented a light-weight tracing utility for DB2 at the buffer pool level to record the requests from the buffer pool manager. Traces from ten real production database workloads and TPC benchmarks (TPC-C and TPC-D) are comprehensively analyzed. TPC-C simulates a complete environment where a large number of users executes transactions against a database, while TPC-D represents a broad range of decision support applications that require complex, long running queries against a database[1]. TPC-C and TPC-D complement each other since they represent different kinds of workloads. However, they still cannot reflect some aspects of the real production database workloads, for example the burst of I/O bandwidth. The tracing utility developed by Hsu *et al.* [19] is used in this thesis to collect the buffer pool level traces.

Hsu *et al.* also studied the logical level I/O reference behaviors of production database workloads and TPC benchmarks [20]. They investigated workloads under different I/O optimization techniques: caching, prefetching, and write buffering. Hsu found that the buffer pool miss ratio is approximately the inverse square root of the ratio of buffer pool size to data size, and that TPC-C does not present any significant sequentiality. The logical level I/O reference behaviors of TPC-W, especially under different caching mechanisms, remain unknown. The experiments in this study will investigate the I/O reference characteristics of TPC-W and evaluate whether the observations in Hsu *et al.*'s research apply to TPC-W.

Elnaffar [15] presented a classification model to automatically categorize a DBMS workload to be an OLTP workload or an OLAP workload. TPC-C, TPC-H, and the TPC-W *Browsing* profile and the *Ordering* profile were used to train the classifiers. The system performance snapshots that are taken during the execution of the workloads, were used as the data objects to build the classifier. Elnaffar chose the TPC-W *Browsing* profile and the *Ordering* profile to represent OLAP and OLTP workloads respectively [15]. Wasserman et al. [46] revealed there are four different

[1] www.tpc.org

groups of queries in a typical OLAP workload. The first group contains queries which have trivial complexity, short run times, and high CPU utilization. The second group is identified by simple-complexity queries that involve intensive I/O activities. The third group is characterized by medium-complexity queries that demand moderate CPU and I/O usage. The fourth group is identified by large and complex queries that show high sequentiality and random I/O usage. Unfortunately, dynamic caching was not considered in either study. Dynamic caching may change the E-commerce workload profiles (Browsing, Shopping, and Ordering) as it can filter out a large portion of queries.

3.4 Summary

In a typical E-commerce system, various dynamic caching mechanisms have been proposed to reduce database server load and/or the network latency. The database server scalability and better response time are achieved in the challenging environment that the database server is becoming the performance bottleneck due to the increasing number of Web users. Related DBMS studies are also presented. However, new questions arise:

- Do the proposed dynamic caching mechanisms change the workload characterization seen by the database server? If so, how? Specifically, does the reference locality of the workload change? Among the buffer pool replacement algorithms mentioned in this chapter, which one has the best performance under dynamic caching? How are the I/Os affected?

- After dynamic caching, how can one classify the E-commerce workloads? Do they behave more like OLTP?

- If the proposed dynamic caching mechanisms do change the workload characterization and/or classification seen by the database server, does this have a impact on system tuning, configuration, and design of the DBMS?

Although a good understanding of these questions is a prerequisite for designing and tuning the database server, the questions are not widely addressed in the literature.

Chapter 4

Methodology and Experimental Setup

This chapter contains the methods used in this study to characterize and classify the TPC-W profiles under dynamic caching. The experimental setup for the evaluation experiments is also described in detail.

4.1 Methodology

In this study, dynamic caching mechanisms are simulated. For the first set of experiments, database traces are collected during the running of TPC-W profiles. Also, system performance parameters are measured, such as response time and CPU utilization. Then, the workload characterizations are carried out using trace-driven simulation. For the second set of experiments, database snapshots are recorded during the running of TPC-W profiles. The snapshots are then used as profile classification objects.

4.1.1 Simulation of Dynamic Caching

Dynamic caching mechanisms are simulated instead of being implemented in a full-featured dynamic cache, since the performance of the database server is the focus of this study. This greatly simplifies the design of the dynamic cache without affecting the actual workloads reaching the database server. The simulation of dynamic caching mechanisms is implemented to represent their caching functionality.

Two dynamic caching policies are simulated: *query-result cache* and *table cache*. They are simulated in the Java implementation of TPC-W kit [6]. Some query results from the database server are stored at the simulated query-result cache, and some tables from the database server are stored at the simulated table cache.

For the simulated *query-result cache*, a query from the Emulated Browser is checked to see whether a corresponding result is resident in the cache before being sent to the database server. If a valid stored result for the query exists in the simulated cache, the cache serves the query with the result; otherwise, the query is forwarded to the database server.

For the simulated *table cache*, a query from the Emulated Browser is checked to see if it requires the stored tables before being sent to the database server. If the query requires the cached tables, the cache serves the query by performing the query on the related tables; otherwise, the query is forwarded to the database server. A query is rewritten so that only parts of the query are performed on the database server, if it involves both cached tables and un-cached tables.

4.1.2 Workload Characterization

Trace-driven simulation is used in this study to examine the effects that dynamic caching mechanisms have on database server workload. A trace-driven simulation consists of two main stages: trace collection and trace processing. In the first stage, relevant information about a system is collected while the system executes the workload of interest, which is achieved either by using hardware probes or by instrumenting the software. In the second stage, the resulting trace of the system is played back to a model of the system under study. More discussions of this technique and its strengths and weaknesses can be found in Jain [22].

In this study, the TPC-W Browsing, Shopping, and Ordering profiles under dynamic caching are used as workloads on a real database server, and the resulting DBMS traces are collected. Then the traces are analyzed by feeding them to a simulator. Although a benchmark E-commerce system is used rather than a real

system, the analysis methodology employed in this study is general. The benchmark system provides a controlled environment to emulate most key features that affect the performance of real E-commerce systems. It can be used to study general characteristics that are not restricted to any specific system.

Some performance measurements are conducted during the running of TPC-W profiles. The response time is measured at the emulated browsers. The cache hit ratio is calculated at the simulated query-result cache. The CPU utilization of database server is also recorded. Details are given later in this chapter.

4.1.3 TPC-W Profile Classification under Caching

A set of experiments are also performed to investigate the classification of TPC-W profiles under dynamic caching. Classification is a common machine learning and data mining task. Given "classified" cases, a classification model can build its rules to predict new "unclassified" cases. Decision tree models are one of the most common classification methods. A decision tree model can extract the classification rules easily and enables the user to understand and justify the results in a more straightforward manner in comparison with other techniques such as neural networks [50].

A decision tree model is built by analyzing training data and then the model is used to classify target data. The decision tree is constructed in a top-down fashion by choosing the most appropriate attribute each time. In a decision tree model, an internal node is a test on an attribute of a case. A branch represents an outcome of the test, and a leaf node represents a class label, i.e. a prediction result. When it is trained with "classified" cases, a decision tree model chooses one attribute at each internal node to split the training cases into the labelled classes. Then the trained decision tree model classify target cases by following a matching path to a leaf node.

The classification of DBMS workloads can be viewed as a data mining process using decision tree models [15]. The process includes two steps: training a decision tree model using "classified" cases and applying it to target classification candidates

to predict. Database snapshots collected during the execution of TPC-W profiles *without caching* are used as data objects to train the classification model in this study. Snapshots *under caching* are then used as inputs to the trained classifier for prediction.

A database snapshot includes important performance statistics of a database server. It records the running state of the database environment, such as queries proportion, pages read, and rows selected. Each snapshot can be viewed as a vector whose dimensions are the attributes that summarize the activities in the system during the execution. More details are provided in the latter portions of this chapter.

4.2 Experiment Design

4.2.1 Experimental Architecture

Figure 4.1 shows the experimental architecture of this study. It is composed of *Emulated Browsers* (EBs), *application logic*, *dynamic cache*, and *database server*. The Emulated Browsers are implemented in Java and can emulate many browsers performing Web interactions. The application logic converts Web requests to SQL queries. The dynamic cache serves queries or forward them to the database server.

In the original implementation of the system [6], the application logic was developed as servlets that run on a real Web server (specifically Jigsaw 2.2.2). All inventory images were stored on the Web server, while only their names were stored in the database. In this study, the Web server is removed and the application logic is implemented as a set of procedures called directly by EBs. The EBs do not request inventory images nor HTML pages. The purpose of these modifications is to remove the overhead incurred by the Web/application server and to be able to focus on the performance of the database server alone.

Two sets of experiments are conducted in this study. The first set investigates how dynamic caching mechanisms change the TPC-W workloads seen by the database server in terms of various performance metrics. The second set examines

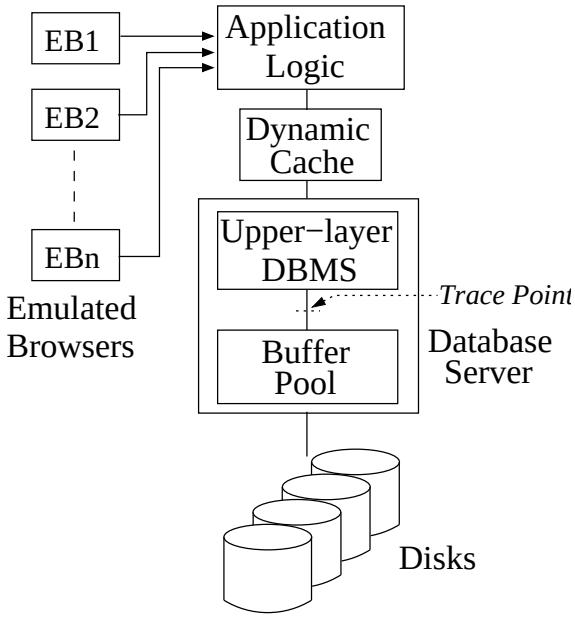


Figure 4.1: The Experiment Architecture

how dynamic caching mechanisms alter the classification of TPC-W profiles.

4.2.2 Metrics for Workload Characterization

Figure 4.2 shows the metrics and the corresponding experiments in the first set. The metrics include response time, CPU utilization, cache hit ratio, number of references, working set size, disk I/Os, reference temporal locality, and reference spatial locality. Multiple buffer pool replacement algorithms are also studied under dynamic caching mechanisms. The algorithms include LRU, LIRS, and ARC. The buffer pool miss ratio is used to study the performance of the algorithms.

Response time is a fundamental metric in E-commerce systems. In this study, the average response time received at the EBs is used as a key metric to reflect when the database server becomes overloaded while the number of EBs increases. The logic behind this assumption is that if the response time is higher than a threshold, the system is considered overloaded. In this study, the response time for every interaction a EB performs is recorded and saved in a file (as shown in Table 4.5). The average response time for all the EBs is then calculated by analyzing the file.

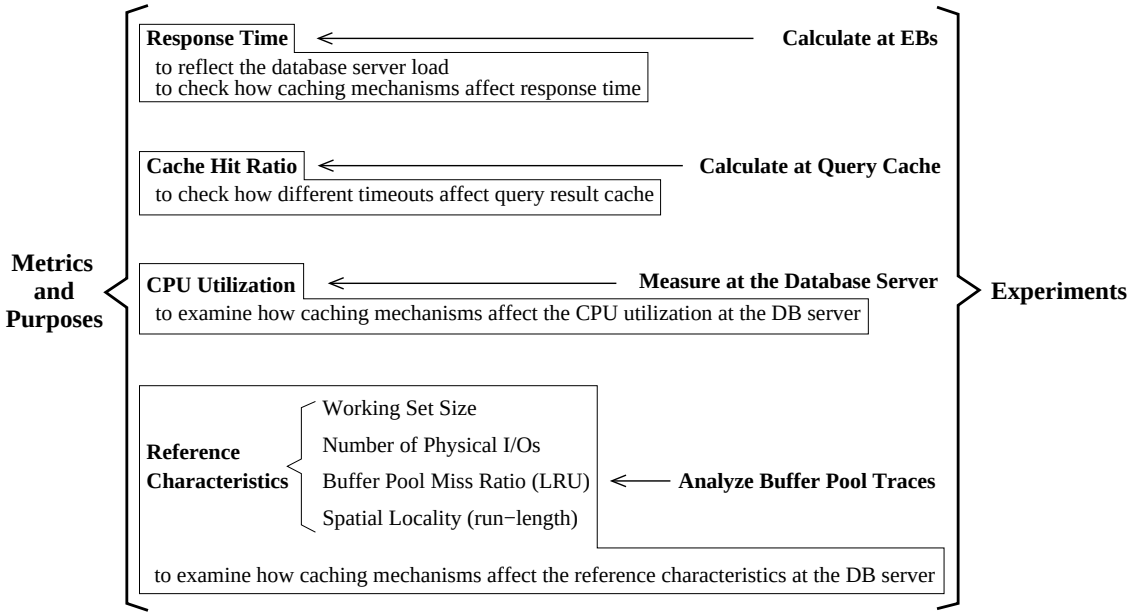


Figure 4.2: The Metrics and Corresponding Experiments

Cache hit ratio is measured for query-result cache to reveal the relationship between the response time and the cache hit ratio. This metric is not reported for table caching for two reasons. First, it is difficult to calculate the number of hits for table caching since some queries may be partially served by the table cache. Second, cache hit ratio makes little sense for table caching from saving overall computation and I/O point of view. Cache hit ratio for query-result cache is calculated at the *Dynamic Cache* layer in Figure 4.1.

The DBMS CPU time is measured to examine how much a dynamic caching approach can reduce the computation at the database server. In this study, the CPU utilization of the DBMS, specifically the process of *db2syscs*, is recorded and saved as a file by *Windows Performance Monitor*. The *Windows Performance Monitor* samples and records the CPU utilization of the DBMS for every 5 seconds. Then the average of 180 samples (15 minutes) after warmup is used.

The traces at the database buffer pool level are recorded (as shown in Figure 4.1 at the *trace point*). Reference characteristics of the database workloads under dynamic caching mechanisms are investigated by using buffer pool simulation program

bps and trace analysis program *rl* [45]. The *bps* program calculates the number of references, the working set size, the number of disk I/Os, and the buffer pool miss ratio for a specified trace. The *rl* program analyzed the spatial locality by calculating the *run length* of the references in traces.

Database working set size is the size of the unique data that is accessed by users during a certain mount of time. The data includes table pages and index pages in the database. A large working set size during a certain time period can cause a heavy load on database server since it generally demands more memory accesses, I/Os, and CPU computation. If dynamic caching mechanisms can reduce working set size of a workload, the resource requirements on database server will be reduced.

Reference temporal locality is an essential factor for managing the DBMS buffer pool. The LRU miss ratio at the buffer pool is used as the metric to measure the temporal locality of the workloads under caching. Several buffer pool replacement algorithms are also evaluated using the buffer pool simulator. They include LRU, LIRS, and ARC. The buffer pool miss ratios are measured for these algorithms.

The *run length* is the number of references in a *sequential run*, which is a sequence of consecutive *reduced page references* [38]. Assume a database consists of n data pages: p_1, p_2, p_3, ..., p_n. An example of the process converting a sequence of page references to sequential runs is shown below.

Sequence of page references:	$\{p_1, p_3, p_4, p_4, p_5, p_6, p_6, p_6, p_7, p_{15}, p_{16}, p_{16}, p_2\}$
Reduced page references:	$\{p_1, p_3, p_4, p_5, p_6, p_7, p_{15}, p_{16}, p_2\}$
Sequential runs:	$\{p_1\}, \{p_3, p_4, p_5, p_6, p_7\}, \{p_{15}, p_{16}\}, \{p_2\}$

By removing immediate re-reference(s), the sequence becomes a sequence of reduced page references. There are 4 sequential runs for the above example and the *run length* for each sequential run is 1, 5, 2, 1 respectively. In each sequential run, either there is only one page of reference or the page numbers are consecutive.

A sequential run with a long run length means the workload has good sequentiality. Sequential runs with short run length indicate random references. If a majority of the sequential runs in a database workload have short run lengths, it indicates that the references show weak sequentiality. In this study, run length is used to ex-

amine the sequentiality of TPC-W workloads and investigate how dynamic caching mechanisms affect the sequentiality of TPC-W workloads. More specifically, the *cumulative run length distribution* is used as the indicator of sequentiality in this study. Given reference sequence s, the *cumulative run length distribution* is defined as follows:

$$F(s;x) = \Pr\{\textit{run length} \leq x \text{ in reference sequence } s\} \text{ [24]}.$$

4.2.3 Metrics and Procedure for TPC-W Profile Classification

The classifier uses database snapshots for training and prediction. Distributions of the attributes of snapshots are analyzed under caching. The attributes in a training or target snapshot include the following measurements:

- *Queries Proportion.* This is the proportion of SELECT statements to UPDATE/INSERT/DELETE statements. It is generally higher in OLAP than in OLTP.

- *Pages Read.* OLAP transactions usually access larger portions of the database than OLTP transactions do.

- *Rows Selected.* OLAP applications tend to select more rows than OLTP applications.

- *Number of Sorts.* OLAP transactions typically perform a larger number of sorts than OLTP transactions do.

- *Ratio of Index Usage.* This is the ratio of data pages obtained from indexes to the pages obtained from tables, in order to serve a query. This ratio is expected to be higher in an OLTP workload than in a OLAP workload since OLTP applications do a lot of index scans while OLAP applications involve a large number of table scans.

- *Logging.* This is the number of pages read from or written to the log file of the database. An OLTP workload generates more logging activity than an OLAP workload does because the OLTP workload tends to have relatively more UPDATE/INSERT/DELETE transactions.

The above attributes represent the behavioural differences between OLAP workloads and OLTP workloads. They help the classifier build its classification rules for prediction. A raw database snapshot also includes other attributes such as Average Sort Time and Number of Locks Held. Those attributes are not included in the snapshots used as training cases or target cases however, because they are highly system dependent so they cannot accurately represent the distinguishable characteristics between OLAP workloads and OLTP workloads [15].

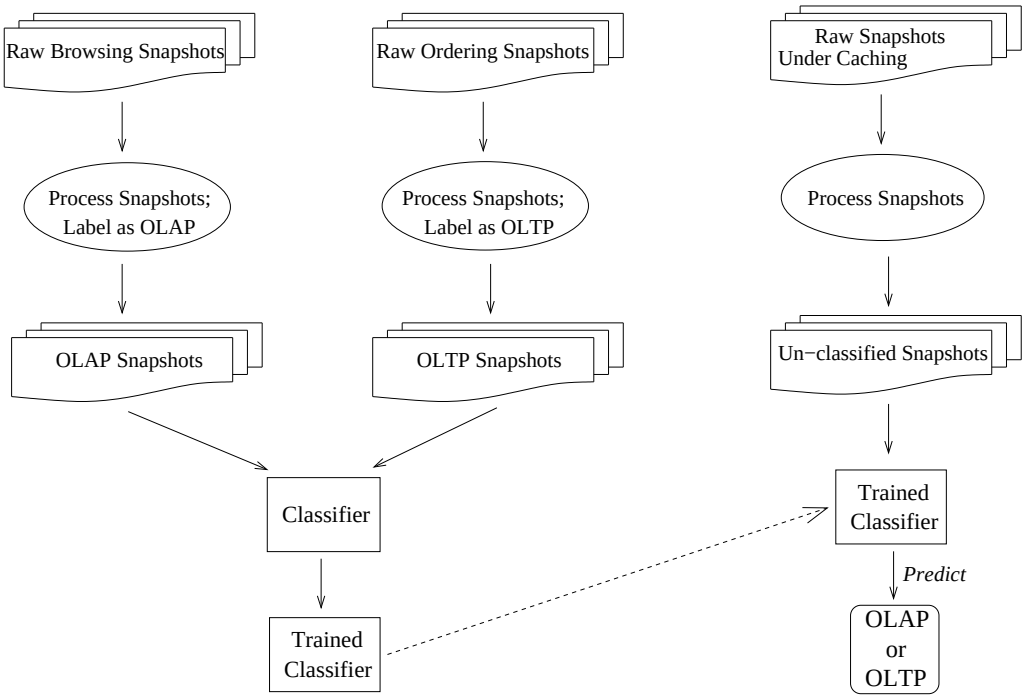


Figure 4.3: The Classfication of the TPC-W Profiles

Figure 4.3 shows the experimental procedure for workload classification. In the first step, database snapshots from TPC-W Browsing profile and Ordering profile without caching were collected. They were processed and labelled as "OLAP" workload and "OLTP" workload respectively. The classifier takes the labelled workloads

as training data to help establish rules for the classification of OLAP and OLTP. In the second step, database snapshots from TPC-W profiles under dynamic caching mechanisms were also collected and processed. Then, the trained classifier is applied to these "unknown" database snapshots. It predicts a snapshot either a "OLAP" snapshot or a "OLTP" snapshot.

4.3 Experiment Configuration

In this study, the emulated browsers, application logic, dynamic cache, and database server are all located on one server machine. This setup removes the network latency between EBs and the database server. It is also easier to automate the experiments since all the components are on one machine.

To get good performance at the database server, the original implementation has been optimized. For example, optimizations have been made to some SQL queries implemented in JDBC. Some queries selected all the fields from tables by doing *"SELECT *"* even when many fields are not necessary. These queries were revised so that they just select the necessary fields. Some queries did joining on tables first, then sorting all the rows, but only taking a small number of the results at the end by doing *"FETCH FIRST 50 ROWS ONLY"*. These queries were revised so that they sort first, then take the necessary rows, and do the joining at the end. This avoids accessing unnecessary rows. All experiments conducted in this study were performed on the optimized implementation.

4.3.1 Dynamic Caching Simulation

Query-result Cache. Query-result cache stores the query results of some read-only queries to the database. All queries to the database are analyzed. The queries that search infrequently updated data (such as author and title search) and the queries that are allowed to be cached for a certain time (such as best sellers search) are identified. The query-result cache is checked for these cache-able queries when they arrive. If the result of a query with the same parameters is already in the cache,

the cached result is returned to the client. Otherwise, this query is first sent to the database server and its reply is then stored in the cache. The cached query results become invalid after a certain timeout period. When the cache is full, the LRU replacement policy is used to evict least recently used query results of the cache until the cache has enough free space for the new query result. The cached data is maintained in memory as a *hash table*. The effectiveness of the query-result cache can be controlled by using different cache sizes and timeout values.

Table Cache. All infrequently updated tables – CUSTOMER, ADDRESS, ITEM, AUTHOR, and COUNTRY – are considered cache-able. If a query accesses only these tables, it is served by the table cache, not reaching the database server. If a query accesses both cached tables and non-cached tables, it is rewritten so that only the part of the query on the non-cached tables is sent to the database server. For example, to implement *getMostRecentOrder()*, five tables will be accessed: CUSTOMER, ADDRESS, COUNTRY, ORDERS, and CC_XATCS. The corresponding query is rewritten so that only the parts that access ORDERS and CC_XATCS are sent to the database server.

In the experiments of this study, the table cache serves faked results to the queries or partial queries that involve data from the cached tables. Faking results will not affect the workload to the database server since how the table cache responds to queries is transparent to the database server. Also, faking results at the table cache does not significantly affect the response time measured in this study, especially when the database server is heavily loaded. The reason is that the table cache is considered highly scalable compared with the database server because the table cache can be scaled up by replication like Web/application servers.

4.3.2 TPC-W Workload Configuration

Based on the TPC-W specifications, an average think-time of no less than 7 seconds and no more than 8 seconds for an EB performing Web interactions is required [41]. Think-time is defined as the time elapsed from the last byte received by an EB to

complete a Web interaction until the first byte sent by the EB to request the next Web interaction. A think-time of 7 seconds is used in this study.

In this study, 10,000 items and 8.64 million customers (3,000EBs) are used to build the TPC-W database, which results in the size of 17.7GB on disk, as illustrated in Table 4.1. Indexes are built on tables to speed up locating and sorting records. TPC-W specifications allow the use of indexes [41]. The indexes in this study are shown in Table 4.2.

Table 4.1: Database Size

Table Name	**Cardinality**	**Table Size**	**Index Size**	**Subtotal**
CUSTOMER	8,640,000	4,578MB	1,287MB	5,865MB
ADDRESS	17,280,000	2,291MB	3,278MB	5,569MB
ORDERS	7,776,000	672MB	511MB	1,183MB
ORDER_LINE	23,328,000	2,340MB	1,943MB	4,283MB
CC_XACTS	7,776,000	821MB	441MB	1,262MB
ITEM	10,000	6MB	4MB	10MB
AUTHOR	2,500	1,020KB	276KB	1,296KB
COUNTRY	92	8KB	24KB	32KB
Total		10.45GB	7.29GB	17.7GB

4.3.3 Buffer Pool Trace Collection and Description

During the running of the TPC-W workloads, the resulting logical page requests sent to the DBMS buffer pool were collected (*trace point* in Figure 4.1) using the tool described in the previous chapter. Each trace record in the trace file is composed of 6 fields: *client_ID*, *request_type*, *object_type*, *object_ID*, *page_number*, and *fix_mode* (used for fixes) or *modified_flag* (used for unfixes). Short descriptions of the fields are listed in Table 4.3.

The *fix_mode* is a lock technique used to control the concurrent read/write access to the requested pages in the buffer pool. If a page is fixed in the *shared* mode, it can be read by multiple users at the same time during the fix period. If a page is fixed in the *exclusive* mode, it can only be read/written by a single user during the fix period. A page is generally fixed in the *exclusive* mode just before a *write*

Table 4.2: TPC-W Database Indexes

Table Name	Index Field Name	Comments
CUSTOMER	C_ID	Unique ID per customer
	C_UNAME	Unique user name for customer
	C_ADDR_ID	Address ID of customer
ADDRESS	ADDR_ID	Unique address ID
	ADDR_ZIP	Zip code or postal code
	ADDR_CO_ID	Unique ID of country
ORDERS	O_ID	Unique ID per order
	O_C_ID	Customer ID of order
	O_DATE	Order date and time
ORDERS_LINE	OL_ID	Unique order line item ID
	OL_O_ID	Order ID of order line
	OL_I_ID	Unique item ID(I_ID)
CC_XACTS	CX_O_ID	Unique order ID(O_ID)
ITEM	I_ID	Unique ID of item
	I_TITLE	Title of item
	I_SUBJECT	Subject of book
	I_A_ID	Author ID of item
AUTHOR	A_ID	Unique author ID
	A_LNAME	Last name of author
COUNTRY	CO_ID	Unique country ID
	CO_NAME	Name of country

Table 4.3: Buffer Pool Trace Fields

Field Name	Description
client_ID	the ID of the client who sends the request
request_type	the type of the request, either *fix* or *unfix*
object_type	the type of the requested page, either *data* or *index*
object_ID	the ID of the requested *table* or *index*
page_number	the logical page number of the requested page
fix_mode (0 |1)	how to fix the requested page: *shared - 0* or *exclusive - 1*
modified_flag (0 |1)	whether the page has been modified: *clean - 0* or *dirty - 1*

operation, and the page is unfixed after the writing. This mechanism can prevent other users from reading stale data and avoid possible conflicts caused by multiple users trying to write a page at the same time.

4.3.4 Snapshot Description and Collection

In this study, a light-weight facility (*getSnapshots*) is developed using C++ upon the DB2 snapshot monitor API (*sqlmon() family*) to collect database snapshots. With a snapshot interval of one second, it is observed that many SQL statements complete within that time interval. This is not always the case, however, especially for the workloads that contain complex queries that are too long to complete within one second. Thus, the snapshots were dynamically resized by coalescing consecutive one-second raw snapshots until encompassing at least one statement completion. The consolidated snapshots are then normalized with respect to the number of SQL statements executed within a snapshot. Consequently, each normalized snapshot describes the characteristics of a single SQL statement.

```
@relation oatp-oltp-train

@attribute QueriesRatio real
@attribute PagesRead real
@attribute RowsSelected real
@attribute IndexRatio real
@attribute Sorts real
@attribute Logging real
@attribute MixType {OLAP, OLTP}

@data
0.939929, 21.6608, 4.15548, 0.213214, 0.116608 , 0.00706714, OLAP
0.931624, 24.094 , 4.66667, 0.18801 , 0.0811966, 0.0213675 , OLAP
0.961702, 24.2809, 4.64681, 0.203645, 0.144681 , 0.0170213 , OLAP
0.94375 ,  23.65 , 4.56562, 0.213795, 0.159375 , 0.015625  , OLAP
... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ...
... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ...
```

Figure 4.4: ARFF File

The snapshots were then converted to *ARFF (Attribute-Relation File Format)* files after they were collected and processed. ARFF is the file format for the decision tree model J4.8 that is used as the classification model in this study. The J4.8

program (*weka.classifiers.trees.J48*) from Weka is a Java implementation of the most recent version of C4.5 (C4.8) [50]. Weka is a collection of machine learning algorithms for data mining tasks. The algorithms are implemented in Java as an open source toolkit. The C4.5 algorithm is one of best-known and most widely-used decision tree models.

Figure 4.4 shows an example of an ARFF file that was the training snapshots of TPC-W Browsing profile after being labelled as OLAP. An ARFF file is an ASCII text file that includes a list of instances sharing a set of attributes. It has two distinct sections. The first section is the *Header* information, which is followed the *Data* information. The Header of an ARFF file contains the name of the relation, a list of the attributes (the columns in the data), and their types. The first 6 attributes use real number as their *datatype*. The last attribute is a nominal attribute which has two possible values: "OLAP" or "OLTP". In a target ARFF file, the last attribute is undecided, marked as "?".

4.4 Experiment Parameters

4.4.1 System Platforms

The database server machine used in our experiments is IBM eServer xSeries 255 (868541X), configured with 4 Intel Xeon MP 1.5GHz CPUs, 8GB RAM, 12 34.7GB IBM U320 hard disks attached to two IBM ServeRAID-4Lx Ultra160 SCSI controllers. All disks are 15,000RPM with an average seek time of about 3.6 ms. The operating system platform is Microsoft's Windows 2000 Server.

The database server is IBM DB2 8.1. The database connectivity is done through JDBC. The Java Virtual Machine is Java 2 Runtime Environment, Standard Edition 1.4.2. DB2 runs under its default settings for TPC-W in all the experiments in this study. Some of the default parameters are shown in Table 4.4.

Table 4.4: DB2 Default Settings

Parameter	Default Value	Description
-BUFFPAGE	25000	Buffer pool size (4KB)
-NUM_IOCLEANERS	5	Number of page cleaners
-APPLHEAPSZ	256	Application heap size (4KB)
-DFT_PREFETCH_SZ	0	Default prefetch size (4KB)
-DBHEAP	600	Database heap size (4KB)
-MON_HEAP_SZ	512	Database monitor heap size (4KB)
-STMTHEAP	2048	SQL statement heap size (4KB)
-STAT_HEAP_SZ	1096	Statistics heap size (4KB)
-LOGBUFSZ	128	Log buffer size (4KB)
-LOGFILSIZ	25000	Log file size (4KB)

4.4.2 TPC-W Running Parameters

Table 4.5 shows the TPC-W (*tpcw.rbe*) running parameters. The *tpcw.rbe* is designed as a multithreaded program, with a single thread devoted to emulate each browser. During the simulation, each thread generates a sequence of user sessions, and within each user session the thread generates a sequence of Web interactions.

Table 4.5: TPC-W (*tpcw.rbe*) Running Parameters

Parameter	Value	Description
-EB	100 – 2000	Number of emulated browsers
-PROFILE	B/S/O	Browsing(B), shopping(S), or ordering(O)
-MN	100/1000	Measured number of interactions
-MI	1800 (seconds)	Measured interval
-OUT	<filename>	File name for response time of interactions
-CACHE	no/query/table	No cache, query-result cache, or table-cache
-TIMEOUT	5/30/60	Timeout for query-result cache

Among all the parameters, the first two parameters (*-EB* and *-PROFILE*) must be present and one of *-MN* and *-MI* must be specified as well. They determine the running behaviors of the emulated browsers. The *-EB* parameter specifies the number of emulated browsers, and the *-PROFILE* parameter specifies one of TPC-W profiles, i.e. Browsing, Shopping, and Ordering. The number of interactions can be set by the number of interactions each emulated browser will perform (*-MN*) or

the time period each emulated browser will run (*-MI*). The response time file can be specified by *-OUT*. The cache type can be set as no cache, query-result cache, or table-cache by the parameter *-CACHE*. The *-TIMEOUT* value is only needed when the *-CACHE* is set to "query" (query-result cache). The TPC-W database is restored to its original state after each run. This is necessary because some interactions will change database tables.

The number of EBs are set from 100 to 2000 for the measurements of response time, number of references, disk I/Os. The number of EBs is set to 1800 for the examination of temporal locality and spatial locality, since it has been found the load on database server with 1800 EBs is the load condition of interest for these characteristics in this study. The number of interactions that each EB performs is set to 1000 when the number of EBs is less than 1000, and it is set to 100 when the number of EBs is equal to or more than 1000.

When the number of EBs is 1800 and the number of interactions *-MN* is set to 100, each run takes about 20 minutes to 50 minutes to finish. The Ordering workload takes the longest time and the Browsing workload takes the shortest. Three different timeouts are used: 5 seconds, 30 seconds, and 60 seconds to investigate the performance of query-result cache. The timeout of 30 seconds is used as the default value when studying the workload characteristics at the database server. For the classification of TPC-W profiles, the number of EBs is set to 1800 and the time period each emulated browser runs (*-MI*) is set to 1800 seconds. In this study, multiple runs (5 to 10) are performed to establish the degree of variance in the results. The variance is less than 3% in all cases.

4.4.3 Buffer Pool Simulator and Classifier Parameters

Table 4.6 shows the parameters of buffer pool simulator *bps*. For a trace, the first 30% (*-WARMUP 0.3*) is used as the warmup period. The range of the buffer pool size is set from 5000 to 80000 pages. LRU is used when studying the temporal locality of the workloads under caching. Several buffer pool replacement algorithms

are evaluated using the buffer pool simulator. They include LRU, LIRS, ARC, Random. The Random algorithm is used as the worst case scenario.

Table 4.6: Buffer Pool Simulator (*bps*) Parameters

Parameter	**Value**	**Description**
-FILE	<Trace file name>	Trace file name
-ALGR	LRU/ARC/LIRS/Random	Buffer pool repl. algr.
-SIZE	5000 – 80000	Buffer pool size
-WARMUP	0.3	Warmup proportion

Table 4.7 shows the parameters of the classifier J4.8. In the training step, the classifier first loads a training file with the parameter -T and saves the model by specifying the parameter -M. In the classification step, the classifier reads in the saved model (-L) and the target file (-C). It predicts the column of 7 (-P 7) for the target file. The classifier outputs either "OLAP" or "OLTP" for the target cases in the file. In this study, the percentage of "OLTP" objects predicted by the J48 classifier is used to reflect whether a dynamic caching mechanism can shift the TPC-W workload profiles. The CDFs of the snapshots attributes are also used to reflect how the dynamic caching mechanisms change the attributes' distributions.

Table 4.7: Profile Classifier (*weka.classifiers.trees.J48*) Parameters

	Parameter	**Description**
Training	-T <trainFilename>	Training file name
	-M <modelFilename>	To save the model file name
Classifying	-C <targetFilename>	Target classification file name
	-P <columnNumber>	The column to predict (7)
	-L <modelFilename>	Load model file

Chapter 5

Workload Characterization

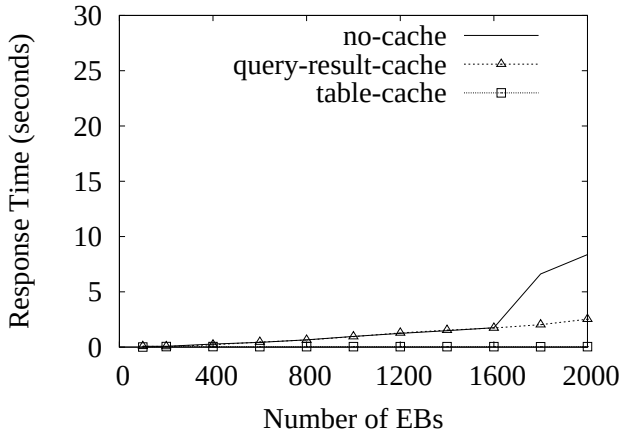
This chapter presents the TPC-W workload characteristics under various dynamic caching mechanisms. First, general characteristics are given. These characteristics include response time, CPU utilization of database server, and query-result cache hit ratio. Then, the caching effects on the following reference characteristics are given: number of references, working set size, number of physical I/Os, temporal locality and spatial locality. The performance of various buffer pool replacement algorithms under caching is analyzed at the end of this chapter.

5.1 Response Time

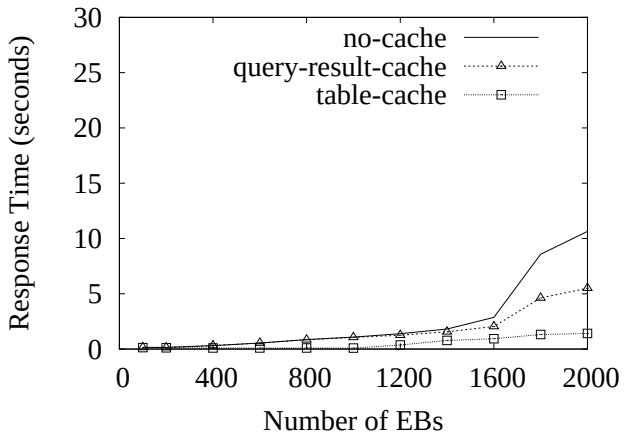
Figure 5.1 presents the average response time as a function of the number of EBs. In the figure, not using any dynamic cache is labeled as *no-cache*. The no-cache configuration is used to gauge the load on the database server.

For the *Browsing* and the *Shopping* workloads, the figure shows that the no-cache system is heavily loaded when the number of EBs is greater than $1,600$, as indicated by the rapid increase in response time. The corresponding number for the *Ordering* workload is 300.

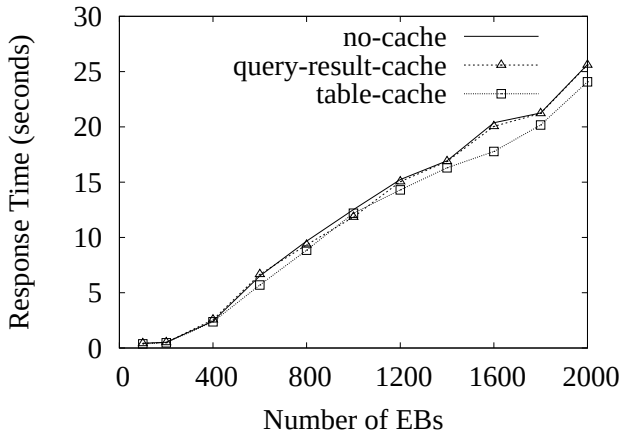
Figure 5.1 shows that, in general, when the system is heavily loaded, dynamic cache substantially reduces the database server response time. Under light load, query-cache has little effect on reducing the database server response time, because a cached query result often expires before the identical subsequent query is received



(a) Browsing



(b) Shopping



(c) Ordering

Figure 5.1: Response Time

by the cache. The figure also shows that using dynamic cache has the largest benefit in the *Browsing* workload and the smallest benefit in the *Ordering* workload. This is because most queries in the *Browsing* workload are cache-able, while the opposite is true for the *Ordering* workload.

In this study, the load with 1,800 EBs is chosen as the load of interest to investigate how dynamic caching mechanisms change database workloads. The reason is twofold. First, the focus of this study is the performance of the system when it is heavily loaded. Secondly, the average response time for the *Browsing* workload and the *Shopping* workload must not exceed 5 seconds [41]. Figure 5.1 shows that the response time of query-result cache and table cache is less than 5 seconds with the load of 1,800 EBs, although the response time of no cache at this load is 6.6 seconds and 8.5 seconds for the *Browsing* workload and the *Shopping* workload respectively. The much longer response time of the *Ordering* workload at the load of EB 1,800 is acceptable since the *Ordering* workload includes more CPU-intensive interactions (e.g. *Buy Confirm*) than the *Browsing* workload and the *Shopping* workload [41].

5.2 Cache Hit Ratio

For query-result cache, getting a high cache hit ratio is crucial. Increasing the timeout threshold for query-result cache can get a higher hit ratio and better response time. Table 5.1 presents hit ratios of the query-result cache and the response times of *Shopping* workload under three timeout threshold values: 5 seconds, 30 seconds, and 60 seconds. Using a longer timeout threshold allows the query-result cache to absorb more queries, thus increasing hit ratios and reducing response times. These results suggest that when the database server is heavily loaded, it may be beneficial for the system to temporarily use a larger timeout threshold at the cost of more obsolete results.

The timeout threshold of 30 seconds is used for the study of how query-result cache affect database workloads in this thesis. It is a standard threshold specified by the TPC-W specifications [41]. Highly frequent database updates prohibit caching a

Table 5.1: Timeout Threshold (1,800 EBs)

Timeout	Cache Hit Ratio	Response Time
5 seconds	15.2%	6.7 seconds
30 seconds	40.7%	4.6 seconds
60 seconds	49.1%	2.7 seconds

query result for a long time, so that the database updates can be reflected accordingly at the cache.

The size of query-result cache is also an important performance factor. In the experiments of this study, the size of each query result ranges from 20 bytes (*a customer name*) to 1400 bytes (*result from a function titleSearch()*). The size of the cache is set as infinite for initial testing. It is found that the size of all the cached results is always less than 20MB, since every cached result expires after 30 seconds. The size of query-result cache is then set to 20MB for the evaluation of caching effects on database server workloads.

5.3 CPU Utilization

Table 5.2 shows the reductions in CPU utilization of DBMS when query-result-cache and table-cache are employed. The use of dynamic cache considerably reduces the CPU utilization. This is because a large number of queries from the workloads are cache-able (for query-result-cache) or they only access the cached tables (for table-cache). Many queries are then filtered out by cache, not demanding any computing resource at the database server.

Table 5.2: Reductions in the CPU Utilization of DBMS

Dynamic Cache	Browsing	Shopping	Ordering
Query-result-cache	76.8%	20.1%	18.7%
Table-cache	89.6%	55.1%	33.9%

The table also shows that the reductions in CPU utilization decrease from the *Browsing* workload, to the *Shopping* workload and the *Ordering* workload. The most

reduction, almost 90%, happens on the *Browsing* workload under table-cache, and the least reduction of about 19% happens on the *Ordering* workload under query-result-cache. The reason is that the *Browsing* workload contains more *browsing* queries that can be served by cache. The *Ordering* workload includes the most *ordering* interactions (e.g. *Shopping Cart*) that generate un-cachable queries accessing highly frequent updated table.

It is also clearly shown in the table that the table-cache is more effective than the query-result-cache in reducing the CPU utilization. The table-cache can filter out more queries than query-result-cache for 3 reasons. First, it can serve some queries that the corresponding results cannot be cached, for example the query of *getStock()* from the *ITEM* table. Second, the table-cache does not have the first-miss situation that a new query cannot be served by the query-result-cache. Thirdly, the queries in the query-result-cache expire after 30 seconds, but this is not an issue for the table-cache.

5.4 Reference Characteristics

Table 5.3 summarizes the duration of the traces and the number of page requests for the sample workload when using 1,800 EBs. It shows that the duration of traces increases from the *Browsing* workload to the *Ordering* workload for all the three cache conditions: no-cache, query-result-cache, and table-cache. This is because the *browse* interactions are generally less CPU-intensive than the *order* interactions. The table also shows that the duration of traces for each workload always decreases from no-cache to table-cache. This implies that caching not only decreases response time (as shown in Figure 5.1) but also the total run time of a workload.

The table shows that the *Browsing* workload has the most number of requests among the three workloads when there is no cache, although each workload is configured to perform the same number of interactions. This is because the *Browsing* workload has more *browse* interactions, for example *Best Seller*, which usually request more pages than *order* interactions. It is also interesting to see that the *Brows-*

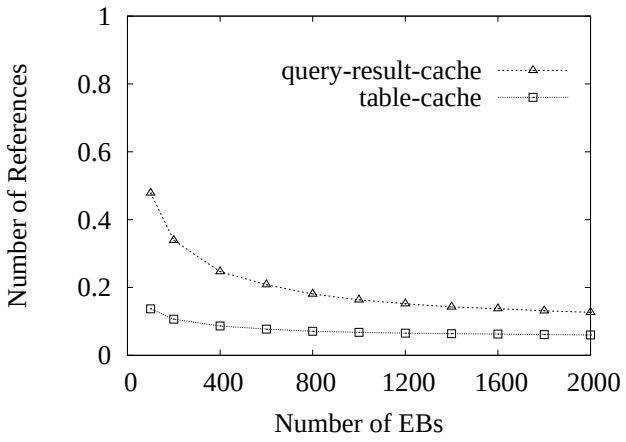
Table 5.3: Buffer Pool Trace Characteristics

Workload	Parameters	No-cache	Query-result-cache	Table-cache
Browsing	Duration (minutes)	22.2	20.3	16.1
	Number of Requests	12.4M	1.6M	0.76M
Shopping	Duration (minutes)	28.3	22.0	20.3
	Number of Requests	10.7M	3.7M	2.6M
Ordering	Duration (minutes)	51.3	50.0	48.0
	Number of Requests	9.7M	7.8M	6.1M

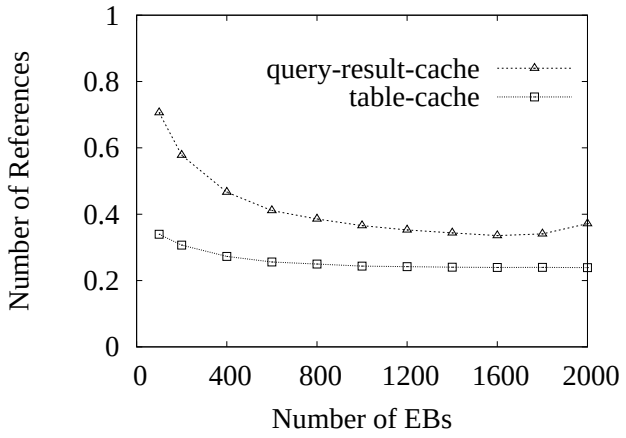
ing workload shows the fewest requests under query-result-cache and table-cache. The reason is that many of the queries that are generated by *browse* interactions are served by cache, and those queries could request a large number of pages if there is no cache.

The table also shows that table-cache is better than query-result-cache at reducing both the running time of a workload and the number of requests. Figure 5.2 shows how much table-cache and query-result-cache can reduce the number of requests in more detail. In this figure, the number of database page requests is specifically called the number of references. Loads of EBs from 100 to $2,000$ are used. The number of references is normalized to that without using dynamic cache (no-cache).

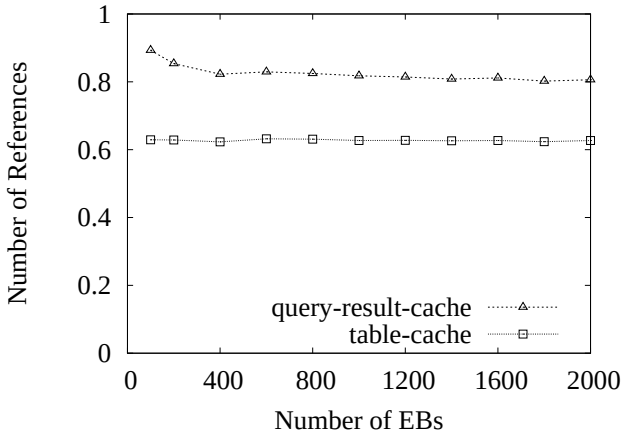
This figure shows that using dynamic cache can considerably reduce the number of database page references, especially for the *Browsing* workload and the *Shopping* workload. In this figure, from the *Browsing* workload (Figure 5.2(a)) to the *Ordering* workload (Figure 5.2(c)), the number of references increases for both the query-result-cache and the table-cache. This is also because that most of the queries from the *browse* interactions are served by the cache but those from the *order* interactions are forwarded to the database server. This figure also explains why the query-result-cache does not reduce the response time much for the loads of EBs from 100 to $1,600$, as shown in Figure 5.1, but it still reduces a considerable amount of CPU utilization of the database server, as shown in Table 5.2. It this figure, it is also clearly shown that table-cache always has a greater impact on reducing the number of references than the query-result-cache.



(a) Browsing



(b) Shopping



(c) Ordering

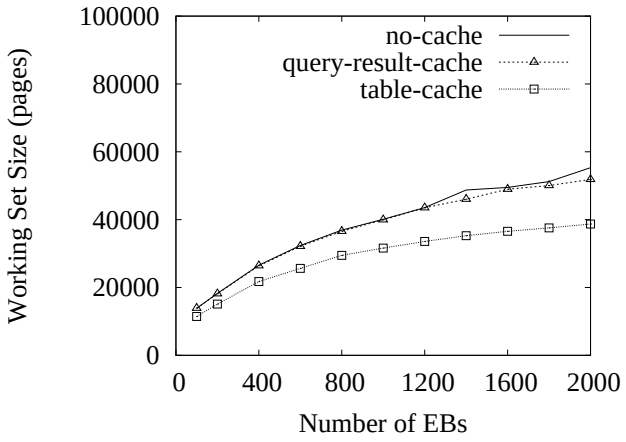
Figure 5.2: Database Page References

5.4.1 Working Set Size

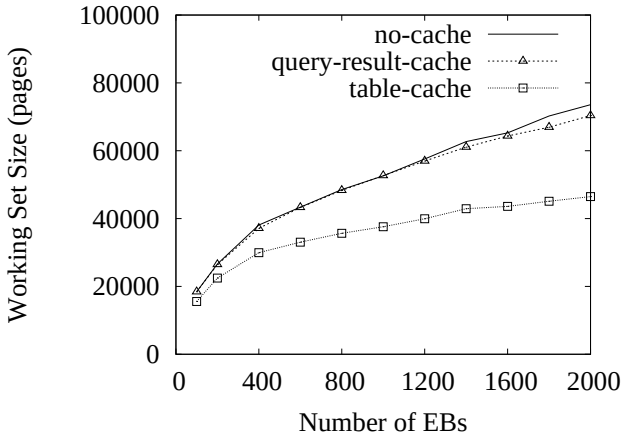
Figure 5.3 presents the number of distinct pages (i.e., the *working set size*) referenced by the three workloads with different numbers of EBs. The figure shows that using query-result-cache does not reduce the working set size to any substantial degree, while using table-cache can substantially reduce the working set size. This is because query-result-cache only reduces the frequency at which the cache-able queries are sent to the DBMS, while table-cache reduces the number of distinct queries reaching the DBMS. The table-cache removes all the distinct page accesses on the cached tables by serving the corresponding queries in all the workloads.

The figure also shows that query-result-cache slightly reduces the working set size for the *Browsing* workload and the *Shopping* workload when the number of EBs is more than $1,200$. This indicates that query-result-cache provides stale cached results to some queries which will access new distinct pages if these queries are sent to the database server. If there is a large number of users, a new data page can be added at the database server by *buying* interactions from the users, and a cached query result from the old pages can still be valid during this period. When a subsequent identical query arrives, the query-result-cache serves the old result instead of sending the query to the database server to access the updated table. This implies that a constant timeout value for query-result-cache can cause inconsistent data between the cache and the database server when there is a large number of users.

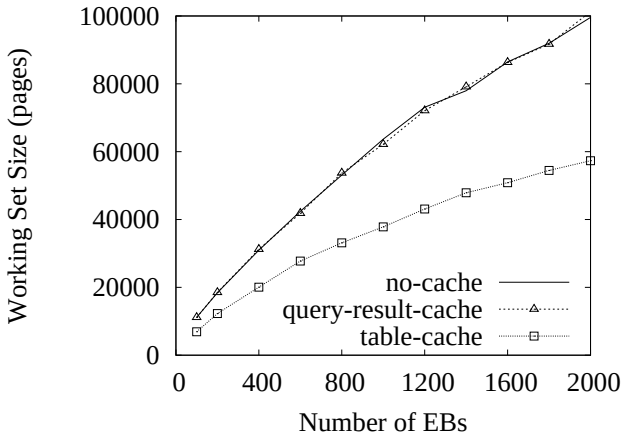
In this figure, the working set size increases from the *Browsing* workload (Figure 5.3(a)) to the *Ordering* workload (Figure 5.3(a)). This is because there are more *order* interactions in the *Ordering* workload, and the *order* interactions can generate new pages on tables at the database server which are not cached, for example the *ORDER_LINE* table. Also, the *Ordering* workload includes fewer *browse* interactions; this is why the query-result-cache does not reduce the working set size for the *Ordering* workload even there are a large number of users. A cached query result often expires before the next identical query arrives.



(a) Browsing



(b) Shopping



(c) Ordering

Figure 5.3: Working Set Size

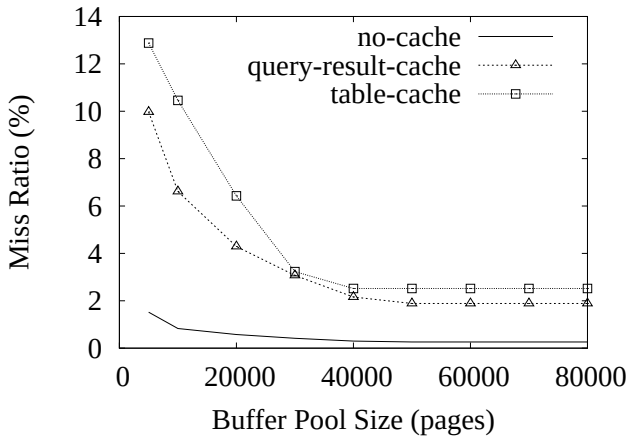
5.4.2 Temporal Locality

Temporal locality describes the tendency that a page will be referenced in the near future again if it has been referenced recently. The buffer pool miss ratio of LRU under different buffer pool sizes shows the temporal locality of the references.

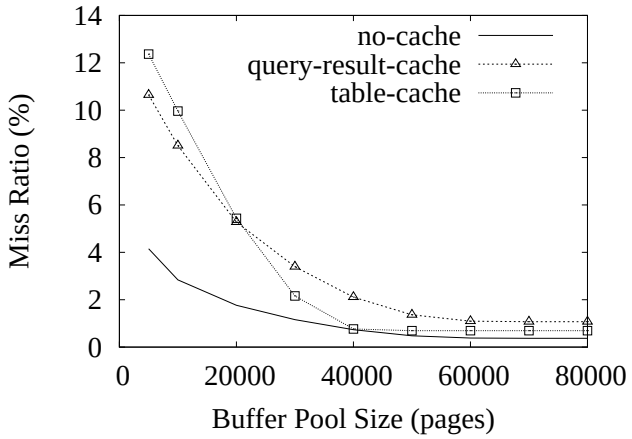
Figure 5.4 presents the overall buffer pool miss ratio as a function of the buffer pool size when using the query-result-cache and the table-cache. In Figure 5.4(a) and 5.4(b), the buffer pool has the lowest miss ratio when dynamic cache is not used. This is because that the database page references under no-cache configuration are composed of a large number of duplicated requests (comparing Figure 5.2 and Figure 5.3). The duplicated page requests directly contribute to the high hit ratio for the no-cache configuration. For the query-result-cache and the table-cache, some identical page requests are filtered out and are not received at the database server.

As shown in Figure 5.4 from top to bottom, the miss ratio for the no-cache systems increases as the proportion of cache-able queries in the workload decreases. This suggests that the buffer pool references generated by the cache-able queries have very good temporal locality. Most of the queries from *browse* interactions are cache-able, and most of the queries from *order* interactions are not cache-able. The *Browsing* workload has the largest proportion of *browse* interactions cache-able queries, while the *Ordering* workload has the fewest proportion of cache-able queries. In other words, these queries mainly consume CPU resources, but cause few disk I/Os.

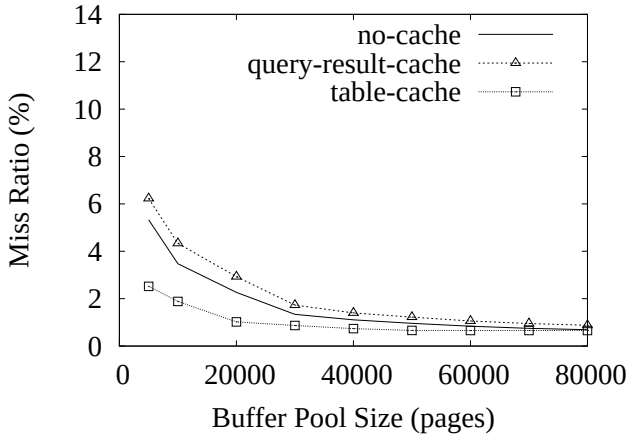
Since the query-result-cache filters out some of the cache-able queries, it always causes higher miss ratios than not using dynamic cache for all three workloads. Increased miss ratios are consistent with previous findings of the cache-filtering effect in multi-layer file servers [49] and Web proxies [16]. Figure 5.4 also shows that from the top figure to the bottom one, the gap between query-result-cache miss ratio and no-cache miss ratio for the same buffer pool size decreases. This is because from the *Browsing* workload to *Ordering* workload, the number of cache-able queries decreases.



(a) Browsing



(b) Shopping



(c) Ordering

Figure 5.4: Buffer Pool Overall Miss Ratio (LRU)

In Figure 5.4, the miss ratio of database page references with table-cache decreases (i.e., towards good temporal locality) from top to bottom. In Figure 5.4(c), the miss ratio with the table-cache is lower than that with no-cache. This is because the miss ratio is affected by two factors: the cache filtering effect and the working set size. The table-cache filters out cache-able queries, which decreases the temporal locality of references (i.e., towards increasing the miss ratio). On the other hand, as shown in Figure 5.3, the table-cache has a smaller working set size than that of no-cache, providing a lower miss ratio given the same buffer pool size. This is also the reason that the miss ratio of the *Shopping* workload with the table-cache, as shown in Figure 5.4(b), becomes lower than that with the query-result-cache when there are more than $20,000$ pages in the buffer pool.

In the *Browsing* workload, since many queries are filtered by a cache, the cache filtering effect substantially alters the workload seen by the server, giving higher miss ratios than that of no-cache. In the *Ordering* workload, since there are very few cache-able queries, the cache filtering effect almost vanishes. The miss ratios for all the workloads flatten out when the buffer pool size is larger than $60,000$ pages, which is about the same size of the working set size for *Browsing* workload and *Ordering* workload under the no-cache configuration and the query-result-cache (as shown in Figure 5.3).

Figure 5.5 shows the *write* buffer pool miss ratio with LRU algorithm. Among all the page references, *read* requests account for a high percentage of the total requests in all the traces, *write* requests occupy a relatively small proportion (between 3.4% and 8.1%). Comparing Figure 5.4 with Figure 5.5, it can be seen that the *write* miss ratios are much higher than the corresponding overall miss ratios for all the workloads under all the caching conditions: no-cache, query-result-cache, and table-cache. This indicates that the *write* requests do not have good temporal locality in comparison with the *read* requests. The difference of miss ratios in Figure 5.4 and Figure 5.5 also indicates the degree to which read operations dominate the overall request pattern. The difference between the overall miss ratio and the *write* miss ratio suggests that a system using separate buffer for *write* requests can obtain

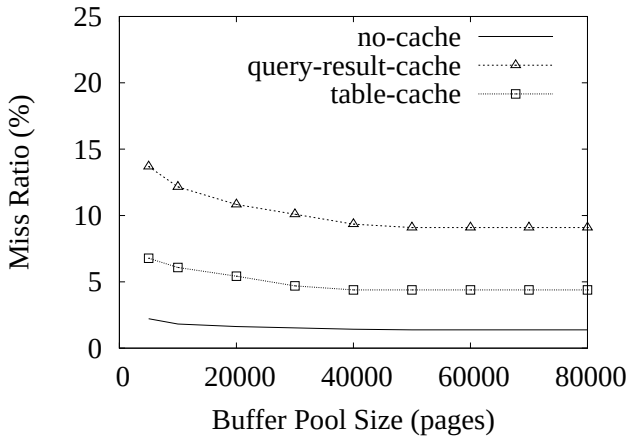
better performance.

5.4.3 Physical I/O Operations

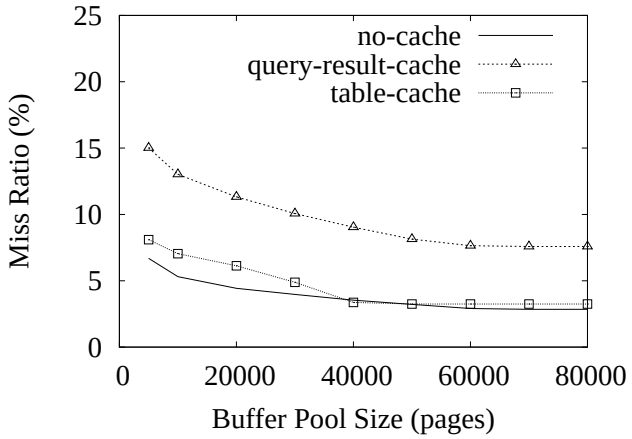
Figure 5.6 presents the number of disk I/Os for the three workloads under no-cache, query-result-cache, and table-cache. The LRU algorithm is used as the replacement algorithm at the database buffer pool. The figure shows that the number of disk I/Os for the no-cache configuration increases from the *Browsing* workload to the *Ordering* workload. This is because the temporal locality of the references decreases from the *Browsing* workload to the *Ordering* workload, as shown in Figure 5.4.

In this figure, the query-result-cache generates a similar number of disk I/Os to that in no-cache, unless the buffer pool is too small ($< 10,000$ pages). Although compared with the no-cache, the query-result-cache can eliminate a large proportion of the database references, Figure 5.6 reveals that it cannot reduce the disk I/Os at the database server, especially for the *Browsing* workload and the *Shopping* workload (shown in Figure 5.2). This implies that the database page references that are reduced by the query-result-cache cause buffer pool hits in the no-cache environment. It also implies that the cache-able queries have a relatively small working set compared to the buffer pool size. If the working set of cache-able queries cannot fit in the buffer pool, these queries will generate disk I/Os in the no-cache configuration. In that case, using the query-result-cache will reduce the number of disk I/Os (as illustrated by the left-most points of the query-result-cache line in the figure).

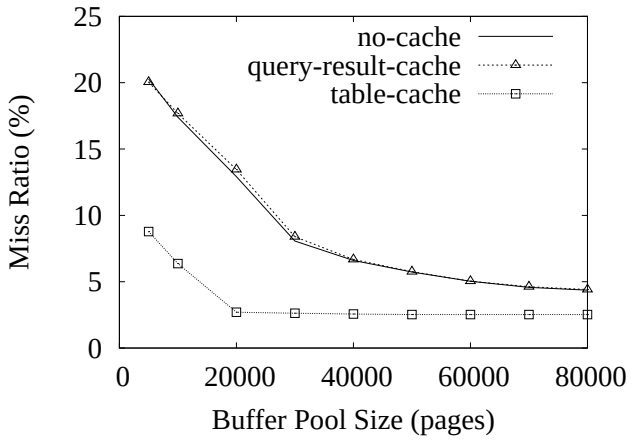
Figure 5.6 shows that the table-cache reduces the number of disk I/Os by a considerable amount when the buffer pool size is less than $60,000$ pages, especially for the *Shopping* workload and the *Ordering* workload. Also, at a given buffer pool size, more disk I/Os are removed from the *Browsing* (Figure 5.6(a)) workload to the *Ordering* workload (Figure 5.6(c)). For example, at the buffer pool size of $30,000$ pages, the table-cache removes 14 thousands disk I/Os for the *Browsing* workload, while it removes 34 thousands for the *Shopping* workload and 39 thousands for the *Ordering* workload.



(a) Browsing

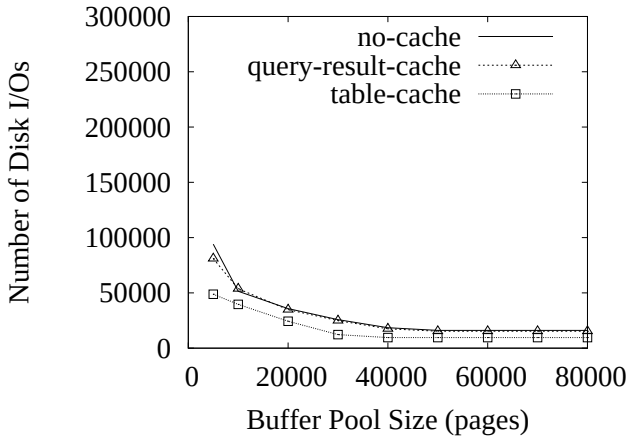


(b) Shopping

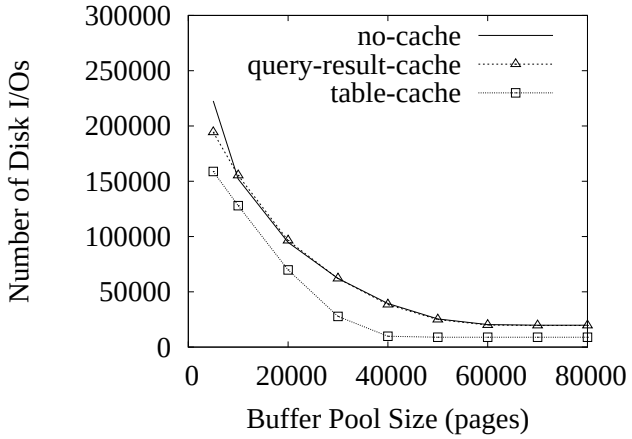


(c) Ordering

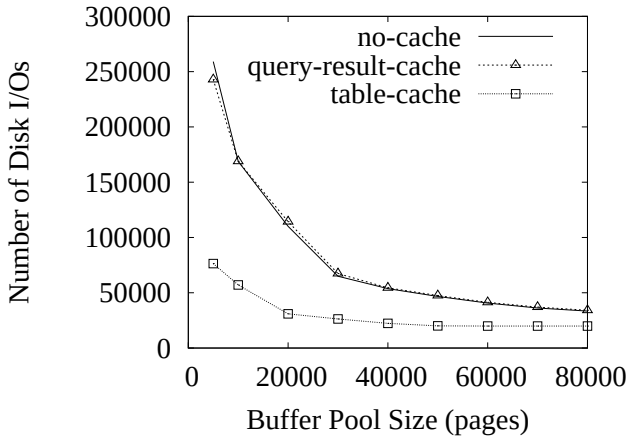
Figure 5.5: Buffer Pool Write Miss Ratio (LRU)



(a) Browsing



(b) Shopping



(c) Ordering

Figure 5.6: Number of Disk I/Os

The fact of more disk I/Os are removed from the *Ordering* workload than those from the *Browsing* workload seems contradictory to the TPC-W specifications [41]. According to the specifications, the number of queries on the cached tables, i.e., read-only queries, decreases from the *Browsing* workload to the *Ordering* workload. However, there is another important contributing factor to the number of removed disk I/Os. It is the temporal locality of the references. The temporal locality of the references increases from the *Browsing* workload to the *Ordering* workload under the table-cache, as shown in Figure 5.4. More disk I/Os can be removed as the temporal locality of references increases since less buffer misses are achieved for a given buffer pool size.

5.4.4 Spatial Locality

Spatial locality describes how likely a page is referenced if nearby pages are referenced. Good spatial locality in database page references implies that the buffer pool can benefit from use of a large page size and/or a prefetch policy. Spatial locality can be measured using *run length* [20]. A run length of n pages means that these n pages are accessed sequentially. Figure 5.7 presents the cumulative distribution of database page run lengths. For example, the point A in Figure 5.7(a) indicates that 40% of the references occur in sequential runs of fewer than 35 references.

Figure 5.7(a) shows that for no-cache running the *browsing* workload, about 50% of the references belong to sequential runs of fewer than 70 references, implying relatively good spatial locality. Figure 5.7(a) also shows that another 50% of the references belong to a sequential run of 244 references. A closer look at the trace found that this run is a sequential scan of the *author* table. Since the queries that invoke this sequential scan can be cached by the query-result-cache, a smaller proportion (20%) of references are involved in this run in the query-result-cache. This sequential run disappears in the run length distribution when using the table-cache, since the *author* table is cached by the table cache server. From the top to the bottom of Figure 5.7, the proportion of references belonging to this sequential

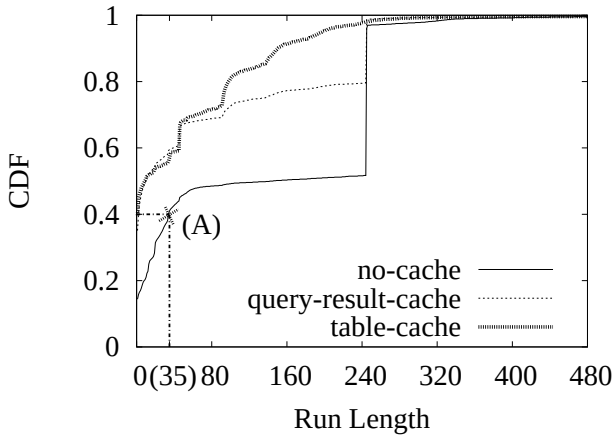
run decreases as the proportion of cache-able queries in the workload decreases. In Figure 5.7, most references occur in sequential runs of small lengths, implying that prefetch can affect the system performance.

5.5 Buffer Pool Replacement Algorithms

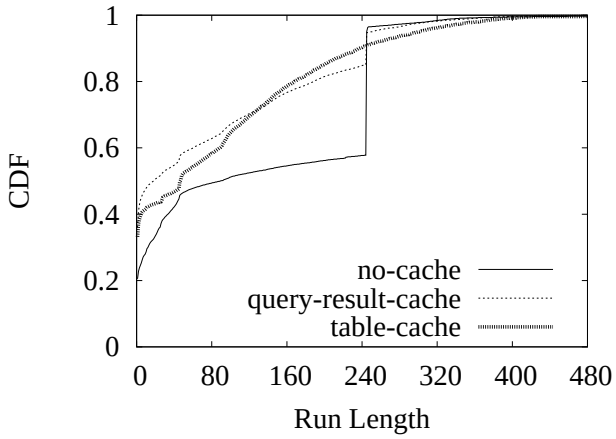
Figure 5.8 shows the buffer pool miss ratio with various replacement algorithms for the *Shopping* workload. The *Random* algorithm is used as the worst case scenario algorithm. The miss ratio of the *Random* algorithm is the same as those of LRU, ARC, and LIRS in Figure 5.8(a) when the buffer pool size is greater than $60,000$ pages. This indicates that the buffer pool can hold all the referenced pages when it is greater than $60,000$ pages.

Figure 5.8 shows that when the buffer pool size is small, ARC and LIRS perform better than LRU for the workloads under all the cache configurations: no-cache, query-result-cache, and table-cache. This is because both ARC and LIRS use the additional inter-reference information to keep records of all the references for selecting the next victim page. In Figure 5.8, ARC performs similarly to LIRS. This figure also illustrates that the temporal locality of the *Shopping* workload is not very sensitive to the buffer pool replacement algorithms.

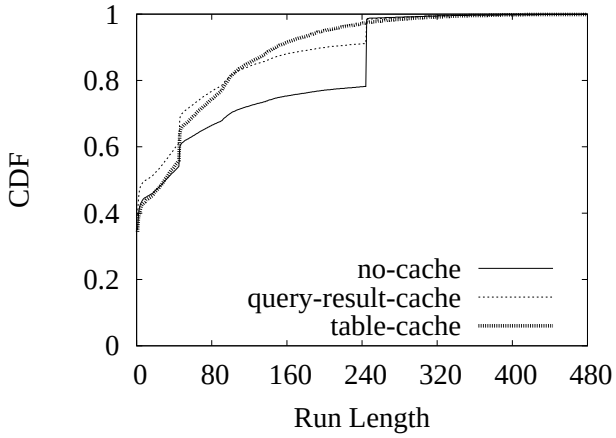
Similar results for the *Browsing* workload and the *Ordering* workload are shown in Figure 5.9 and Figure 5.10 respectively. Figure 5.9 and Figure 5.10 also show an interesting fact: there is no clear winner between LIRS and ARC for TPC-W workloads. For example, LIRS has much lower miss ratio for the buffer pool size of $20,000$ pages and $30,000$ pages for the *Browsing* workload with table-cache (as in Figure 5.9(c)), but ARC shows lower miss ratio when the buffer pool size is greater than $40,000$ pages for the *Ordering* workload with query-result-cache (as in Figure 5.10(b)).



(a) Browsing

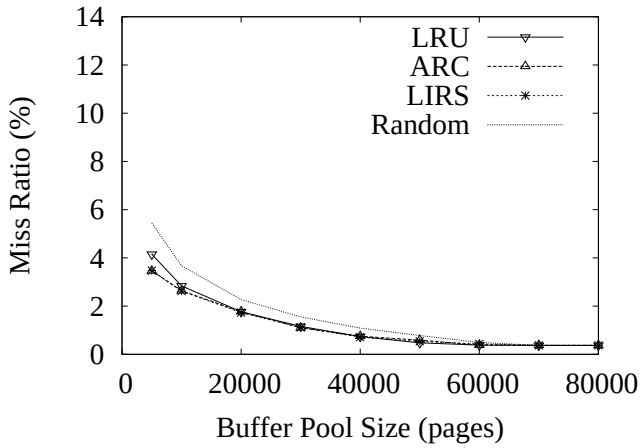


(b) Shopping

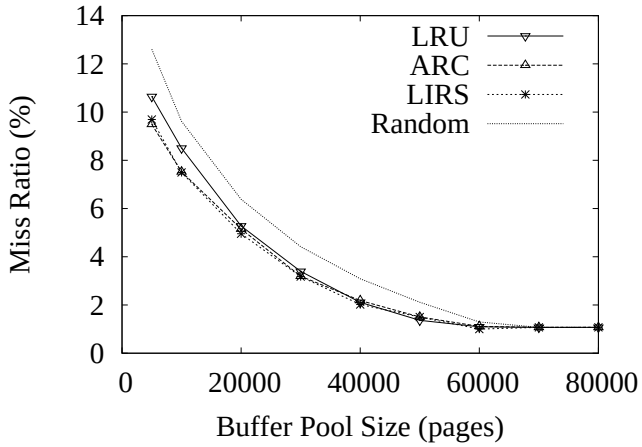


(c) Ordering

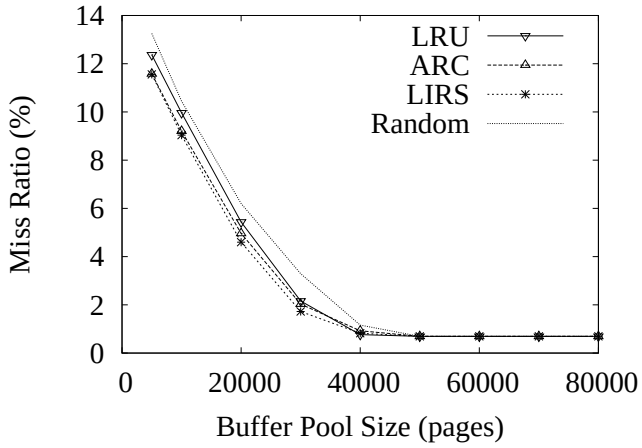
Figure 5.7: Run Length of Database Page References



(a) No-Cache

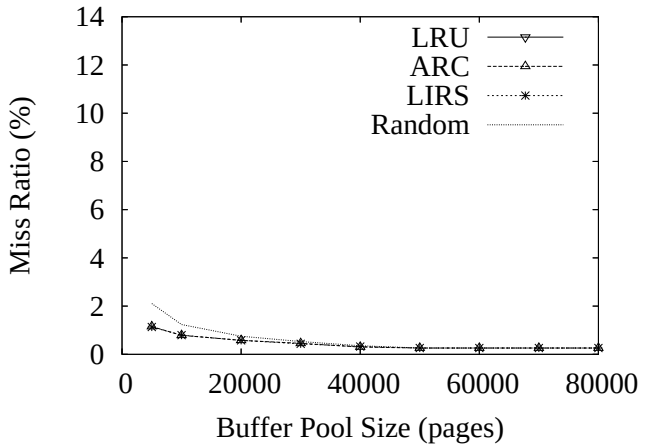


(b) Query-Result-Cache

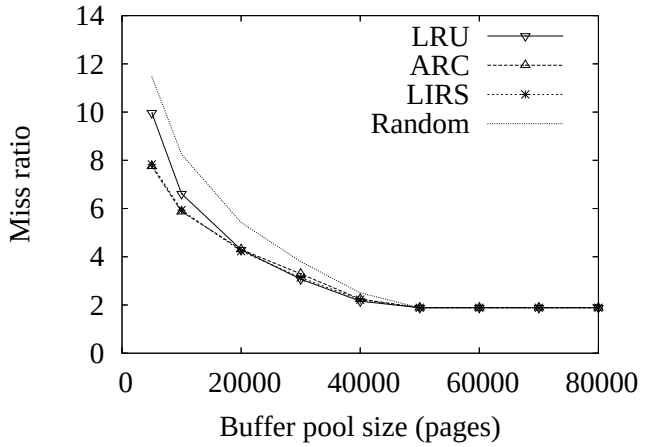


(c) Table-Cache

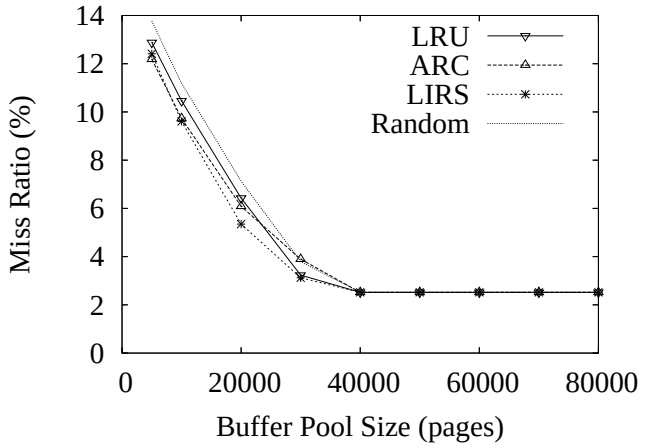
Figure 5.8: Buffer Pool Overall Miss Ratio (Shopping)



(a) No-Cache

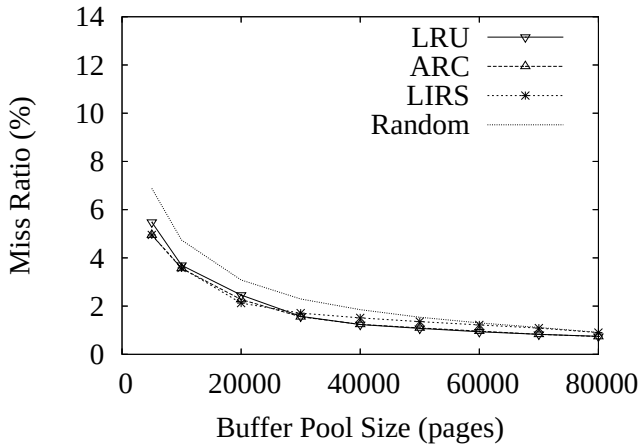


(b) Query-Result-Cache

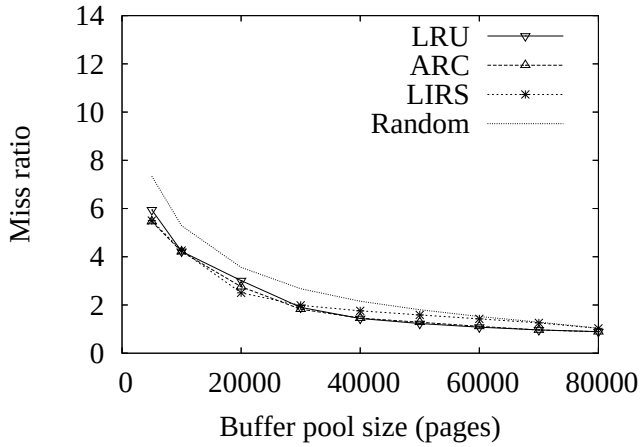


(c) Table-Cache

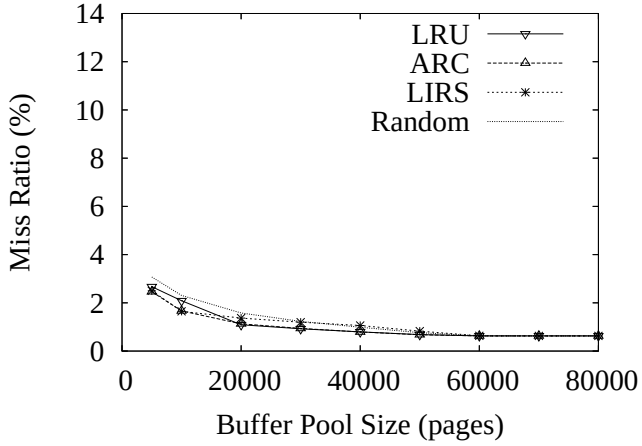
Figure 5.9: Buffer Pool Overall Miss Ratio (Browsing)



(a) No-Cache



(b) Query-Result-Cache



(c) Table-Cache

Figure 5.10: Buffer Pool Overall Miss Ratio (Ordering)

5.6 Summary

The experimental results in this chapter show that the dynamic caching mechanisms change the characteristics of database workloads. Using dynamic caching mechanisms can get good response time and considerably reduce CPU utilization at the database server when the database server is heavily loaded. The dynamic caching mechanisms also alter the reference characteristics of the database workloads. They can remove a large number of page references. They reduce the working set size, the number of disk I/Os, and the spatial locality of workloads as well, but with various scales. Also, they alter the temporal locality of the page references. Moreover, the results show that the table-cache is more effective than the query-result-cache at the caching effects. The experimental results in this chapter also show that the *write* miss ratios of the workload references are relatively high, and that there are minor performance differences among the examined replacement algorithms.

The experimental results in this chapter present several database performance indications. If a database server in an E-commerce system becomes heavily loaded, dynamic caching mechanisms can be deployed to reduce the load. Table cache can reduce the database server load more effectively than query result cache. Database buffer pool prefetching algorithms could moderately improve the system performance. The evaluated various buffer pool replacement algorithms cannot contribute much performance difference to the database server. A separate buffer pool for the *write* pages could be a good option to improve the system performance.

Chapter 6

Workload Classification

This chapter shows the experimental results on workload classification at the database server under dynamic caching mechanisms. First, the attributes of database snapshots with/without dynamic caching are examined in terms of their distributions. Then, the classification results predicted by the data mining tool are presented.

6.1 Attributes Without Caching

Database snapshots collected during the execution of TPC-W profiles are used as data objects for the classifier. A database snapshot includes important performance data of a database server. Each snapshot can be viewed as a vector whose dimensions are the attributes that summarize the activities in the system during the execution. There are 6 attributes in a snapshot object for the classifier: *Queries Proportion, Pages Read, Rows Selected, Number of Sorts, Ratio of Index,* and *Logging*. These attributes can reflect the behavioural differences between OLAP workloads and OLTP workloads [15].

Figure 6.1 shows the attribute values of the *Browsing*, *Shopping*, and *Ordering* profiles. All the values for each profile are normalized with respect to the *Browsing* profile. The figure shows that the snapshot attributes of the *Browsing* profile are quite different from those of the *Ordering* profile. The *Browsing* profile always gets higher values for Queries Proportion, Pages Read, Rows Selected, and Sorts, while the ordering profile has much higher values for Index Ratio and Logging. The

attributes of the *Shopping* profile always fall into the gap between the *Browsing* profile and the *Ordering* profile. This makes sense because the ratio of request types is between the *Browsing* and the *Ordering* profile by definition [41].

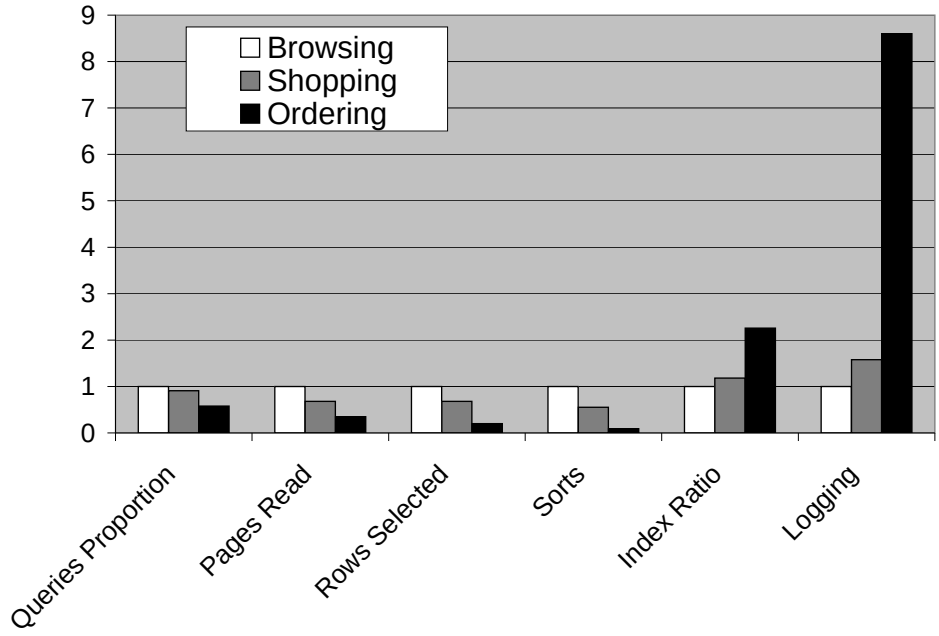
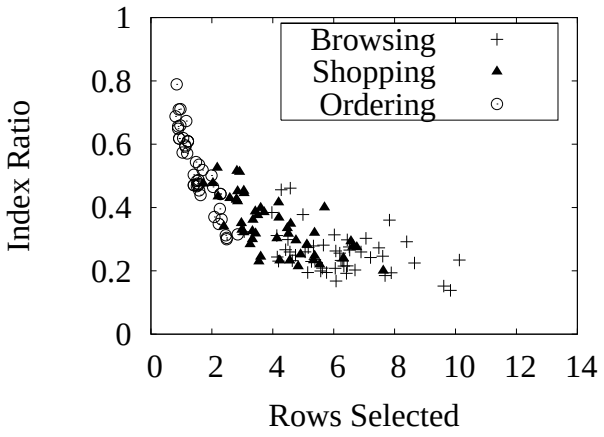


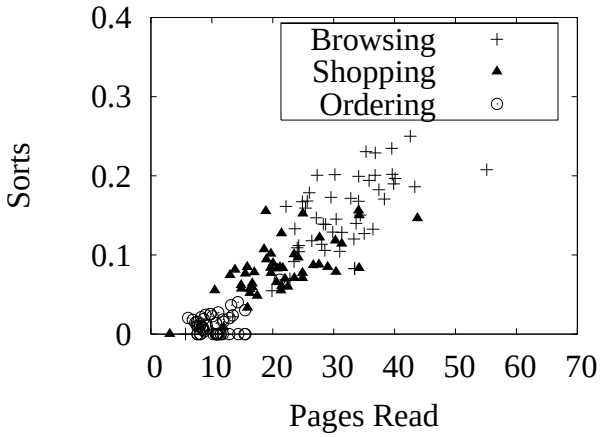
Figure 6.1: Snapshot Attributes

To further understand the attributes shown in Figure 6.1, the distributions of attribute values for the *Browsing* workload and the *Ordering* workload are also plotted in Figure 6.2 in three pairs, in which 50 random attribute values are selected. There are 12 other pairwise comparisons and they all show similar trends. The three pairs of *(Rows Selected, Index Ratio)*, *(Pages Read, Sorts)* and *(Queries Proportion, Logging)* in Figure 6.2 are selected randomly.

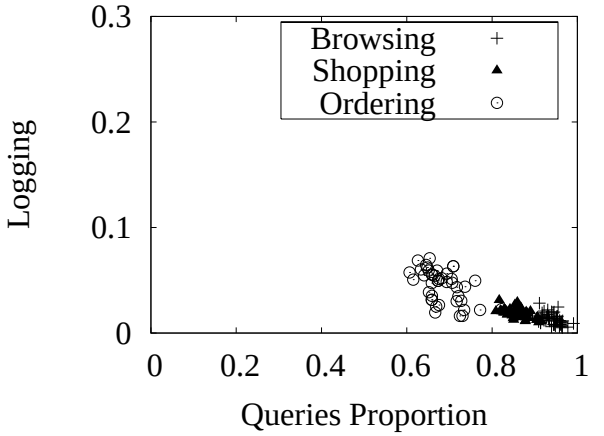
Figure 6.2 shows that the attributes of the *Browsing* profile and the *Ordering* profile span a relative large range within the distributions, but they are clearly in two distinct groups. Take the *Pages Read* as an example, as shown in Figure 6.2(b), it can range from 5 to 55 for the randomly selected 150 values. Most of the values for the *Browsing* profile are below 20, while most of the values for the *Ordering* profile are above 20. The distributions of the *Shopping* profile attributes mainly stay between those of the *Browsing* profile and the *Ordering* profile, although they



(a) Rows-Index



(b) Pages-Sorts



(c) Queries-Logging

Figure 6.2: Attributes for Browsing, Shopping, and Ordering without Caching

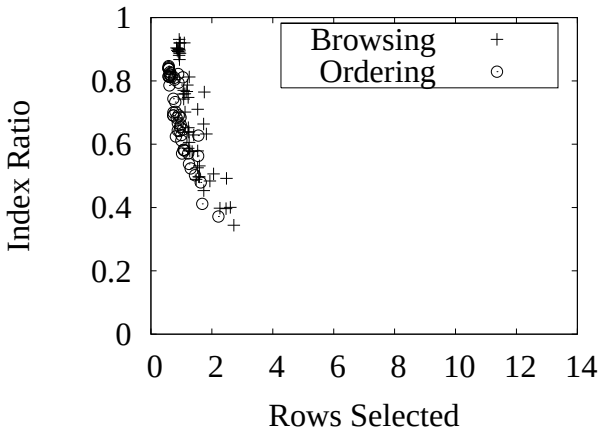
have some overlap with those of the other two profiles, which is also shown in Figure 6.2.

6.2 Attributes Under Caching

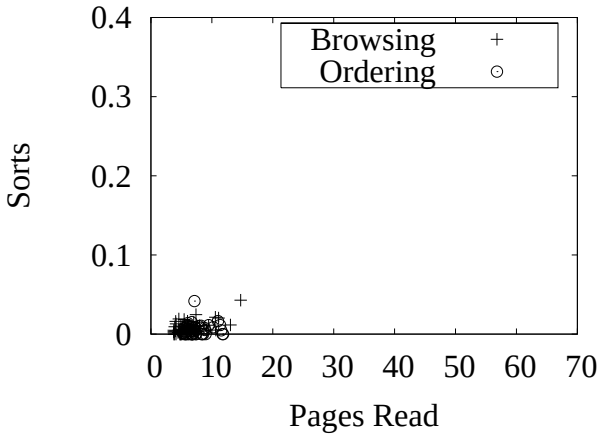
The distributions of attribute values of the TPC-W *Browsing* and *Ordering* profiles under query-result-cache and table-cache are shown in Figure 6.3 and Figure 6.4 respectively. These two figures show that query-result-cache and table-cache reduce the Queries Proportion, Pages Read, Rows Selected, and Sorts, while increase Index Ratio and Logging, which is towards the characteristics of OLTP workloads. In Figure 6.3 and Figure 6.4, it is also shown that the table-cache is more effective than the query-result-cache at changing the attributes although they alter the attributes to the same direction.

The distributions of most of the *Shopping* profile attributes under caching remain between those of the *Browsing* profile and the *Ordering* profile under caching. Detailed distributions are illustrated by the CDF of the attributes in the next 6 figures. The following 6 figures, from Figure 6.5 to Figure 6.10, further show the CDFs of the 6 attributes under query-result-cache and table-cache for all the three profiles. With regard to Pages Read and Rows Selected, shown in Figure 6.5 and Figure 6.6, both types of caching blur the distinction among the profiles. For *Browsing* and *Shopping* profiles, the Pages Read and the Rows Selected decrease dramatically under the query-result-cache and the table-cache. These two attributes for the *Ordering* profile also decrease, but to a lesser degree. Under the query-result-cache and the table-cache, the distributions of the Pages Read and Rows Selected for all the profiles shift to the direction of OLTP workload. Furthermore, table cache tends to be more effective at the shifting of the distributions.

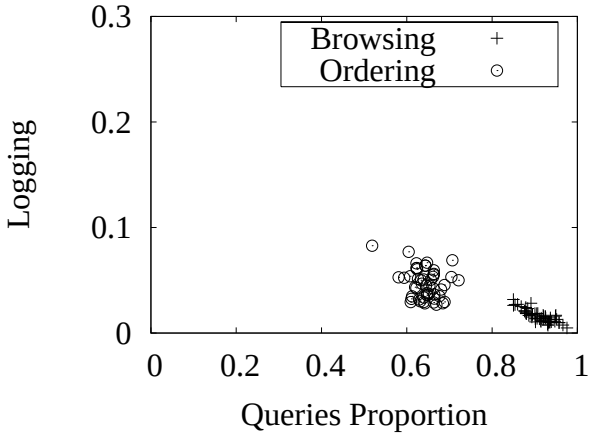
Figure 6.7 and Figure 6.8 also show similar results. The distributions of Queries Proportion shift to the left in a large degree under the table-cache, and so do the distributions of Sorts under the query-result-cache. The distributions of Queries Proportion under the query-result-cache (Figure 6.7(b)) barely shift to the left with



(a) Rows-Index

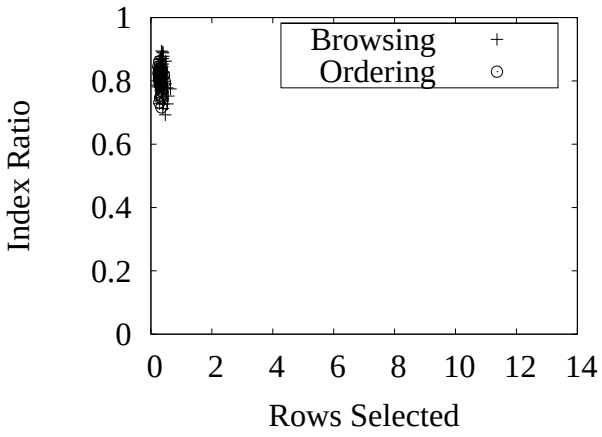


(b) Pages-Sorts

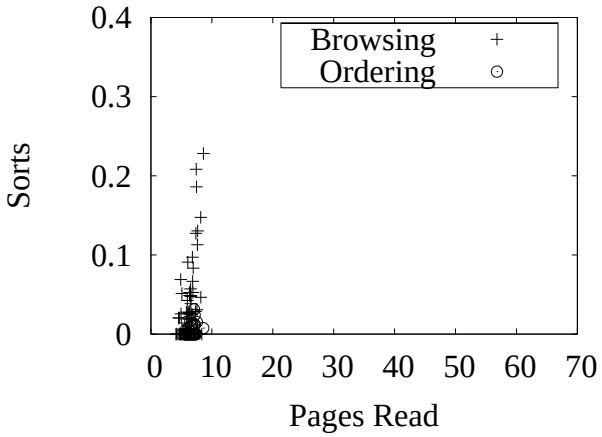


(c) Queries-Logging

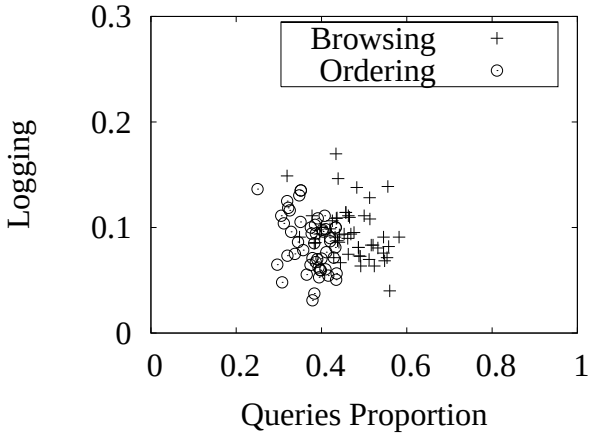
Figure 6.3: Attributes for Browsing and Ordering with Query-Result Cache



(a) Rows-Index



(b) Pages-Sorts



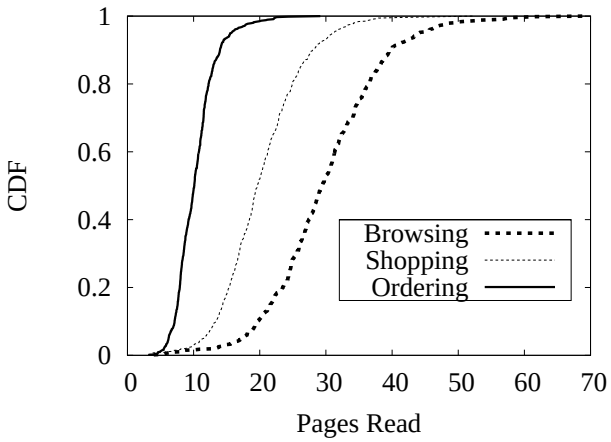
(c) Queries-Logging

Figure 6.4: Attributes for Browsing and Ordering with Table Cache

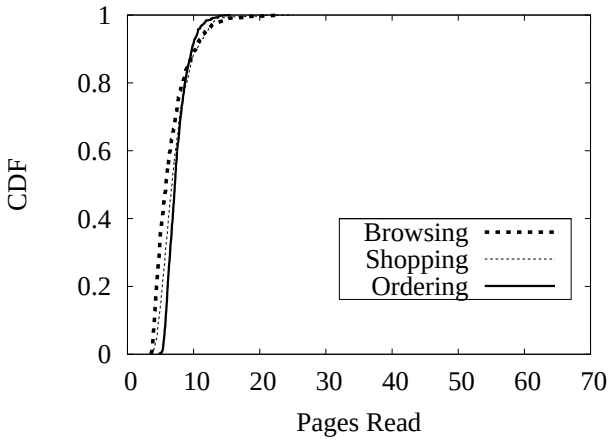
a large scale. This is interesting since the query-result-cache is supposed to filter out only *SELECT* statements, not *UPDATE/INSERT/DELETE* statements. This happens because more *UPDATE/INSERT/DELETE* statements from *buying* interactions arrive at the database server during a certain time period since the query-result-cache improves the response time. It is also interesting that under the table-cache, the distribution of Queries Proportion for the *Browsing* profile (Figure 6.7(c)) falls between that of the *Shopping* profile and the *Ordering* profile. The reason is that the *Browsing* profile produces more *SELECT* statements than the *Shopping* profile does. Figure 6.7(b) and Figure 6.7(c) reveal that the value of Queries Proportion with caching does not only depend on which profile the database server is processing, but also how the cache filters out the queries in the profile.

Figure 6.9 and Figure 6.10 show that the query-result-cache and the table-cache push the distributions of Index Ratio and Logging Size for all the profiles to the right with various scales. The two figures show that the distributions of the Index Ratio and Logging Size for the *Browsing* profile under the no-cache configuration, shown in Figure 6.9(a) and Figure 6.10(a) are very close to those of the *Shopping* profile. However, it is interesting to see that the distribution of Index Ratio for the *Browsing* profile under the table-cache (Figure 6.9(c)) is very close to that of the *Ordering* profile. The distribution of Logging Size for Browsing profile under the table-cache also shows a similar result, which is shown in Figure 6.10(c).

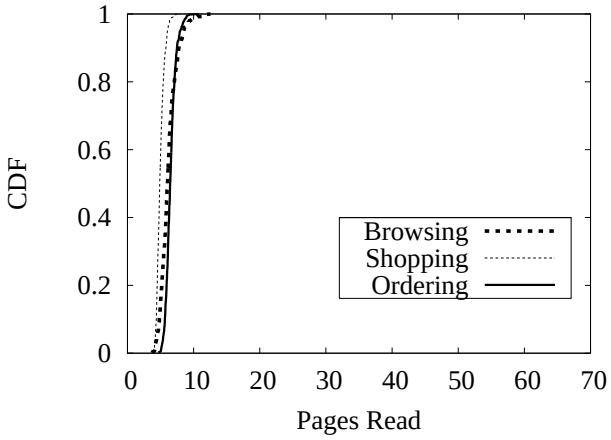
The 6 figures (from Figure 6.5 to Figure 6.10) show that the query-result-cache and the table-cache push all the snapshot attributes of the three profiles towards those of OLTP workloads. The reason is that many OLAP queries can be served by the query-result-cache or the table-cache, while OLTP queries (for example a query to do the function of *updateShoppingCart*) cannot be served by the cache. The figures also show that the table-cache is more effective than the query-result-cache at the shifting. The distributions of the *Shopping* workload snapshot attributes under caching still fall in between that of the *Browsing* profile and the *Ordering* profile in most cases. The trained classifier uses these attributes under caching to classify a snapshot to be a "OLAP" snapshot or a "OLTP" snapshot.



(a) No Caching

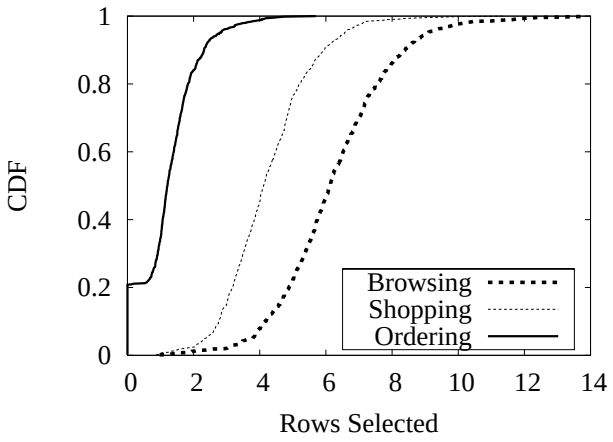


(b) Query-Result Cache

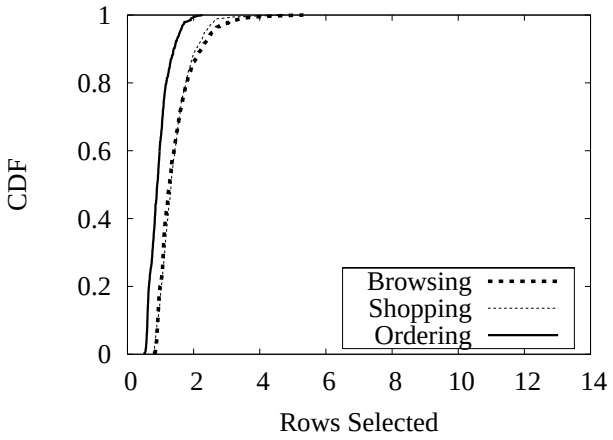


(c) Table Cache

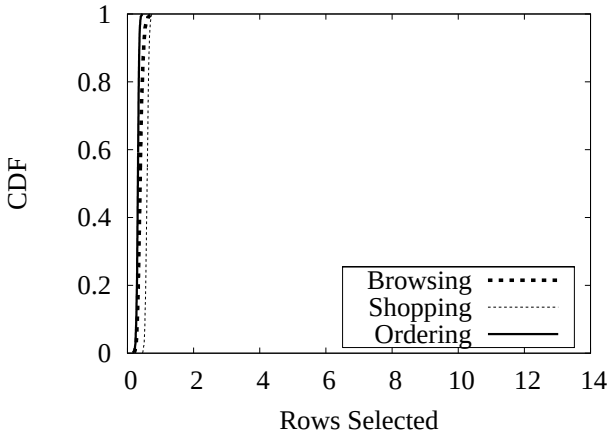
Figure 6.5: Pages Read Cumulative Distributions



(a) No Caching

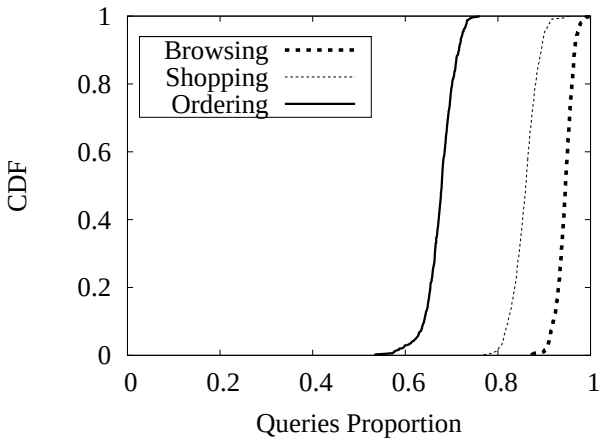


(b) Query-Result Cache

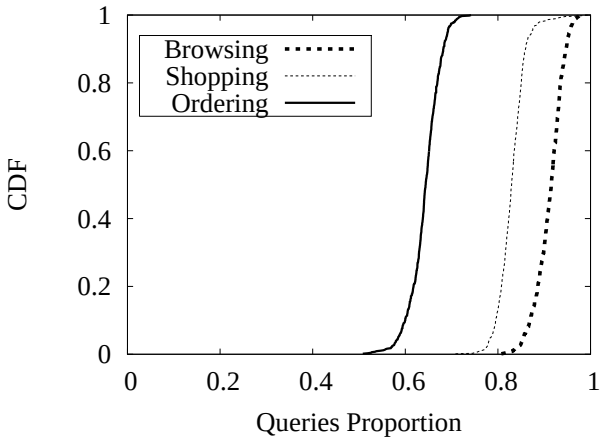


(c) Table Cache

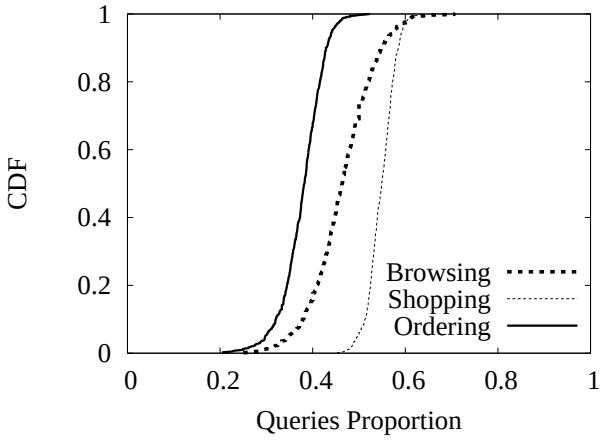
Figure 6.6: Rows Selected Cumulative Distributions



(a) No Caching

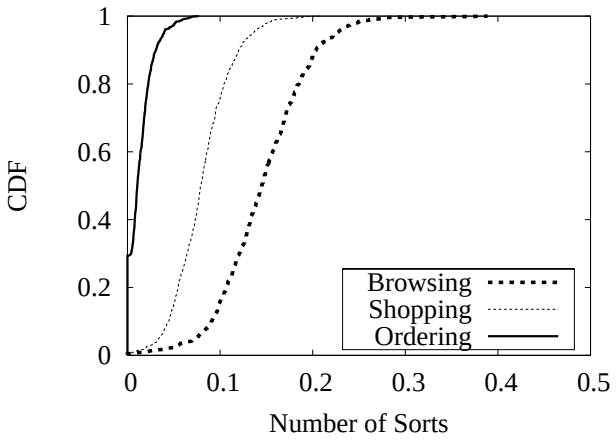


(b) Query-Result Cache

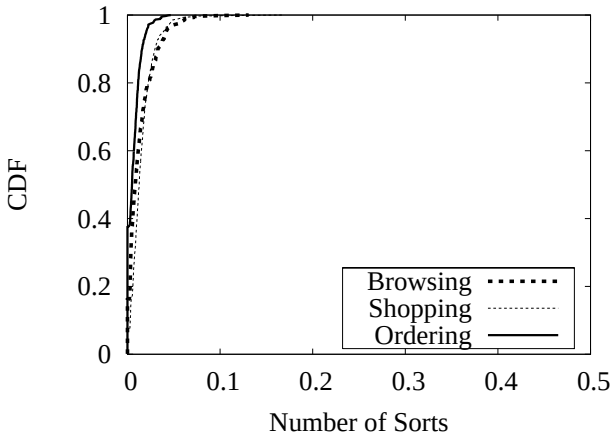


(c) Table Cache

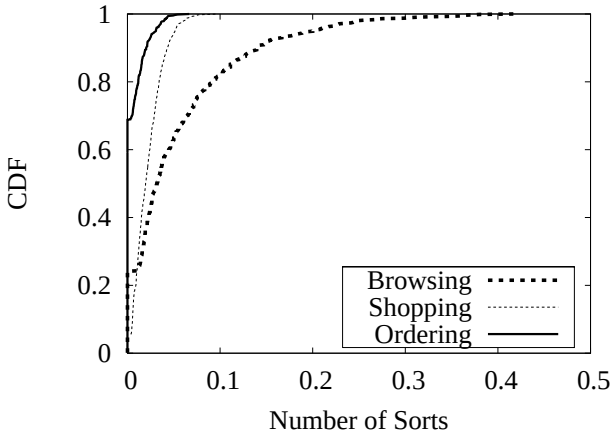
Figure 6.7: Queries Proportion Cumulative Distributions



(a) No Caching

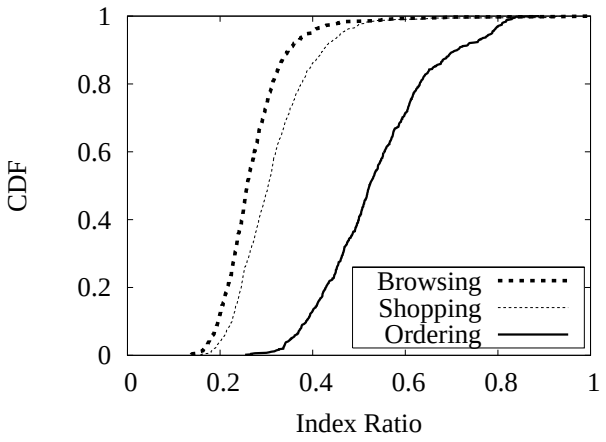


(b) Query-Result Cache

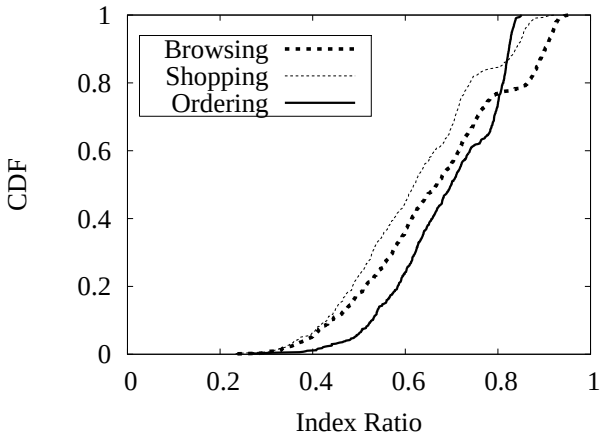


(c) Table Cache

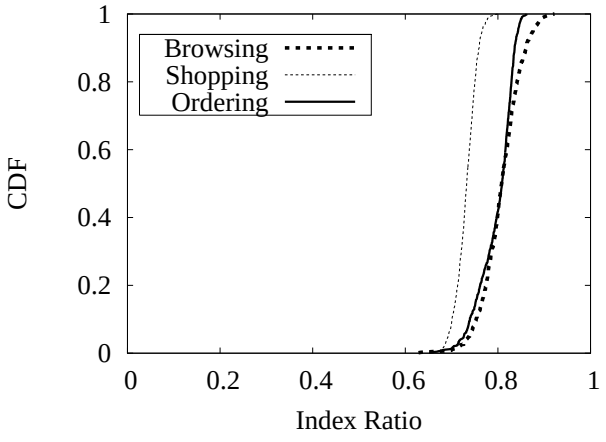
Figure 6.8: Number of Sorts Cumulative Distributions



(a) No Caching

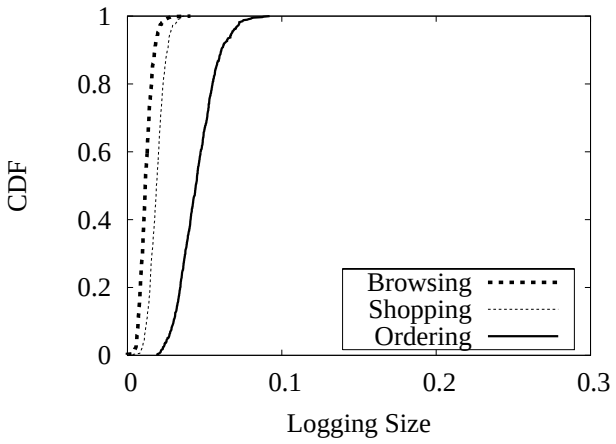


(b) Query-Result Cache

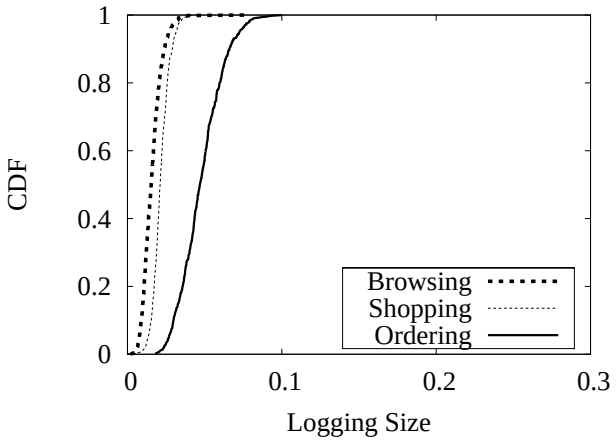


(c) Table Cache

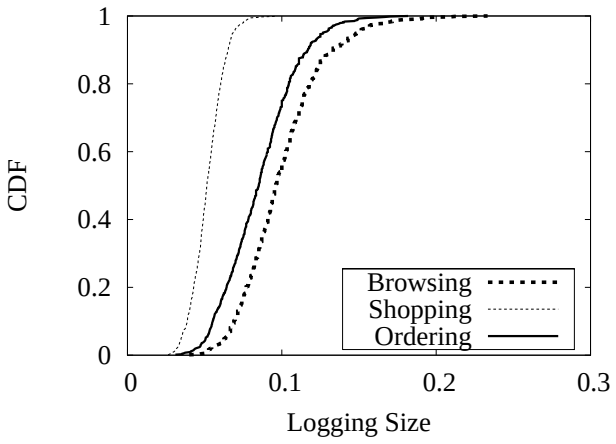
Figure 6.9: Index Ratio Cumulative Distributions



(a) No Caching



(b) Query-Result Cache



(c) Table Cache

Figure 6.10: Logging Size Cumulative Distributions

6.3 Classification Under Caching

Figure 6.11 shows the prediction results from the trained classifier J4.8. The three profiles under the no-cache configuration are also tested. The snapshots used for the no-cache configuration are new collected snapshots, not the training snapshots. The y-axis is the percentage of "OLTP" predicted by the classifier over all the predicted results from the target snapshots of a certain workload profile.

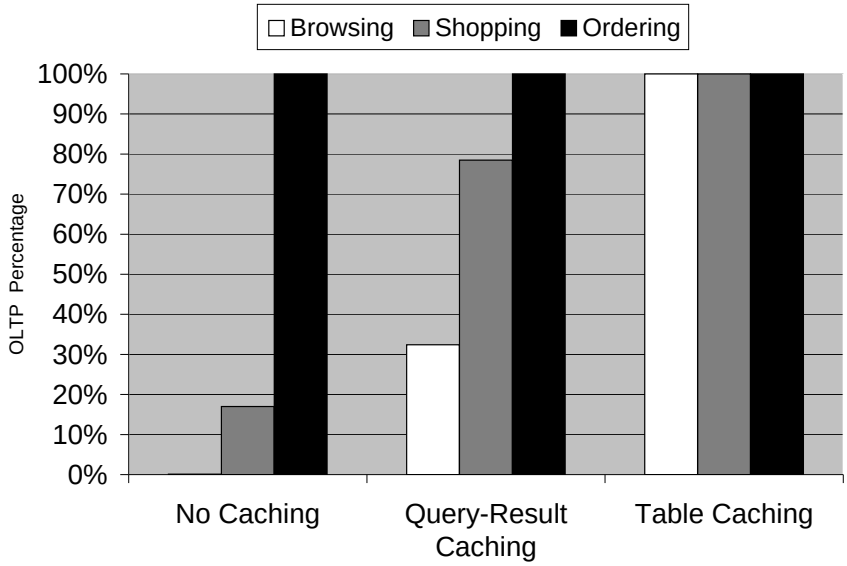


Figure 6.11: OLTP Percentage of TPC-W Profiles

The first group shows how the trained classifier predict for the *Browsing* profile, the *Shopping* profile, and the *Ordering* profile when there is no cache applied. Among all the predicted results, the *Browsing* profile gets 0.1% of "OLTP", while 17% for the Shopping profile and 100% for the Ordering profile. The second group is the predicted results under the query-result-cache. In this case, the *Browsing* profile gets 32% of "OLTP", while 79% for the *Shopping* profile and 100% for the *Ordering* profile. The third group under the table-cache shows that all the profiles get 100% of "OLTP" among the predicted results. This clearly indicates that under the table-cache, the *Browsing* profile and the *Shopping* profile change to "OLTP". This figure also shows that query-result-cache is also effective at pushing TPC-W

Browsing profile and *Shopping* profile towards "OLTP" workloads.

The results suggest that if table caching is deployed, a DBMS tuned for OLAP workloads similar to the TPC-W profiles should be adjusted to an OLTP-only DBMS. As well, if table caching is used, the benchmarking of E-commerce workload can just focus on the performance of the OLTP (the *Ordering* profile) transactions. As E-commerce Web sites provide better searching and data mining services, the proportion of cache-able queries may increase in future E-commerce workloads. Using dynamic caching may bring more benefits to such systems.

6.4 Summary

This chapter shows how the dynamic caching mechanisms affect the classification of database workloads. The snapshot attributes of the TPC-W *Browsing* profile and the *Ordering* profile have quite distinct distribution groups if there is no dynamic caching mechanism. The table-cache and query-result-cache change the snapshot attribute distributions of the TPC-W profiles. The dynamic caching mechanisms decrease the values of *Queries Proportion, Pages Read, Rows Selected, and Sorts*, and increase the values of *Index Ratio and Logging*. The trained classifier predicts all the three TPC-W profiles as "OLTP" workloads under the table-cache.

The experimental results in this chapter indicates that dynamic caching mechanisms can turn an "OLAP" workload into an "OLTP" workload seen by a database server. If table caching is used in an E-commerce system, a DBMS tuned for OLAP workloads similar to the TPC-W *Browsing* and *Shopping* profiles should be adjusted to an OLTP-only DBMS.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

The performance of the database server in an E-commerce system is crucial. Due to the wide use of dynamic caching, the workload seen by the database server is dramatically changed. Understanding these changes is important to the design, tuning and capacity planning of the database server.

This thesis studied the workload characteristics and classification of the database server in a benchmark E-commerce system. Two dynamic caching mechanisms are used: query-result-cache and table-cache. The main findings about the reduction effects of caching are: using dynamic cache can considerably reduce the response time of the database server when it is heavily loaded; using dynamic cache can considerably reduce the CPU usage of the database server when it is heavily loaded; using dynamic cache can substantially reduce the number of page references; using dynamic cache can reduce the number of disk I/Os.

The experimental results in this thesis also show that dynamic caching mechanisms affect the temporal locality and spatial locality that are exhibited in the TPC-W workload references. In most cases, the temporal locality becomes worse after using dynamic cache, but to a smaller degree than that reported in file servers and Web proxies. Interestingly, for workloads with few cacheable queries (i.e., the TPC-W *Ordering* workload), using table-cache increases temporal locality of database page references. This result is contrary to that in file servers and Web proxies. The

write temporal locality exhibited in the TPC-W workload references is poor in both systems with and without dynamic cache. The TPC-W workload references exhibit moderate spatial locality, which can be further reduced by use of dynamic cache.

In this thesis, it is found that the snapshot attributes of the TPC-W *Browsing* profile and the *Ordering* profile have quite distinct distribution groups if there is no dynamic caching mechanism. Moreover, Dynamic caching mechanisms typically blur the distinctions of the snapshot attributes among the TPC-W profiles. Dynamic caching mechanisms can dramatically pushes the TPC-W *Browsing* profile and *Shopping* profile towards OLTP workloads. This suggests that if caching is deployed, a DBMS tuned for OLAP workloads similar to TPC-W *Browsing* profile or *Shopping* should be adjusted to an OLTP-oriented DBMS.

Another finding of this thesis is that the buffer pool replacement algorithms of LRU, ARC, and LIRS do not show much performance difference in both systems with and without dynamic cache. It is also found that the table-cache does a better job than the query-result cache upon both reducing the database server load and pushing the TPC-W *Browsing* profile and *Shopping* profile towards OLTP workloads. The table-cache is more effective than the query-result cache at reducing the response time, database server CPU utilization, the number of page references, and the number of disk I/Os.

7.2 Future Work

Future possible research directions of this study include: further configure the experimental system and investigate the characteristics and classification of database workloads; study the effects that dynamic caching mechanisms have on database server workloads by breaking down the response time; examine real e-commerce workloads under dynamic caching mechanisms.

The current experimental configurations can be extended in various ways. To simulate a real E-commerce system more closely, the table-cache in this study can be implemented to process queries by accessing table data, not return faked results;

also, the network latency can be configured to reflect the corresponding delays. Buffer pool prefetching mechanisms can be implemented in the buffer pool simulator program *bp*. The buffer pool miss ratio can be used to measure how prefetching mechanisms affect the workloads under dynamic caching mechanisms. Slightly lower miss ratios are expected since prefetched pages can serve some sequential accesses.

According to a study [53] on the bottlenecks in E-commerce system, the number of items in the *ITEM* table can be an important factor for the database performance. The number of items used in this study is $10,000$. Large numbers can be used for future experiments, for example $100,000$ or $1,000,000$. Also, future experiments can be designed to investigate using parallel database and distributed database techniques in E-commerce systems. It would be very interesting to compare the performance difference between these techniques and caching mechanisms in an E-commerce environment.

The version of the TPC-W implemented in this study is $v1.8$. A new TPC-W version $v2.0$ is now available for public review [41]. The new TPC-W version simulates the *Business to Business* E-commerce model. It differs from the version 1.8. Take the Web interactions as an example, instead of including 14 Web interactions in version 1.8, the new version has 9 different Web interactions: *New Customer, Change Payment, Create Order, Shipping, Stock Management, Order Status, New Products, Product Detail,* and *Change Item*. The new TPC-W version can be implemented to study how dynamic caching mechanisms affect the database workloads in a *Business to Business* model.

Another main direction in which this work could be taken is to study the effects that dynamic caching mechanisms have on database response time. The response time of a database server can be broken down to *CPU time, I/O time,* and *Lock time* [29]. The CPU time is the time a transaction spends running on the processor and waiting for the processor. The I/O time is the time a transaction spends issuing and waiting for synchronous I/O to complete, and the lock time is the time a transaction spends waiting for database locks. If there is a small number of emulated browsers (less than 200) and no caching is used, the CPU time accounts

for most of the response time for a TPC-W transaction, about 80%, while the I/O time accounts for about 15% of the response time and the lock time accounts for 5% or less [29]. It could be very interesting to study how the response time breaks down when there are a large number of emulated browsers, for example $1,000$. The lock time is expected to increase its percentage in the response time since a large number of emulated browsers can cause high data contention. Also, it can be very interesting to study how dynamic caching mechanisms affect the breakdown of the response time when the database server is heavily loaded.

Real E-commerce workloads under dynamic caching would also be interesting to study. Analyzing traces from real E-commerce systems under dynamic caching mechanisms can reflect how the TPC-W workloads represent real workloads and how the dynamic caching mechanisms affect the real workloads on the database server. However, real database traces are hard to get since they are usually confidential for commercial reasons.

References

[1] Mehmet Altinel, Christof Bornhovd, Sailesh Krishnamurthy, C. Mohan, Hamid Pirahesh, and Berthold Reinwald1. Cache Tables: Paving the Way for an Adaptive Database Cache. In *Proceedings of the* 29^{th} *International Conference on Very Large Data Bases (VLDB 2003)*, pages 718–729, Berlin, Germany, September 2003.

[2] Khalil Amiri, Sanghyun Park, Renu Tewari, and Sriram Padmanabhan. DBProxy: A Dynamic Data Cache for Web Applications. In *Proceedings of the* 19^{th} *IEEE International Conference on Data Engineering (ICDE'03)*, pages 821–831, Bangalore, India, March 2003.

[3] Jesse Anton, Lawrence Jacobs, Xiang Liu, Jordan Parker, Zheng Zeng, and Tie Zhong. Web Caching for Database Applications with Oracle Web Cache. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, pages 594–599, Madison, WI, June 2002.

[4] Guangwei Bai and Carey Williamson. Workload Characterization in Web Caching Hierarchies. In *Proceedings of IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'02)*, pages 13–22, Fort Worth, TX, October 2002.

[5] Phil Bradley. *The Business and Economy Internet Resource Handbook*. Library Association Publishing, October 2000.

[6] Harold W. Cain, Ravi Rajwar, Morris Marden, and Mikko H. Lipasti. An Architectural Evaluation of Java TPC-W. In *Proceedings of the* 7^{th} *International Symposium on High-Performance Computer Architecture*, pages 229–240, Monterrey, Mexico, January 2001.

[7] Hao Che, Zhijung Wang, and Ye Tung. Analysis and Design of Hierarchical Web Caching Systems. In *Proceedings of IEEE INFOCOM 2001*, pages 1416–1424, Anchorage, AL, April 2001.

[8] Zhifeng Chen, Yuanyuan Zhou, and Kai Li. Eviction-based Cache Placement for Storage Caches. In *Proceedings of the 2003 USENIX Annual Technical Conference (USENIX'03)*, pages 269–282, San Antonio, TX, June 2003.

[9] Jonathan Coppel. E-commerce: Impacts and Policy Challenges (Economics Development Working Papers No. 252), June 2000. Organization for Economic Cooperation and Development (OECD).

[10] Frank J. Corbató. A Paging Experiment With the Multics System. In *In Honor of P. M. Morse*, pages 217–228. MIT Press, Cambridge, Mass., 1969.

[11] Anindya Datta, Kaushik Dutta, Helen Thomas, Debra VanderMeer, Krithi Ramamritham, and Dan Fishman. A Comparative Study of Alternative Middle Tier Caching Solutions to Support Dynamic Web Content Acceleration. In *Proceedings of the* 27^{th} *International Conference on Very Large Data Bases (VLDB 2001)*, pages 667–670, Roma, Italy, September 2001.

[12] Anindya Datta, Kaushik Dutta, Helen Thomas, Debra VanderMeer, Suresha, and Krithi Ramamritham. Proxy-based Acceleration of Dynamically Generated Content on the World Wide Web: An Approach and Implementation. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, pages 97–108, Madison, WI, June 2002.

[13] Ronald P. Doyle, Jeffrey S. Chase, Syam Gadde, and Amin M. Vahdat. The Trickle-Down Effect: Web Caching and Server Request Distribution. In *Proceedings of Web Caching and Content Delivery Workshop (WCW'01)*, Boston, MA, June 2001.

[14] Wolfgang Effelsberg and Theo Haerder. Principles of Database Buffer Management. *ACM Transactions on Database Systems*, 9(4):560–595, December 1984.

[15] Said S. Elnaffar. A Methodology for Auto-recognizing DBMS Workloads. In *Proceedings of the* 12^{th} *Annual IBM Centre for Advanced Studies Conference (CASCON 2002)*, pages 74–88, Toronto, ON, September 2002.

[16] Rodrigo Fonseca, Virgílio Almeida, Mark Crovella, and Bruno Abrahão. On the Intrinsic Locality Properties of Web Reference Streams. In *Proceedings of IEEE INFOCOM 2003*, pages 448–458, San Francisco, CA, March-April 2003.

[17] Kevin W. Froese and Richard B. Bunt. The Effect of Client Caching on File Server Workloads. In *Proceedings of the* 29^{th} *Hawaii International Conference on System Sciences (HICSS'96)*, pages 150–159, Kihei, HI, January 1996.

[18] Xubin He and Qing Yang. Performance Evaluation of Distributed Web Server Architectures under E-Commerce Workloads. In *Proceedings of the* 1^{st} *International Conference on Internet Computing (IC'2000)*, pages 285–292, Las Vegas, Nevada, June 2000.

[19] Windsor W. Hsu, Alan Jay Smith, and Honesty C. Young. Analysis of the Characteristics of Production Database Workloads and Comparison with the TPC Benchmarks. Technical Report CSD-99-1070, Computer Science Division, University of California, Berkeley, CA, November 1999.

[20] Windsor W. Hsu, Alan Jay Smith, and Honesty C. Young. I/O Reference Behavior of Production Database Workloads and the TPC Benchmarks – An Analysis at the Logical Level. Technical Report CSD-99-1071, Computer Science Division, University of California, Berkeley, CA, November 1999.

[21] Christian Huitema. Network vs. Server Issues in End-to-end Performance. In *Keynote Speech at Workshop on Performance and Architecture of Web Servers (PAWS-2000)*, Santa Clara, CA, June 2000.

[22] Raj Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling.* John Wiley & Sons, New York, NY, 1991.

[23] Song Jiang and Xiaodong Zhang. LIRS: An Efficient Low Inter-reference Recency Set Replacement Policy to Improve Buffer Cache Performance. In *Proceedings of the 2002 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 31–42, Marina Del Rey, CA, 2002.

[24] John P. Kearn and Samuel DeFazio. Diversity in Database Reference Behavior. *ACM Performance Evaluation Review*, 17(1):11–19, May 1989.

[25] Wen-Syan Li, Wang-Pin Hsiung, Dmitri V. Kalashnikov, Radu Sion, Oliver Po, Divyakant Agrawal, and K. Selçuk Candan. Issues and Evaluations of Caching Solutions for Web Application Acceleration. In *Proceedings of the* 28^{th} *International Conference on Very Large Data Bases (VLDB 2002)*, pages 1019–1030, Hong Kong, China, August 2002.

[26] Qiong Luo, Sailesh Krishnamurthy, C. Mohan Hamid Pirahesh, Honguk Woo, Bruce G. Lindsay, and Jeffrey F. Naughton. Middle-Tier Database Caching for e-Business. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, pages 600–611, Madison, WI, June 2002.

[27] Qiong Luo and Jeffrey F. Naughton. Form-Based Proxy Caching for Database-Backed Web Sites. In *Proceedings of the* 27^{th} *International Conference on Very Large Data Bases (VLDB 2001)*, pages 191–200, Roma, Italy, September 2001.

[28] Anirban Mahanti, Carey Williamson, and Derek Eager. Traffic Analysis of a Web Proxy Caching Hierarchy. *IEEE Network*, 14(3):16–23, May 2000.

[29] David T. McWherter, Bianca Schroeder, Anastassia Ailamaki, and Mor Harchol-Balter. Priority Mechanisms for OLTP and Transactional Web Applications. In *Proceedings of the* 20^{th} *International Conference on Data Engineering (ICDE 2004)*, pages 535–546, Boston, MA, April 2004.

[30] Nimrod Megiddo and Dharmendra S. Modha. ARC: A Self-Tuning, Low Overhead Replacement Cache. In *Proceedings of the* 2^{nd} *USENIX Conference on File and Storage Technologies (FAST'03)*, pages 115–130, San Francisco, CA, March 2003.

[31] Daniel A. Menascé and Virgilio A.F. Almeida. *Scaling for E-Business: Technologies, Models, Performance, and Capacity Planning.* Prentice Hall Professional Technical References, Upper Saddle, NJ, 1^{st} edition, May 2000.

[32] C. Mohan. Caching Technologies for Web Applications. Tutorial at the 27^{th} International Conference on Very Large Data Bases (VLDB 2001), Roma, Italy, September 2001.

[33] Michael N. Nelson, Brent B. Welch, and John K. Ousterhout. Caching in the Sprite Network File System. *ACM Transactions on Computer Systems*, 6(1):134–154, February 1988.

[34] M. Tamer Özsu and Patrick Valduriez. Distributed and Parallel Database Systems. *ACM Computing Surveys*, 28(1):125–128, March 1996.

[35] David A. Patterson and John L. Hennessy. *Computer Organization and Design: The Hardware/Software Interface*. Morgan Kaufmann, 3^{rd} edition, August 2004.

[36] Tom Sheldon. *Encyclopedia of Networking & Telecommunications*. McGraw-Hill, New York, NY, 3^{rd} edition, June 2001.

[37] Abraham Silberschatz, Henry F. Korth, and S. Sudarshan. *Database System Concepts*. McGraw-Hill, New York, NY, 4^{th} edition, July 2001.

[38] Alan J. Smith. Sequentiality and Prefetching in Database Systems. *ACM Transactions on Database Systems*, 3(3):223–247, September 1978.

[39] Michael Stonebraker. Operating System Support for Database Management. *Communications of the ACM*, 24(7):412–418, 1981.

[40] The TimesTen Team. High Performance and Scalability through Application-Tier, In-Memory Data Management. In *Proceedings of the* 26^{th} *International Conference on Very Large Data Bases (VLDB 2000)*, pages 677–680, Cairo, Egypt, September 2000.

[41] TPC Benchmark^{TM} W. Website, 2005. `http://www.tpc.org/tpcw/`.

[42] Udaykiran Vallamsetty, Krishna Kant, and Prasant Mohapatra. Characterization of E-Commerce Traffic. *Electronic Commerce Research*, 3(1-2):167–192, January 2003.

[43] Jia Wang. A Survey of Web Caching Schemes for the Internet. *ACM Computer Communication Review*, 29(5):36–46, October 1999.

[44] Qing Wang. Workload Characterization and Customer Interaction at E-commerce Web Server. Master's thesis, Department of Computer Science, University of Saskatchewan, August 2004.

[45] Wenguang Wang and Rick Bunt. Simulating DB2 Buffer Pool Management. In *Proceedings of the* 10^{th} *Annual IBM Centre for Advanced Studies Conference (CASCON 2000)*, pages 88–97, Toronto, ON, November 2000.

[46] Ted J. Wasserman, Patrick Martin, David B. Skillicorn, and Haider Rizvi. Developing a Characterization of Business Intelligence Workloads for Sizing New Database Systems. In *Proceedings of the* 7^{th} *ACM International Workshop on Data Warehousing and OLAP*, pages 7–13, Washington, DC, November 2004.

[47] Dee A. B. Weikle, Sally A. McKee, and Wm.A. Wulf. Caches As Filters: A New Approach To Cache Analysis. In *Proceedings of IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'98)*, pages 2–12, Montreal, PQ, July 1998.

[48] Carey Williamson. On Filter Effects in Web Caching Hierarchies. *ACM Transactions on Internet Technology*, 2(1):47–77, 2002.

[49] Darryl L. Willick, Derek L. Eager, and Richard B. Bunt. Disk Cache Replacement Policies for Network Fileservers. In *Proceedings of the* 13^{th} *International Conference on Distributed Computing Systems (ICDCS 1993)*, pages 2–11, Pittsburgh, PA, May 1993.

[50] Ian H. Witten, Eibe Frank, and Morgan Kaufmann. *Data Mining*. Morgan Kaufmann, San Francisco, CA, 1^{st} edition, 1999.

[51] Theodore M. Wong and John Wilkes. My Cache or Yours? Making Storage More Exclusive. In *Proceedings of the 2002 USENIX Annual Technical Conference (USENIX'02)*, pages 161–175, Monterey, CA, June 2002.

[52] Khaled Yagoub, Daniela Florescu, Valérie Issarny, and Patrick Valduriez. Caching Strategies for Data-Intensive Web Sites. In *Proceedings of* 26^{th} *International Conference on Very Large Data Bases (VLDB 2000)*, pages 188–199, Cairo, Egypt, September 2000.

[53] Qi Zhang, Alma Riska, Erik Riedel, and Evgenia Smirni. Bottlenecks and Their Performance Implications in E-commerce Systems. In *Proceedings of the* 9^{th} *International Workshop on Web Content Caching and Distribution (WCW 2004)*, pages 273–282, Beijing, China, October 2004.

[54] Yuanyuan Zhou, James F. Philbin, and Kai Li. The Multi-Queue Replacement Algorithm for Second Level Buffer Caches. In *Proceedings of the 2001 USENIX Annual Technical Conference (USENIX'01)*, pages 91–104, Boston, MA, June 2001.

[55] Yingwu Zhu and Yiming Hu. Can Large Disk Built-in Caches Really Improve System Performance? In *Proceedings of the 2002 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 284–285, Marina Del Rey, CA, 2002.