WORKLOAD CHARACTERIZATION OF AUTONOMIC DBMSs USING STATISTICAL AND DATA MINING TECHNIQUES

By Zerihun Zewdu Tegegn

A Thesis
Submitted to the school of Graduate Studies of Addis Ababa University in partial fulfillment of the requirements for the Degree of Master of Science in Computer Science

Addis Ababa University, Faculty of Informatics
Department of Computer Science

Addis Ababa University
School of Graduate Studies
Faculty of Informatics
Department of Computer Science

Workload Characterization of Autonomic DBMSs using Statistical and Data mining techniques

BY: ZERIHUN ZEWDU TEGEGN

Advisor: Meiso Denko (Ph. D.)
Co-Advisor: Mulugeta Libsie (Ph. D.)

Approved By:
Examining Board:

1. Mulugeta Libsie (Ph. D.), Co-Advisor _____
2. _____ _____
3. _____ _____
4. _____ _____
5. _____ _____

ACKNOWLEDGMENT

I would like to express my sincere gratitude to all those who gave me support to complete this thesis. First and foremost, I would like to thank my advisors, Dr. Mulugeta Libsie and Dr. Mieso Denko, for guiding me through the completion of this thesis work. I have learned a lot from their invaluable comments and suggestions throughout the course of the research period.

I would like to thank the Department of Computer Science staff in Addis Ababa University for being supportive in a lot of ways while I am working on my thesis. A special thanks goes to Dr. Solomon Atnafu from the department.

I would like to pass my gratitude to all fellow students for working together and helping each other during some hard times.

Finally I would like to thank my family for all their support and motivation.

TABLE OF CONTENTS

Pages

List of Tables

List of Figures

List of Acronyms

DSS	Decision Support Systems
OLTP	Online Transaction Processing
TPC	Transaction Processing Performance Council
TPC-H	Transaction Processing Performance Council – H
TPC-C	Transaction Processing Performance Council – C
DB2	IBM’s database management system, Database 2
MySQL	An open source Database Management System
QUEST	Quick Unbiased Efficient Statistical Tree
C&RT	Classification and Regression Tree
CHAID	Chi-square Automatic Interaction Detection
SPSS	Statistical Package for Social Sciences
CR	Com_ratio, MySQL status variable
IDR	Innodb_data_read, MySQL status variable
QH	Qcache_hits, MySQL status variable
ILW	Innodb_log_writes, MySQL status variable
IBRAR	Innodb_bufferpool_read_ahead_rnd, MySQL status variable
SR	Sort_rows, MySQL status variable
IRR	Innodb_rows_read, MySQL status variable
Q	Questions, MySQL status variable
Self-CHOP	Self-Configure, Self-Heal, Self-Optimize & Self-Protect

Abstract

Autonomic configuration is one of the most important components of an autonomic system. Database Management Systems (DBMSs) are one of the areas where autonomic configuration is highly required. In order for a DBMS to configure itself on changing external workloads, it should be able to detect and classify the workloads into their dominant categories, mainly into DSS (Decision Support Systems) and OLTP (Online Transaction Processing). Previous research works in this area have proposed a methodology for classification of workloads. But the tests are performed using limited algorithms and on only one commercial DBMS.

In this thesis a model where an autonomic DBMS can identify and characterize the type of workload acting up on it is developed and the most important database status variables which are highly affected by changing workloads are identified. This is important for a self configuring autonomic DBMS because it needs to reconfigure itself based on identified changing workloads. Two algorithms are selected for database workload classification: hierarchical clustering and classification & regression tree for classifying database workloads after running database workloads from TPC benchmark queries and transactions. The costs of these workloads are measured in terms of status variables of the selected DBMS (MySQL). These costs are used to show whether a workload is DSS or OLTP using the selected classification algorithms.

After a set of extensive experiments and analyses, we have found out that all the DBMS status variables are not equally important in classifying the collected workloads. In fact, some of the workloads do not have a significant relevance apart from increasing the classification complexity. We have identified these variables and listed them in this thesis. Even though both the selected classification algorithms are good at classifying the collected workloads, hierarchical clustering algorithm has an additional advantage of showing the degree of correlation among clusters. This can be important in the area of database workload shift detection.

Chapter 1

Introduction

Computer networks and computer systems are growing in size and complexity and computer systems that weigh in millions of lines of code require skilled IT professionals to install, configure, and maintain. Computing system's complexity seems to be approaching the limits of human capability and yet the need to increased interconnectivity and integration rushes to a point where systems become too complex and massive for even the most skilled system integrators and administrators to install, configure, optimize, maintain, and merge [1]. IBM has introduced the idea of autonomic computing systems that can manage themselves given high level objectives from administrators, in 2001. The issue of the new paradigm, Autonomic Computing, is to design, develop, deploy and manage systems by inspiring from strategies used by biological systems to cope with complexity, heterogeneity, and uncertainty. An autonomic system has four major characteristics – self-configure, self-heal, self-optimize, and self-protect – which are often referred to as self-CHOP characteristics [2]. There is a significant difference between autonomic computing and autonomic communication. While autonomic communication is more oriented towards distributed systems and services and to the management of network resources at both the infrastructure and the user levels, autonomic computing is more directly oriented towards application software and management of computing resources [3].

One of the characteristics of an autonomic system is its ability to configure itself. For a system to best handle changing environments, it must be able to configure itself autonomically. As some systems present hundreds of configuration alternatives, configuration can be very difficult and time-consuming which can only be performed by a very skilled and experienced

system administrator knowing each and every configuration parameter with all its possible values.

Another characteristic of an autonomic system is its ability to self-heal – It must be able to recover from routine and extraordinary events that might cause some of its parts to malfunction. Healing in a computing system means calling into an action redundant or underutilized elements to act as replacement parts.

An autonomic computing system needs to seek ways to improve its operation, identifying and seizing opportunities to make itself more efficient in performance. An autonomic system monitors, experiments with, and tunes its own parameters and will learn to make appropriate choices about keeping functions or outsourcing them. It proactively seeks to upgrade its function by finding, verifying, and applying the latest updates.

The last characteristic of an autonomic system is its ability to protect itself against possible attacks. An autonomic system will be self protecting in two senses. It will defend the system as a whole against large-scale, correlated problems arising from malicious attacks or cascading failures that remain uncorrected by self-healing measures. It also anticipates problems based on early reports from sensors and takes steps to avoid or mitigate them.

Kephart and Chess [1] summarize these four characteristics of self-management as they are now and what they would be with autonomic computing as shown in Table 1-1.

Table 1-1 Four aspects of self-management as they are now and would be with autonomic computing.

Concept	Current Computing	Autonomic Computing
Self-Configuration	Corporate data centers have multiple vendors and platforms. Installing, configuring and integrating systems is time consuming and error prone.	Automated configuration of components and systems follows high-level policies. Rest of system adjusts automatically and seamlessly.
Self-Optimization	Systems have hundreds of manually set, nonlinear tuning parameters, and their number increases with each release.	Components and systems continually seek opportunities to improve their own performance and efficiency.
Self-Healing	Problem determination in large, complex systems can take a team of programmers weeks.	System automatically detects, diagnoses, and repairs localized software and hardware problems.
Self-Protection	Detection of and recovery from attacks and cascading failures is manual.	System automatically defends against malicious attacks or cascading failures. It uses early warning to anticipate and prevent system wide failures.

One of the areas where self configuration is needed is the area of relational database management systems. This is because DBMSs these days have a very high volume of data with a lot of queries applied on them every minute. It is very important that these systems configure themselves upon changing environmental workloads with minimal human intervention. Database administrators tune a DBMS based on their knowledge of the system and its workload. In other words, the type of workload on a DBMS is a key criterion for tuning

[4]. The focus of this research work is characterization of a DBMS workload using statistical and data mining techniques.

1.1.Motivation

An autonomic DBMS needs to sense as to what kind of workload is acting up on it and at what intensity before it goes on tuning appropriate parameters to perform better. Workload characterization is important for a DBMS to recognize the type of workload and act in tuning appropriate parameters.

A database workload can be predominantly of an Online Transaction Processing (OLTP) or a Decision Support System (DSS). Although there is no clear boundary to decide whether a workload is OLTP or DSS, there are some peculiar features for differentiating between the two that will be explained later in the thesis. It is the effort of classifying workloads as a percentage of OLTP and DSS that is referred to as workload characterization.

Although there are some research works on the area of DBMS workload characterization [4, 5, 6, 7, 8], all the experiments in all the research works we have reviewed are always performed in one DBMS namely IBM's DB2 which tends to limit the usability of the workload characteristics in other DBMSs. Furthermore the classifier used to classify workloads is again always a decision tree algorithm and there is no clear description of how database workload identification is performed.

There are other classification techniques from the field of data mining and statistics which were not attempted by other researchers in the past as far as the published papers we have reviewed are concerned and the fact that this issue is left as a future research area by E. Krayzman [9].

1.2. Problem Statement

The main objective of this thesis work is to develop a method where a DBMS can identify and characterize the type of workload acting upon it using statistical and data mining techniques. This is an important step for an autonomic DBMS to auto configure itself up on changing database workloads.

Specific objectives include:

- Propose a technique for detecting different types of database workloads.
- Develop a data mining and statistical model for database workload classification.
- Test the performance of the classification technique using industry standard benchmarks.
- Identify database status variables which have a significant influence in identifying workloads as compared to others.

1.3. Scope

This research work is limited to the four tasks mentioned above. It does not attempt to adjust database configuration parameters depending on the output of the workload classifier. For this reason this research work can only be taken as a partial problem solving attempt in autonomic database configuration.

The entire experiment in this research work is performed in MySQL database and most of the workload detection database status variables selected may not be available as they are in other DBMSs. But we believe that the basic idea still remains and the techniques used in this research work can also be used in related works with relative ease.

1.4. Methodology

We populate a MySQL database with data from TPC-Benchmark [12, 13] databases and run queries and transactions of different kind provided from the same benchmarking company. We measure the cost of each query and transaction in terms of MySQL status variables selected based on the criteria given in Section 5.3.1. Based on the proposed model in chapter 4, we identify the workloads as either OLTP or DSS using the selected classification and clustering algorithms. The selection of the algorithms is based on the explanation given in Section 4.2.1 and Section 4.2.2. Finally, we provide the output of the thesis.

1.5. Organization of the Thesis

This research work is organized into seven chapters. Chapter one gives a general overview of autonomic systems in general and autonomic DBMS in particular. It includes three subtopics: motivation of the research work, objectives of the research under the title problem statement, and scope of the research work. Chapter two explains major technologies and issues used in the research work. Chapter three is a review of related work around workload modeling, workload characterization and other research works related to autonomic DBMS and database workloads. Chapter four explains the experiment environment and the tools and technologies used in performing the experiment made. Chapter five is a detailed report of the findings of the experiment. Chapter six is a presentation of the experimental findings and finally chapter seven concludes the thesis work by providing a general overview of the thesis, contribution and possible future research area.

Chapter 2

Related Concepts

This chapter is an explanation of related technologies and concepts which are used in the thesis. It explains what a database workload is, how a DBMS reflects the availability of a workload using its status variables and how it is possible to take a snapshot of a particular workload. It also discusses available data classification techniques, a concept which is used in this thesis to classify database workloads into categories.

2.1. Database workload

A database workload is the demand placed upon the DBMS. This workload is commonly classified as Decision support Systems (DSS) and Online Transaction Processing (OLTP). DSS are specific classes of computerized information systems that support business and organizational decision making activities. DSS help decision makers compile useful information from raw data, documents, personal knowledge, and/or business models to identify and solve problems and make decisions. Typical information that DSS might gather and present are usually a compilation of data like for instance comparative sales figures between one week and the next. A list of such information is given in [10]:

As explained in Section 1.1, there is no clear boundary to differentiate between DSS and OLTP workloads. However, some peculiar features of DSS workload are [11]: large amounts of data, little write activity (except for temporary space and data loading), relatively few users (low number of concurrent connections), large and complex queries (processing lots of data) and access mainly via sequential scans.

In this thesis we have used data, queries and transactions from The Transaction Processing Performance Council (TPC) Benchmark™ [12, 13]. TPC-H is a decision support benchmark. It consists of a suite of business oriented ad-hoc queries, among other things. The queries and the data populating the database have been chosen to have broad industry-wide relevance while maintaining a sufficient degree of ease of implementation. The benchmark illustrates DSS that examine large volumes of data, execute queries with a high degree of complexity and give answers to critical business questions. TPC-H consists of eight separate and individual tables and twenty – two decision support queries and two database refresh functions that must be executed as part of the TPC-H benchmark. The minimum database required to run the benchmark holds business data from 10,000 suppliers containing almost ten million rows representing a raw storage capacity of about one gigabyte [12]. To provide some insight as to what these queries may look like, here is one of the twenty two queries listed in the TPC-H benchmark.

```
select s_acctbal, s_name, n_name, p_partkey, p_mfgr, s_address, s_phone, s_comment
from part, supplier, partsupp, nation, region
where
p_partkey = ps_partkey
and s_suppkey = ps_suppkey
and p_size = [SIZE]
and p_type like '%[TYPE]'
and s_nationkey = n_nationkey
and n_regionkey = r_regionkey
and r_name = '[REGION]'
```

```
and ps_supplycost = (
select min(ps_supplycost)
from partsupp, supplier, nation, region
where
p_partkey = ps_partkey
and s_suppkey = ps_suppkey
and s_nationkey = n_nationkey
and n_regionkey = r_regionkey
and r_name = '[REGION]'
)
order by s_acctbal desc, n_name, s_name, p_partkey;
```

TPC Benchmark C (TPC-C) is an OLTP workload. It is a mixture of read-only and update intensive transactions that simulate the activities found in complex OLTP application environments. It does so by exercising a breadth of system components associated with such environments, which are characterized by: the simultaneous execution of multiple transaction types that span a breadth of complexity, online and deferred transaction execution modes, multiple online terminal sessions, moderate system and application execution time, significant disk input/output, transaction integrity (ACID properties), non-uniform distribution of data access through primary and secondary keys, databases consisting of many tables with a wide variety of sizes, attributes, and relationships, and contention on data access and update [13].

Like DSS, there are some typical features of an OLTP workload which may include: lots of users, high transaction rates, transactions are relatively small in terms of data processed,

significant write activity (insert, update, and delete), fast access via indexes, high buffer cache usage and limited scope for parallelism [11].

A TPC-C database consists of nine separate individual tables with a wide range of record and involves a mix of five concurrent transactions of different types and complexity. The performance metric for this benchmark is expressed in transactions-per-minute-C (tpmC). The benchmark is comprised of basic operations designed to exercise system functionalities in a manner representative of complex OLTP application environments.

One of the five transactions specified in TPC-C benchmark called the new-order transaction consists of entering a complete order through a single database transaction. It represents a mid-weight, read-write transaction with a high frequency of execution and stringent response time requirements to satisfy on-line users [13].

2.2. MySQL status variables

The status of any MySQL server can be analyzed by taking a look at MySQL server's 251 status variables in the MySQL version 5.0 server on which this research is performed. These status variables provide information about the operation of the server. They can be viewed by using the SHOW STATUS statement in MySQL command prompt. Not all the 251 variables are important to study the workload characteristics of a database. The criteria for selecting some of the status variables among the 251 are given in Section 5.3.1. What follows is an explanation of the selected MySQL server variables that have particular relevance to workload characterization [7,11,14].

Com_select: is a MySQL server status variable that indicates the number of select statements executed on the server. Similarly, there is one status variable for each type of

statement. For instance, *Com_delete* and *Com_insert* count DELETE and INSERT statements, respectively. However, if a query result is returned from query cache, the server increments the Qcache_hits status variable, not Com_select. This is an important server status variable because for example the ratio of SELECT statements to the sum of UPDATE, INSERT, and DELETE statements is usually higher in DSS than in OLTP. We name this ratio Com_ratio.

Innodb_data_read: The amount of data read so far, in bytes. This variable is important because DSS transactions usually access larger portions of a database than do OLTP transactions. Similar to this variable is, another one called *Innodb_data_writes* which counts the number data writes.

Qcache_hits: The number of query cache hits. OLTP workloads experience a higher hit ratio on buffer pool cache area than DSS workloads.

Innodb_log_writes: The number of physical writes to the log file. An OLTP workload generates more logging activity than does a DSS workload because the OLTP transactions tend to have more read and modify statements in them.

Innodb_buffer_pool_read_ahead_rnd: The number of random read-aheads initiated by InnoDB. This happens when a query scans a large portion of a table but in a random order. DSS applications typically access large numbers of pages because of the substantial amount of full table and index scan operations. OLTP applications typically access relatively few random pages.

Sort_rows: The number of sorted rows. DSS transactions typically perform a larger number of sorts than do OLTP transactions.

Innodb_rows_read: The number of rows read from InnoDB tables. A DSS query returns more rows than an OLTP query. Other status variables related to this one are: Innodb_rows_inserted and Innodb_rows_deleted.

Questions: The number of statements that clients have sent to the server. OLTP transactions are usually executed by a number of clients concurrently. For this reason OLTP workloads send a lot of statements to the server than DSS workloads.

2.3. Snapshots

A snapshot is the status of a database server at any instant. A MySQL snapshot can be taken every second at any time. It is used to display the cost of a query. MySQL has a command "mysqladmin extended-status" which is used to display the status of all the 251 status variables mentioned above. If this command is issued first, that means taking the database state at this moment, then a particular query is run and finally same command is issued to get another state of the database after the query. It is possible to come up with the cost of the query by subtracting the first snapshot from the second. As an example Table 2-1 shows the cost of running a particular OLTP query. We have only displayed the first ten, one important variable from the middle, and the last ten as it is not important to display all the 251. The last variable shows the number of seconds the MySQL server has been up. Here in Table 2-1, the difference shows the amount of time the query took, which is 6193 seconds.

Table 2-1. The cost of an OLTP workload in terms of MySQL status variables

Variable_name	Value
Aborted_clients	0
Aborted_connects	0
Binlog_cache_disk_use	0
Binlog_cache_use	4945
Bytes_received	18581744
Bytes_sent	40842122
Com_admin_commands	0
Com_alter_db	0
Com_alter_table	0
Com_analyze	0
.	.
.	.
.	.
Qcache_hits	15973
.	.
.	.
.	.
Table_locks_immediate	133117
Table_locks_waited	0
Tc_log_max_pages_used	0
Tc_log_page_size	0
Tc_log_page_waits	0
Threads_cached	0
Threads_connected	0
Threads_created	12
Threads_running	0
Uptime	6193

2.4. Data Classification techniques

Data can be classified into groups based on the behavior it displays through different parameters. In order to classify a database workload into either OLTP or DSS we need to look at important parameters that can differentiate between the two for a particular database selected. The decision as to what kind of parameters to select for classification has a significant impact on the performance of the classification technique used as the selection of irrelevant parameters would not help. For example, Uptime is a MySQL server status variable which has little or no relevance to differentiate between a DSS and an OLTP workload. There are some data classification techniques in use from the statistical as well as data mining field

of study. We will explain classification trees and clustering here. The later is a method used in this research work.

2.4.1. Classification trees

Classification trees, also known as decision trees, are methods used to predict membership of cases or objects in the classes of a categorical dependent variable from their measurements on one or more predictor variables. Classification tree analysis is one of the main techniques used in data mining [16]. In this technique of data classification, all objects are considered as a single group. The group is split into two subgroups using high values of a variable for one group and low for the other. The two subgroups are then each split using the values of a second variable. The splitting process continues until a suitable stopping point is reached. For example, suppose subjects are to be classified as,

$\prod_{1:}$ heart-attack prone

$\prod_{2:}$ not heart-attack prone

On the basis of age, weight and exercise activity.

The procedure of classification is shown in Figure 2.1. The branch of the tree to the right is classified as not heart attack prone as the members of that branch are not over 45. The left wing of the branch is defined as being overweight and undertaking no regular exercise can be used to classify the subject as $\prod_1$ (heart attack prone) [17].

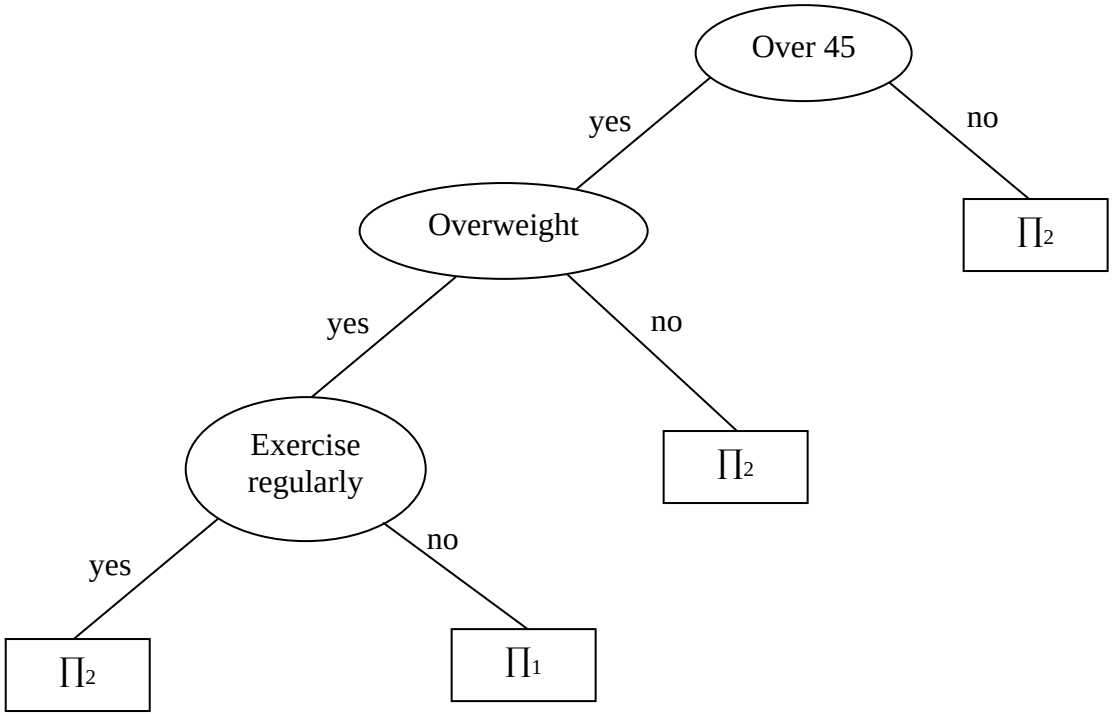

Figure 2.1 The procedure of classifying subjects using classification trees

A variety of classification tree algorithms are developed to predict membership of cases or objects in the classes of a categorical dependent variable from their measurements on one or more predictor variables. Some of the algorithms include: QUEST, C&RT, and CHAID.

QUEST (Quick, Unbiased, and Efficient Statistical Trees) is a classification tree algorithm that includes a number of innovative features for improving the reliability and efficiency of the classification tree it computes [16]. The main strengths of QUEST are unbiased variable selection and fast computational speed [18].

C&RT (Classification and Regression Tree), also abbreviated as CART, is a classification tree algorithm that uses an exhaustive grid search of all possible univariate splits to find the splits for a classification tree. As the name “Classification & Regression Trees” implies, this

component works with two problem types: those in which the dependent variable is categorical (referred to as classification problems) and those in which the target variable is continuous (referred to as regression problems) [21]. The QUEST and C&RT analysis options compliment each other. C&RT searches can be lengthy when there are a large number of predictor variables with many levels, and it is biased toward choosing predictor variables with more levels for splits, but because it employs an exhaustive search, it is guaranteed to find the splits producing the best classification.

CHAID (Chi-square Automatic interaction detection) is another classification tree algorithm that performs multi-level splits rather than binary splits when computing classification trees. A multi-level split performs k-1 splits, where k is the number of levels of the splitting variables, as compared to a binary split which performs one split regardless of the number of levels of the splitting variable [16].

2.4.2. Clustering

The term cluster analysis encompasses different algorithms and methods for grouping similar objects into respective categories. In cluster analysis, no assumption is made concerning the number of groups or the group structure [18]. Cluster analysis is an exploratory data analysis tool which aims at sorting different objects into groups in a way that the degree of association between two objects is maximal if they belong to the same group and minimal otherwise. This means that, cluster analysis can be used to discover structures in data without providing an explanation or interpretation [16]. Among the available clustering algorithms, we will mention Hierarchical tree clustering and K-means clustering because these to clustering algorithms are widely used in different applications and researches.

In a hierarchical tree clustering, the threshold regarding the decision when to declare two or more objects to be members of the same cluster is lowered in very small steps. As a result more and more objects are linked together and aggregate larger and larger clusters of increasingly dissimilar elements. Finally in the last step, all objects are joined together. The joining or tree clustering method uses the dissimilarities (similarities) or distances between objects when forming the clusters. Similarities are a set of rules that serve as criteria for grouping or separating items. These distances can be based on a single dimension or multiple dimensions, with each dimension representing a rule or condition for grouping objects. The most straightforward way of computing distances between objects in a multi-dimensional space it to compute Euclidean distances. If we had two or three dimensional space this measure is the actual geometric distance between objects in the space.

The k-means clustering algorithm will produce exactly k different clusters of greatest possible distinction. The best number of clusters leading to the greatest separation is not known at priori and must be computed from the data [16].

Chapter 3

Related Work

Identifying the type of workload in a database management system is an important component of autonomic configuration of databases. Allocations for resources such as main memory can be different depending on what type of workload the database management system encounters and how to change this and other configuration parameters by sensing how a workload changes [4]. There are some research works around autonomic configuration of databases. Some are directly related to workload characterization [6] and some are related to identification of the workload itself [4, 7]. This chapter is a review of research endeavors in the area of autonomic database configuration in general and workload identification and characterization in particular.

An Autonomic DBMS must be able to maintain optimal performance by dynamically sensing changes in the database workload and perform configuration and tuning tasks on the appropriate database configuration parameters including determining appropriate allocations for main memory areas such as the buffer pools and the sort heap, mapping database objects to buffer pools and adjusting the many DBMS configuration parameters to maintain acceptable performance [5]. Given the observed characteristics of a workload, an autonomic database tuner observes the influence of the workload to find the optimal settings for that workload [9].

3.1. Database Workload models

A research work in workload models for autonomic database management systems by Martin et. al [5] proposes exploratory workload models for the analysis and planning tasks of autonomic DBMS. Several performance oriented parameters including CPU utilization,

average rate of random I/O, and the average rate of sequential I/O were monitored for each individual query of the TPC-H benchmark. A combination of Singular Valued Decomposition (SVD) and Semi Discrete Decomposition (SDD) is used to partition the workload. SVD and SDD are examples of unsupervised data mining techniques where the goal is to discover structured information in the dataset without knowing or providing hints as to what that structure might look like.

Workload models, like the workloads they represent, are not static and will change over time. The effective creation and maintenance of workload models will be key to the viability of autonomic DBMSs. The maintenance approach can be either on-line or off-line. An off-line approach will periodically regenerate the model off-line and then replace the current version of the model with new version. An on-line approach to maintaining the workload models will, on the other hand, perform incremental maintenance. It will therefore have smaller storage requirements than an off-line approach but runs the risk of more interference with the applications running on the autonomic DBMS. The authors conclude by suggesting issues that must be addressed in the future: how to provide effective and efficient monitoring of the managed database element and how to perform effective incremental maintenance of the models, that is, methods by which recent data can be used to incrementally improve models and thus avoid the need for storing potentially large amounts of data in order to periodically recreate the models from scratch.

The structure and complexity of SQL statements, the makeup and behavior of transactions/queries, and the composition of relations and views in a DB-2 like environment is studied by Yu et. al [6] using a RElational Database Workload AnalyzeR (REDWAR), developed at IBM research lab. SQL statements composition is studied to analyze the

percentage of SQL statements having each type of constructs such as WHERE, GROUP BY, HAVING, ORDER BY, subquery, aggregate function, etc. The number of predicates in the WHERE clause, the distribution of the predicate types and operand types, and the number of columns in the SELECT clause are analyzed. The numbers of relations in the FROM, the GROUP BY, and the ORDER BY clauses, respectively, are also investigated. The transaction statistics gathered include the number of tuples scanned and retrieved/updated/inserted/deleted, the number of pages accessed, and the elapsed time for each transaction invocation. Then REDWAR is applied to examine a production DB2 system which runs an accounting-type application in a petroleum company. The authors conclude by presenting the workload analysis result in detail.

As described by Oh et. al. [19] database tuning is an activity that helps databases to perform more efficiently by avoiding or minimizing the time and effort required by database administrators. Configuration parameters for databases are complex and growing from time to time and database administrators should be aware of the characteristics and effects of each and every configuration parameter to come up with an optimal configuration that provides the best database performance.

This research work makes analysis on how resource usages respond by changing resource size in database systems, proposes a method that automatically select resources affecting the system performance significantly and that can be applied to resource usage analysis to select adequate resources that may enhance the performance of database systems. Resource usages may be recognized by collecting and analyzing workload data. Workload data collection involves determining target resources, resource changes, and performance indicators. Target resources are selected to be four database resources: data buffer, private memory, shared

memory, and I/O processes. The resource sizes are changed by tuning the database system parameters. Each resource size: data buffer, shared memory, private memory, and I/O processes can be determined by tuning four system parameters, namely: db_cache_size, shared_pool_size, pga_aggregate_target, and dbwr_io respectively. To analyze the performance characteristics of database systems, 14 performance indicators of a database system are given. A detailed analysis of the resource usages in terms of changing tuning parameters, hence changing the size of resources is also given. Selection of resources for autonomic tuning is performed in such a way that performance indicators that are affected by changes of the resource can be recognized as having a correlation coefficient and a variation coefficient. A correlation coefficient measures the degree to which two variables are linearly related. Correlation coefficient is used to determine the incremental or decremental relationship between resource size and performance indicator. The correlation coefficient does not consider the magnitude of changing values of the performance indicators. For this another value called variation coefficient is used. This value is important in that it indicates how much the performance indicators have to be changed which allows to identify whether the correlation coefficient is meaningful or not. In the experiment, the thresholds of the correlation coefficient and variation coefficient are set to be 0.5 and 0.1 respectively. A detailed analysis of these thresholds with the performance indicators is given in the paper.

3.2. DB2 database configuration tool

The research work by Kwan et. al. [20] is a presentation of an experimental result for a database configuration tool for DB2 Universal Database for UNIX and Windows, known as the DB2 configuration advisor. The performance of a database with tuned configuration has a measurable and significant improvement over a database with untuned configuration. The

increased complexity and volume of data seen in modern database systems has increased the task of database administrators. The database administrator now faces a diverse combination of tasks including:

- Requirements planning and capital investment.
- Database schema and data management design.
- Database physical design, creation and tuning.
- Maintenance and administration.
- Change management.

Within DB2, the Autonomic computing initiative at IBM has its objective to undergo a research focused on internal tuning and configuring technology within the product deliverable. The objective of the advisor is to define a set of initial database configuration parameters and memory assignments to optimize system performance without extensive monitoring of the system. These configuration parameters include parameters that control memory distribution (sorting, locking, caching database pages and working heaps), parallelism, I/O optimization (asynchronous page readers and writers), many aspects of logging (file size, buffer size), and recovery [20].

The DB2 Configuration advisor is designed on the principle that configuration choices can be made by modeling each database configuration setting as a mathematical expression which combines three sets of information regarding the system environment, the database characteristics and user priorities. The three sets of information are:

1. User specification of the database environment (designed as a small set of basic input, generally requiring minimal skill).

2. Autonomically sensed system characteristics (such as number of CPUs, disks, amount of RAM, number and size of relational tables, etc.)
3. Expert heuristics for database configuration, as reported by experienced database administrators and performance tuning experts.

The configuration model expresses the configuration settings as a mathematical expression. The model for the configuration settings is further divided into three distinct classes:

1. Independently modeled configuration settings, which can be modeled independent of other configuration settings.
2. Dependent configuration setting, where the value of one setting affects the model of another (or perhaps co-dependency).
3. Zero sum game relationships (such as memory for sort, caching, locking, etc.) in which a fixed resource must be divided among a set of configuration parameters.

To examine the effectiveness of the configuration advisor, four experiments are performed on systems running distinct workloads and environments, comparing the performance of the database after tuning by the configuration advisor to the system performance achieved through tuning by an expert database administrator. It is reported that the result after the experiment was the best system performance for the database with the new configuration performing at 224.72% of the original administrator-tuned throughput.

Further refinement of the current modeling for database configuration, incorporating automatic workload characterization schemes into the inputs that are used by the configuration advisor and investigating closed-loop performance feedback for the advisor are left as an open research area.

3.3. Autonomic buffer pool configuration

A research work by Powley et. al [25] is an illustration of how autonomic principles can be applied to a DBMS to provide autonomic sizing of buffer pools, a key resource in a DBMS, using the open source database, PostgreSQL. Effective use of the buffer area can influence the performance of a DBMS by reducing the number of disk accesses performed by a transaction. Many DBMSs divide the buffer area into a number of independent buffer pools and database objects are allocated among the pools. PostgreSQL does not implement multiple buffer pools, so it is needed to add multiple buffer pool functionality to the DBMS. To support the monitoring required for the buffer pool sizing algorithm, additional code is added to the PostgreSQL statistics collector to include statistics for each buffer pool access including the number reads, and the average data access time incurred to fetch a data object. An analysis/diagnosis stage analyzes the performance data collected by the monitor to determine whether or not there has been a shift in performance and, if so determining the possible causes. A suitable criterion to evaluate the efficiency of the buffer pool is average data access time, that is, the average time to satisfy a logical read request. A set of experiments were performed and the results are given.

3.4. Shifts in database workloads

Since a DBMS may experience changes in the type of workload it handles during its normal processing cycle, it is not enough for autonomic DBMSs to identify the current type of workload, but also to predict when a change in the workload type will occur. Elnaffar et. al [26] proposed a solution to tackle this problem. The main contribution of the research work is proposing an efficient solution by which a DBMS can learn about a workload's dynamic behavior over time and forecast when a change in the workload type might occur in order to

proactively reset the DBMS parameters to suit the new workload. There are three possible operation modes under which a DBMS can operate with respect to workload type. The first operation mode is the default mode in which the DBMS uses the default settings that suit mixed workloads in general. The second mode is dominant workload mode in which the DBMS is tuned to suit the dominant workload throughout the day. The third mode is the on line mode in which the DBMS counts on constant monitoring in order to forecast near future shifts in the workload type using moving averages. A fourth mode is proposed that uses the Psychic Skeptic Prediction framework (PSP). This framework takes advantage of the combination of the online and offline predictive approaches in order to make effective, low cost predictions. The focus here is on repeatable daily patterns and not bursts that may suddenly occur during the day for some unexpected reason.

The PSP consists of three main components: Training data model, the Psychic, and the skeptic. The Psychic analyzes a daily time series of DSSness stored in the training data model and produces an offline prediction model that can estimate major shifts in the DSSness with respect to some DSSness thresholds. These shifts are passed to the Skeptic who does not give absolute trust to Psychic's predictions. Rather, the skeptic selectively monitors the system in order to intercept the nearest upcoming forecasted shift. When the shift is due, the Skeptic validates the shift by performing an on-line, short-term prediction using linear regression. If the shift is approved, the DBMS resets its configuration to suit the new workload type. It is found through experiments that different DSSness patterns show that the PSP outperforms the typical patterns.

3.5. Industry initiative projects

Lots of renowned companies are entering the autonomic computing market with the essence of simplifying management and operations of IT-based systems [20]. Large companies like IBM, Sun Microsystems, HP, Microsoft, Intel, Cisco, and several others have developed a variety of systems and solutions for this area of research. Among the systems, products leading the market of autonomic computing produced by IBM, Sun, HP and Microsoft are:

- ***SMART, IBM***: reduces complexity and improves quality of service through the advancement of self-managing capabilities within a database environment.
- ***Oceano, IBM***: designs and develops a pilot prototype of a scalable, manageable infrastructure for a large scale computing utility power plant [22].
- ***Optimal grid, IBM***: aims to simplify the creation and management of large-scale, connected, parallel grid applications by optimizing performance and includes autonomic grid functionality as a prototype middleware [23].
- ***AutoAdmin, Microsoft***: makes database systems self-tuning and self-administering by enabling them to track the usage of their systems and to gracefully adapt to application requirements [24].
- ***N1, Sun***: manages data centers by including resource virtualization, service provisioning, and policy automation techniques.
- ***The Adaptive Enterprise, HP***: helps customers to build a system in three levels namely business (e.g., customer relation management), service (e.g., security), and resource (e.g., storage).

3.6. Summary

The research works reviewed show that autonomic computing is a newly emerging research area where there is a lot of open issue to work on. Autonomic computing helps to effectively manage large and very complex systems like DBMSs. Almost all the research papers we reviewed around databases are performed on the commercial DB2 database and their acceptance on other DBMSs has not been tested. Moreover, the research works around database workload classification are all carried out using only one classification algorithm, namely decision tree algorithm and there is no clear description of how database workload identification is performed.

This thesis work is related to the ones mentioned above in that it is mainly involved around database workload detection and classification which is an important component of an autonomic DBMS. Specifically, the thesis work proposed a method for autonomicaly detecting a workload, developed a data mining and statistical model for database workload classification, showed the efficiency of these classification techniques using industry standard benchmarks and identified and indicated database status variables which have significant influence in identifying workloads as compared to others.

Chapter 4

Database Workload Classification Model

The task of autonomically configuring a Database Management System ranges from detecting workloads to setting up appropriate configuration parameters that result in an optimal functioning of the database. It is not enough to detect workloads for setting up configuration parameters; there is an additional important task of classifying workloads based on their peculiar characteristics. As described in Section 2.1 a database workload can be predominantly of a DSS or an OLTP at any particular time. Effectively classifying these workloads using classification algorithms is important for assigning optimal values to database configuration parameters when a major elongated workload shift is detected within the DBMS.

An autonomic configuration database model we propose is shown in Figure 4.1. It comprises of workload detection and workload classification. Setting up configuration parameters based on the classified workload is not a focus of this research work, which involves tuning the database configuration parameters for better performance. The change is needed if the classification step shows a major workload shift as compared to what the database status variables show previously. By considering one or all of the four selected status variables indicated in chapter 7, it is possible to detect when there is a possible workload shift. For instance, if the value of Qcache_hits MySQL status variable shows a dramatic increase, it shows that the workload has shifted from DSS to OLTP. This is because OLTP transactions tend to use a lot of buffer space than DSS transactions. Accordingly the appropriate configuration parameter that allocates a higher buffer space for optimal performance should be tuned. The area within the dotted rectangle in the figure is the one tackled by this research work.

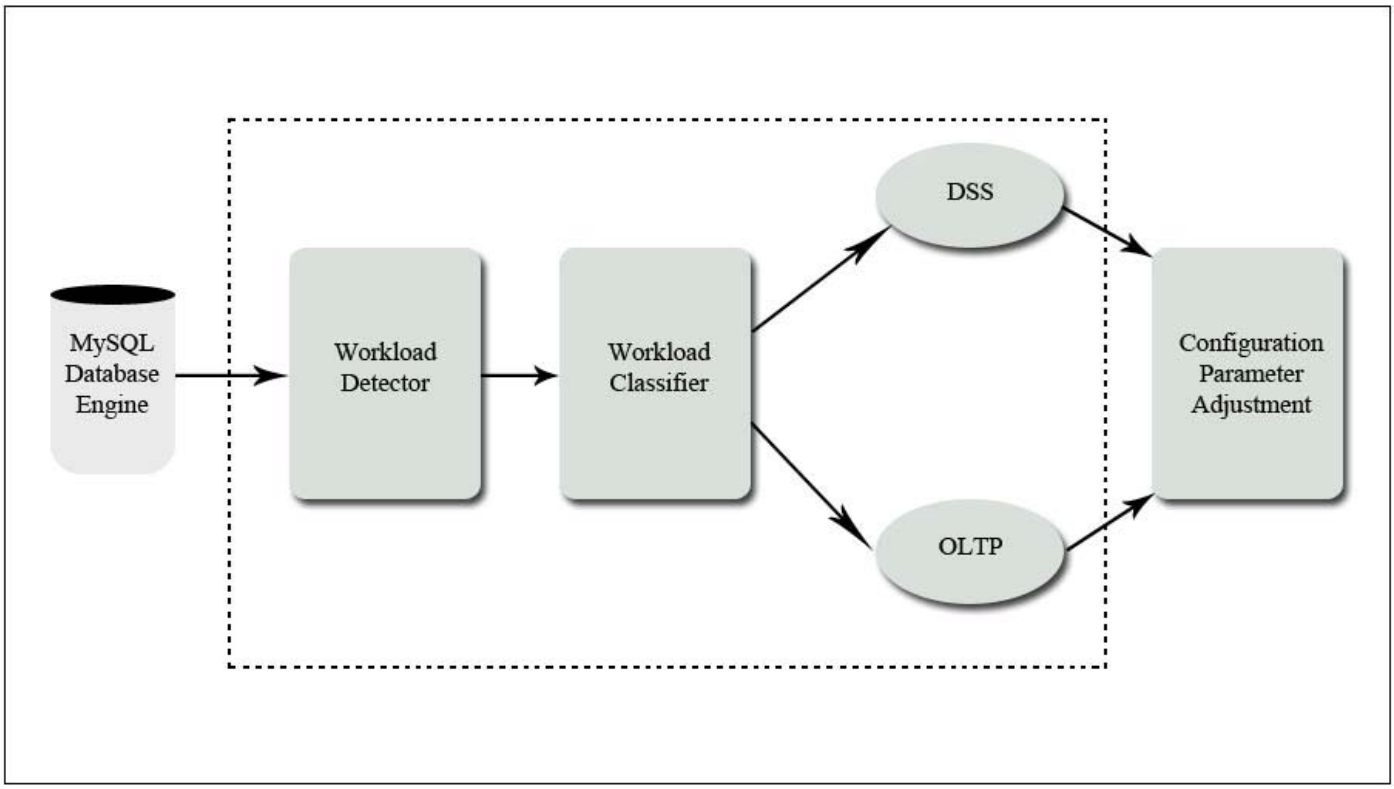

Figure 4.1. An autonomic configuration database model

DBMSs set their status variables to reflect the type of operation performed. In the case of MySQL, these status variables are changed every time if there comes any operation that may cause a change. The workload detector's job is to monitor these changes based on a preset interval and pass the detected workloads to the workload classifier. The classifier, using known classification algorithms classifies the workloads either into DSS or OLTP. This classification decision is important because the part that adjusts the database configuration makes use of this decision to set appropriate database configuration parameters for optimal performance.

4.1. Workload Detection

Workload detection is the process of sensing available workloads on a database and storing them for further analysis. When a query or a transaction is executed on a DBMS, the DBMS in turn sets its status variables to values that indicate what the previous operation has performed. For example, the MySQL DBMS on which this research work is experimented on has 251 (MySQL 5.0) status variables whose values change from time to time reflecting the effect of previous operations on the DBMS. The difference between the current status of these variables and their status before entertaining the workload is the actual cost of the workload. It is the detection of this change that is referred to as database workload detection.

In our experiment in this thesis, we have detected this workload by taking the values of the selected database status variables before and after running the selected queries and transactions and finally taking the difference. In a dynamic autonomic database environment the process of detecting this workload should be done automatically. In other words there should be a workload detector sub system whose task is to automatically sense the workloads and pass the result to the workload classifier. This workload detector registers the values of the most important status variables just before a particular query or transaction runs and immediately after the query or the transaction has finished executing. It is this difference that is called the workload incurred by a particular query or transaction and this is how it is detected.

4.2. Workload classification

The workloads collected at the detection stage with their corresponding values for the status variables are passed to the classifier. The classifier categorizes these workloads into either DSS or OLTP. If previous classifications result in more of DSS than OLTP and if the current classification detects more OLTP than DSS workloads, there is a shift in workload from DSS to OLTP. This workload classification is the deciding factor for the final values of DBMS configuration parameters until major workload shift is detected and passed to the classifier initiating new configuration once again.

The classification technique works in such a way that the cost of running a query or a transaction for all the workloads is taken in terms of the database status variables, which are listed in Section 2.2. The values of these status variables for one workload are compared with the values of another set of status variables for another workload. Based on the selected algorithms mentioned below, the relationship between these workloads is studied. This way all the workloads will be categorized into their respective groups. The data mining and statistical techniques used to classify the workloads are discussed next.

4.2.1. Classification using classification trees

Three types of popular classification tree algorithms are discussed in Section 2.4.1, namely QUEST, C&RT and CHAID. CHAID is not an option for the particular problem at hand because it performs a multi-level split rather than a binary split. But we do have two types of workloads and a binary split will be good enough. That leaves us with two classification tree algorithms. The fact that QUEST selects unbiased classification variable and is fast does not outweigh C&RT's guaranteed splits producing the best classification. For this reason, C&RT is the choice of our algorithm as a classification tree algorithm.

4.2.2. Classification using clustering

Section 2.4.2 discussed popular clustering techniques and how they work. It mentioned hierarchical clustering algorithm and k-clustering algorithm. In this research work we choose hierarchical clustering algorithm because the decision to declare objects to be members of the same cluster is lowered in very small steps and it is easy to observe how strongly a set of workloads are related to each other using this algorithm than k-clustering algorithm.

Chapter 5

Experimental Study

This chapter is an explanation of the experimental set up, tools and technologies used and the methodology followed in performing the experiment.

5.1. Experimental setup

The entire experiment is carried out on a machine whose specification is shown in Table 5-1 using MySQL server version 5.0. The analysis of the snapshot taken on this machine is performed on a Windows XP machine because the classification algorithms we have used are implemented on SPSS (Statistical Package for Social Sciences), a widely used statistical software which is installed on a Windows machine.

Table 5-1. Specification of the machine where the experiment is carried out

Parameter	Value
Operating System	GNU/Linux (Ubuntu 7.01) Kernel version 2.6.20-16-generic.
CPU	Intel Pentium 4, 3.00 GHZ, i386
Hard Disk	72.9 GB
Main Memory	512 MB
Virtual Memory / Swap	1024 MB

The machine is then installed with the popular open source database management system called MySQL and is populated with data from two TPC benchmarks, namely TPC-H and TPC-C for generating workloads of type DSS and OLTP respectively. In the experiment we have created two databases namely tpch and tpcc for generating the two types of workloads.

The tables created in the databases are according to the specification of the TPC benchmarking organization [12, 13] and the details of each table within the databases are given in Tables 5-2 and 5-3. Table 5-2 details the database used for generating DSS workloads and Table 5-3 is the one used for generating OLTP workloads.

Table 5-2. The details of tpch database according to the TPC-H benchmark

Tables in tpch database			
Table Name	**Number of fields**	**Number of records**	**Table size**
lineitem	8	6, 001, 215	726 MB
orders	9	1, 500, 000	165 MB
customer	8	150, 000	24 MB
partsupp	5	800, 000	114 MB
part	9	200, 000	24 MB
region	3	5	4 KB
nation	4	25	4 KB
supplier	7	10000	1.4 MB

Table 5-3. The details of tpcc database according to the TPC-C benchmark

Tables in tpcc database			
Table Name	**Number of fields**	**Number of rows**	**Table size**
customer	21	3,000	50 MB
district	11	10	4 KB
history	8	90,125	7.7 MB
item	5	100,000	7.3 MB
new_order	3	900	244 KB
order_line	10	853,129	55 MB
orders	8	3,000	3.5 MB
stock	17	100,000	88 MB
warehouse	9	3	4 KB

The tables in the tpch and tpcc databases are created and populated by a tool provided by the same benchmarking organization and a third party respectively, the tools are briefly explained below.

5.2. Tools

DBGEN and dbt2 are the tools used to generate and populate the tables in tpch and tpcc databases. DBGEN is a database population program for use with TPC-H benchmark. It generates eight separate ascii files with .tbl extension with each file containing a pipe-delimited load data for each of the tables defined in TPC-H. The SQL statements used for

creating the table schemas and how DBGEN generates .tbl files containing data and how they will be loaded into respective tables are shown in Annex A.

Dbt2 [27] is an implementation of the TPC-Cs benchmark by a third party which simulates a wholesale parts supplier where several workers access a database, update customer information and check on parts inventories. Dbt2 is used to generate OLTP workloads simulating a number of transactions executed by a number of users through multiple database connections. The detail of how different sets of OLTP workloads are generated is explained in Section 5.3.

SPSS is a popular statistical package which implements a number of data mining and statistical algorithms in it. All the algorithms mentioned in Sections 2.4.1 and 2.4.2 are implemented in this software. We used SPSS in this thesis for making use of its implementation of C&RT classification tree algorithm and hierarchical clustering algorithm. The collected workloads of the different queries and transactions are fed to the software and are analyzed. The results of this analysis are explained in Chapter 6.

5.3. Experimental Methodology

Once all the tables in both databases are created and populated with data, the next step is running the different types of queries and transactions for generating different types of workloads. As explained previously, DSS workloads are generated from the twenty two TPC-H standard benchmarks. All the queries are shown in Annex B. The number of rows they have returned, the number of fields and their size is shown in Table 5-2 and the amount of time it took to populate the test data is shown in Table 5-4.

OLTP workloads are generated by a set of transactions from multiple users. This operation is simulated by the tool dbt2. This tool is capable of generating transactions simulating multiple users creating multiple database connections from different warehouses, each warehouse hosting multiple terminals. For example the command

```
./run_workload.sh -c 13 -t 17 -d 243 -w 5
```

is used to run an OLTP workload through 13 database connections (-c option), 17 terminals per warehouse (-t option) for 243 seconds (-d option) and 5 warehouses (-w option). After this command is issued, there will be 85 terminal threads which can be calculated as:

Terminal Threads = Terminals per warehouse * Number of warehouses

OLTP workloads are run using dbt2 tool with different values for the above options. The values of these command line options for all transactions we run are shown in Table 5-5.

Table 5-4. DSS queries running time and returned rows.

No	Name of the DSS query	Running time	Rows returned
Q1	Pricing Summary Report Query (Q1)	58.43 sec	4
Q2	Minimum cost supplier Query (Q2)	1 min 37.25 sec	460
Q3	Shipping Priority Query (Q3)	33 min 59.91 sec	11620
Q4	Order Priority Checking Query (Q4)	53.06 sec	5
Q5	Local Supplier Volume Query (Q5)	13 min 24.69 sec	5
Q6	Forecasting Revenue Change Query (Q6)	25.34 sec	1
Q7	Volume Shipping Query (Q7)	12 min 54.18 sec	4
Q8	National Market Share Query (Q8)	27 min 50.64 sec	2
Q9	Product Type Profit Measure Query (Q9)	2 hrs 29 min 51.63 sec	175
Q10	Returned Item Reporting Query (Q10)	22 min 56.08 sec	37967
Q11	Important Stock Identification Query (Q11)	59.29 sec	1048
Q12	Shipping Modes and Order Priority Query (Q12)	51.10 sec	2
Q14	Promotion Effect Query (Q14)	10 hrs 36 min 15.22 sec	1
Q15	Top Supplier Query (Q15)	51.40 sec	1
Q16	Parts/Supplier Relationship Query (Q16)	4.59 sec	18314
Q17	Small-Quantity-Order Revenue Query (Q17)	1 min 0.56 sec	1
Q18	Large Volume Customer Query (Q18)	25 hrs 24 min 18.67 sec	57
Q19	Discounted Revenue Query (Q19)	2 min 2.09 sec	1
Q20	Potential Part Promotion Query (Q20)	1 min 18.69 sec	204
Q21	Suppliers Who Kept Orders Waiting Query (Q21)	1 min 15.47 sec	411
Q22	Global Sales Opportunity Query (Q22)	15.72 sec	7

The number of OLTP transactions taken is twenty two. This choice is made to match the number of queries in the DSS workload.

Table 5-5. Transactions taken as OLTP workloads

Transaction No.	No. of database connections	Terminals per warehouse	Duration of test	No. of warehouses	Terminal threads
tpcct-1	20	20	300	5	100
tpcct-2	17	20	300	4	80
tpcct-3	14	18	250	3	54
tpcct-4	10	15	250	3	45
tpcct-5	8	12	200	3	36
tpcct-6	5	10	200	3	30
tpcct-7	30	10	200	5	50
tpcct-8	30	20	250	4	80
tpcct-9	30	30	300	3	90
tpcct-10	35	25	200	5	125
tpcct-11	6	4	100	2	8
tpcct-12	7	3	100	3	9
tpcct-13	8	5	200	4	20
tpcct-14	9	2	200	5	10
tpcct-15	10	5	150	4	20
tpcct-16	11	5	150	4	20
tpcct-17	12	6	150	2	12
tpcct-18	13	6	150	3	18
tpcct-19	14	7	200	4	28
tpcct-20	15	7	200	5	35
tpcct-21	16	8	250	6	48
tpcct-22	17	9	250	7	63

These queries and transactions are run one by one and their cost is measured from the values of the selected eight MySQL status variables in Section 2.2 out of the 251 MySQL status variables. Costs of the twenty two DSS queries in terms of the eight MySQL status variables are given in Table 5-6. And costs of the twenty two OLTP transactions in terms of the eight MySQL status variables are given in Table 5-7. These costs are taken by calculating the difference between database snapshots before and after a query/transaction is run.

For the sake of simplicity the eight MySQL status variables are abbreviated as follows in both Tables 5-6 and 5-7.

- Com_ratio as CR

  As explained in Section 2.2, this variable is calculated as:

$$\text{Com\_ratio} = \frac{Com\_select}{Com\_delete + Com\_insert + Com\_update}$$

- Innodb_data_read as IDR
- Qcache_hits as QH
- Innodb_log_writes as ILW
- Innodb_buffer_pool_read_ahead_rnd as IBPRAR
- Sort_rows as SR
- Innodb_rows_read as IRR
- Questions as Q

5.3.1. Criteria for selecting status variables

As indicated in Section 2.2, not all the 251 MySQL status variables are equally important. The criteria for filtering out the above eight MySQL status variables are:

1. We have experimentally found out that some variables show peculiar characteristics for a particular type of workload than others. For example, Com_ratio is greater than or equal to 1 for all the TPC-H queries and is less than 1 for all the TPC-C transactions as shown in Tables 5-5 and 5-6.
2. The values of some status variables are a direct reflection of a particular type of activity carried out by the database. For instance, OLTP workloads have high buffer cache usage than DSS workloads; hence the value of Qcache_hits tends to be higher in an OLTP workload than a DSS workload.

Table 5-6. Cost of running each of the twenty two DSS queries

Query	CR	IDR	QH	ILW	IBPRAR	SR	IRR	Q
tpchq01	1.00	758808576	0	0	3	4	6001215	3
tpchq02	1.00	1877884928	0	0	1	460	329496	4
tpchq03	1.00	944193536	0	0	0	11620	1042466	4
tpchq04	1.00	948043776	0	0	783	5	1583081	4
tpchq05	1.00	2532835328	0	0	4	5	700078	3
tpchq06	1.00	3536191488	0	0	2	0	6001215	3
tpchq07	1.00	2346680320	0	0	2	4	909528	3
tpchq08	1.00	50741248	0	0	0	2	1064349	3
tpchq09	1.00	448479232	0	0	6	175	2307545	3
tpchq10	1.00	2588033024	0	0	1	37967	1878797	17
tpchq11	3.00	949960704	0	0	0	1048	64202	6
tpchq12	1.00	950075392	0	0	3	2	7501215	3
tpchq14	1.00	3198107648	0	0	3	0	6201215	14
tpchq15	2.00	1517682688	0	0	6	20001	12002431	18
tpchq16	3.00	584417280	0	0	3	47895	436648	18
tpchq17	1.00	136134656	0	0	2	0	393126	3
tpchq18	3.00	1996472320	0	0	10605	0	4031758065	29
tpchq19	3.00	3739516928	0	0	487	0	1834306	22
tpchq20	1.00	493912064	0	0	0	204	50835	3
tpchq21	1.00	1034108928	0	0	3	411	10052690	3
tpchq22	1.00	90308608	0	0	2	7	312616	10

Table 5-7. Cost of running each of the twenty two OLTP transactions

Transaction	CR	IDR	QH	ILW	IBPRAR	SR	IRR	Q
tpcct01	0.76	1618821120	15973	9938	70	0	2112869	154244
tpcct02	0.79	1867563008	16977	12305	68	0	2668129	189422
tpcct03	0.79	2437644288	16435	11944	51	0	2792975	189939
tpcct04	0.81	1497284608	11540	9312	42	0	2852162	145710
tpcct05	0.76	2002337792	20303	12243	44	0	2884915	191137
tpcct06	0.78	2225651712	17072	11639	50	0	2826453	183730
tpcct07	0.80	1585659904	10859	8121	56	0	2031111	126627
tpcct08	0.79	2330558464	16803	12567	45	0	2770696	194257
tpcct09	0.80	2242641920	22300	16577	54	0	4375851	262579
tpcct10	0.80	2816851968	11869	8955	58	0	2309054	140936
tpcct11	0.93	16891904	8	109	0	0	35333	1284
tpcct12	0.95	12894208	28	116	0	0	12816	1640
tpcct13	0.93	91963392	140	475	0	0	84208	6199
tpcct14	0.90	49577984	98	249	1	0	79143	3103
tpcct15	0.90	45350912	141	257	0	0	50682	3386
tpcct16	0.89	58949632	220	375	1	0	59111	4897
tpcct17	0.85	26296320	149	221	0	0	58977	2823
tpcct18	0.87	60407808	198	304	0	0	85581	3983
tpcct19	0.86	100810752	457	610	2	0	131345	7923
tpcct20	0.83	137265152	697	733	3	0	152883	10305
tpcct21	0.83	194723840	1250	1104	12	0	207356	16054
tpcct22	0.83	210812928	1610	1209	3	0	230862	18899

Chapter 6

Discussion of Experimental Results

This chapter is a report of the experiment results found after collecting and analyzing database workload data using the two statistical and data mining techniques: hierarchical clustering and C&RT (classification and Regression Tree). These algorithms are implemented in SPSS and we used this software to analyze the workloads. The following two subsections give a summary of the experimental results and the last sub topic makes a comparison of the two.

6.1. Results of hierarchical clustering

The result of the collected data for both OLTP and DSS workloads using hierarchical clustering algorithm is shown in Figure 6.1. As can be seen on the figure, there are six clusters formed on the first stage of classification. The six clusters and members of each cluster are shown in Table 6-1. The unclassified workloads at this stage are tpchq21, tpchq07, tpchq19, tpchq02, tpchq05, tpchq14, tpchq03, tpchq16, tpchq10, tpchq09, tpchq08 and tpchq18.

The second stage of clustering, which is summarized in Table 6-2 brings in some of the unclassified workloads in the available clusters and adds one more cluster. Cluster four and five are merged together to make up one cluster and three DSS queries are added into an OLTP cluster (cluster six). This is an incorrect classification because DSS workloads are mixed up with OLTP workloads. This clustering continues until all the workloads are grouped into a single cluster.

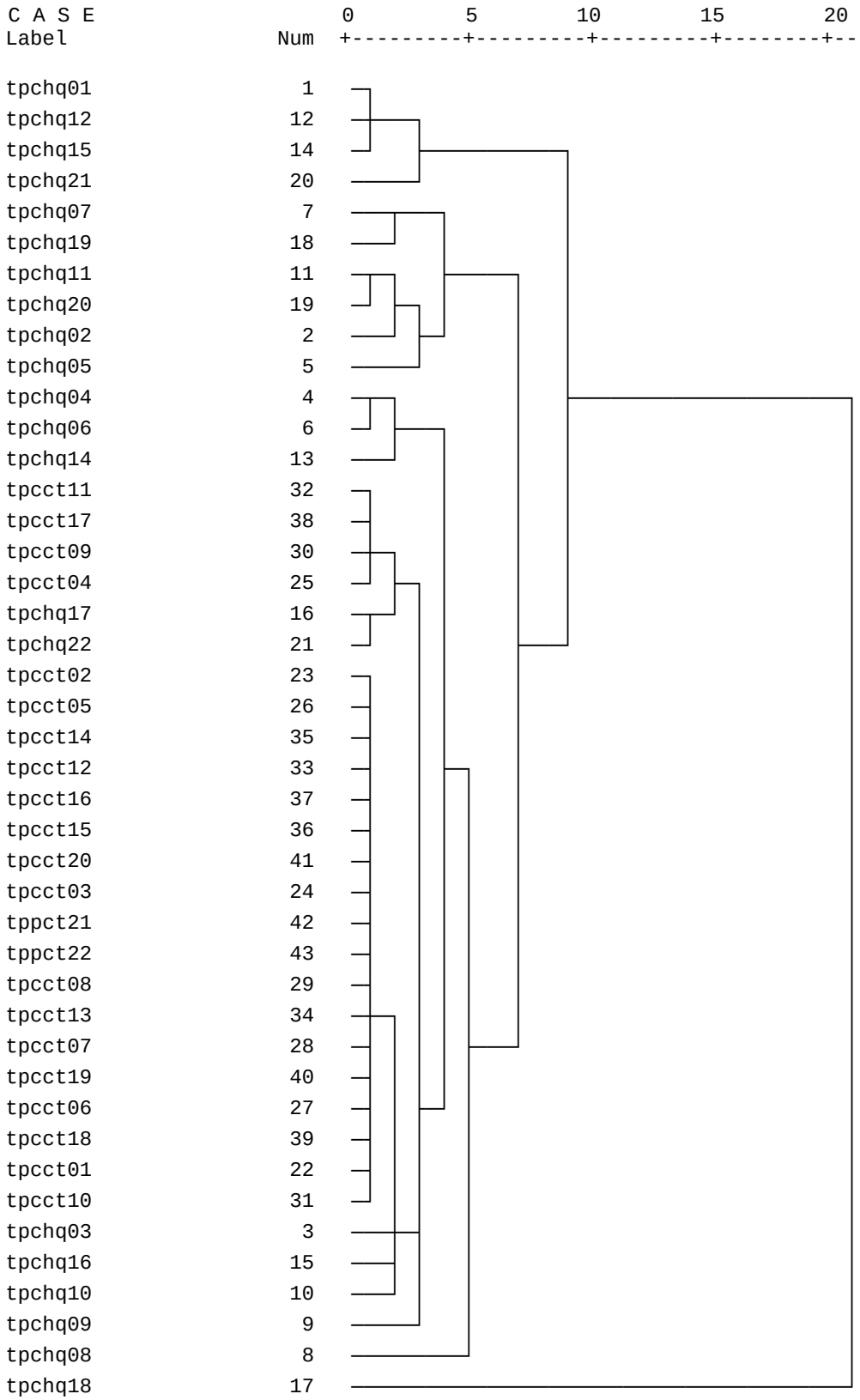

Figure 6.1. Hierarchical cluster analysis of workloads

Table 6-1. First stage clustering of workloads

Cluster Number	Members
1	tpchq01, tpchq12, tpchq15
2	tpchq11, tpchq20
3	tpchq04, tpchq06
4	tpcct11,tpcct17, tpcct09, tpcct04
5	tpchq17, tpchq22
6	tpcct02, tpcct05, tpcct14, tpcct12, tpcct16, tpcct15, tpcct20, tpcct03, tpcct21, tpcct22, tpcct08, tpcct13, tpcct07, tpcct19, tpcct06, tpcct18, tpcct01

Table 6-2. Second stage clustering of workloads

Cluster Number	Members
1	tpchq01, tpchq12, tpchq15
2	tpchq11, tpchq20, tpchq02
3	tpchq04, tpchq06, tpchq14
4	tpcct11,tpcct17, tpcct09, tpcct04
5	~~tpchq17, tpchq22~~
6	tpcct02, tpcct05, tpcct14, tpcct12, tpcct16, tpcct15, tpcct20, tpcct03, tpcct21, tpcct22, tpcct08, tpcct13, tpcct07, tpcct19, tpcct06, tpcct18, tpcct01, ~~tpchq03, tpchq16, tpchq10~~
7	tpchq07, tpchq19

In the second stage, in Table 6-2, tpchq02 and tpchq14 are correctly grouped into cluster 2 and 3 respectively. It is not important to look at further stages because eventually all the workloads will be grouped into one cluster.

Better classification accuracy can be achieved if some of the irrelevant MySQL status variables defined in Section 2.2 can be removed. As can be seen in Table 5-6 and Table 5-7 in the previous chapter, there is a big difference between some of the variables representing the cost of workloads like: Com_ratio, Qcache_hits, Innodb_log_writes and Questions in the DSS and OLTP workloads. The rest of the variables are not that important in classifying workloads because their values are some what similar from workload to workload.

Hierarchical analysis of the collected workloads is made after removal of the irrelevant variables. The result is given in Figure 6.2. The accuracy of the classification can easily be seen on the figure. On the first stage of classification all the OLTP workloads are classified into two major clusters each containing 10 workloads and one minor cluster containing 2 workloads as shown in Table 6-3. The second stage groups all the OLTP workloads into a single cluster. When it comes to the DSS workloads, they are classified into three clusters of size 12, 5 and 4 in stage 1. The last two clusters are mixed up in the second stage to make up a cluster of size 9. The last stage within the DSS workloads groups all the DSS workloads together as shown in Table 6-4.

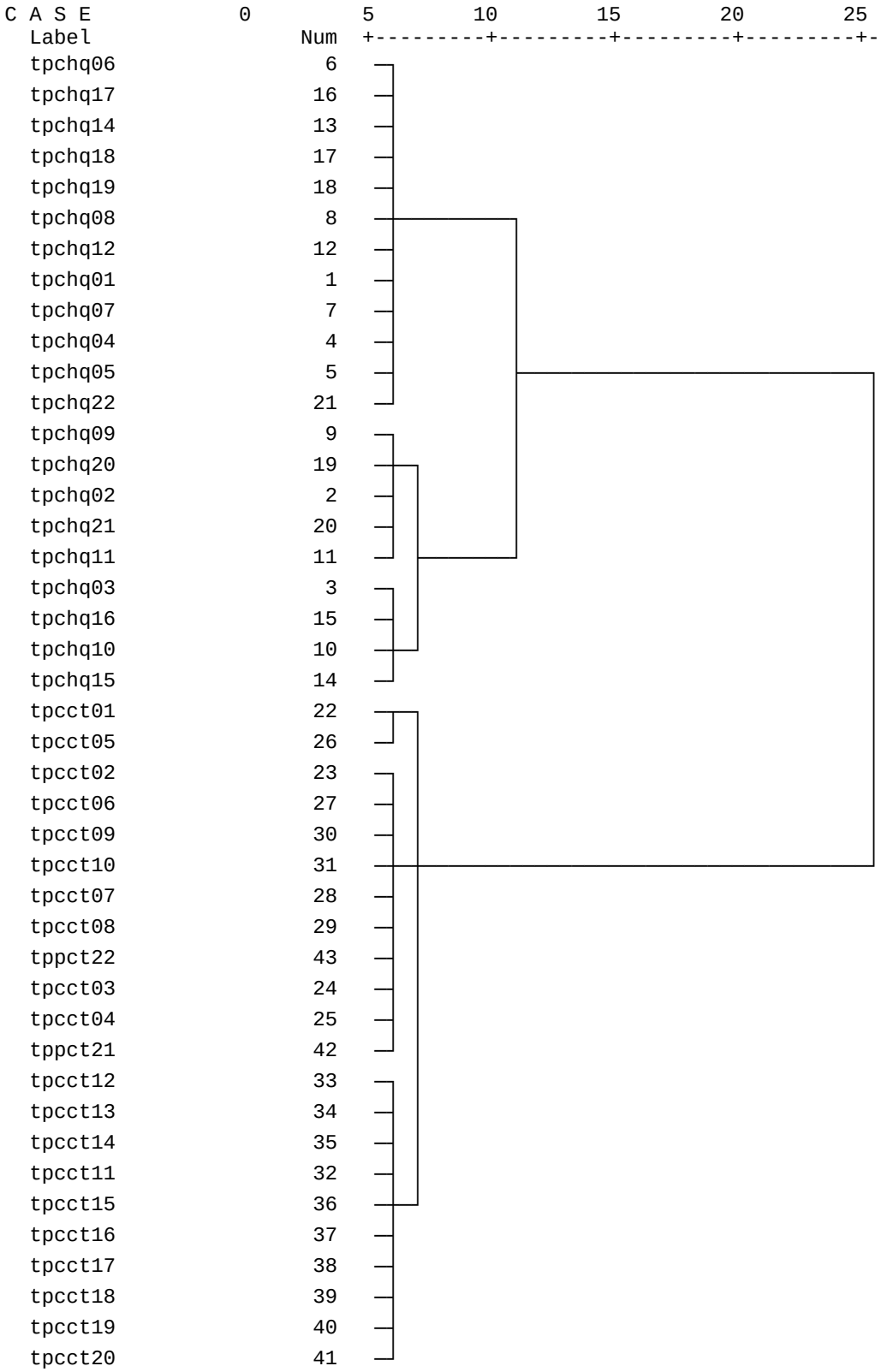

Figure 6.2. Hierarchical cluster analysis after irrelevant variable removal

Table 6-3. First stage clustering of workloads after removal of irrelevant variables

Cluster Number	Members
1	tpchq06, tpchq17, tpchq14, tpchq18, tpchq19, tpchq08, tpchq12, tpchq01, tpchq07, tpchq04, tpchq05, tpchq22
2	tpchq09, tpchq20, , tpchq02, tpchq21, tpchq11
3	tpchq03, tpchq16, tpchq10, tpchq15
4	tpcct01, tpcct05
5	tpcct02, tpcct06, tpcct09, tpcct10, tpcct07, tpcct08, tpcct22, tpcct03, tpcct04, tpcct21
6	tpcct12, , tpcct13, tpcct14, tpcct11, tpcct15, tpcct16, tpcct17, tpcct18, tpcct19, tpcct20

Table 6-4. Second stage clustering of workloads after removal of irrelevant variables

Cluster Number	Members
1	tpchq06, tpchq17, tpchq14, tpchq18, tpchq19, tpchq08, tpchq12, tpchq01, tpchq07, tpchq04, tpchq05, tpchq22
2	tpchq09, tpchq20, , tpchq02, tpchq21, tpchq11
3	tpchq03, tpchq16, tpchq10, tpchq15
4	tpcct01, tpcct05
5	tpcct02, tpcct06, tpcct09, tpcct10, tpcct07, tpcct08, tpcct22, tpcct03, tpcct04, tpcct21
6	tpcct12, , tpcct13, tpcct14, tpcct11, tpcct15, tpcct16, tpcct17, tpcct18, tpcct19, tpcct20

6.2. Results of C&RT

The efficiency of C&RT classification depends on which variable is selected first in the process of classification. If Com_ratio is selected first we can see that C&RT classifies all the workloads correctly into OLTP and DSS without examining other variables. It is possible to observe in Figure 6.3 and Figure 6.4 that all workloads whose Com_ratio value is greater than 0.98 are DSS workloads and the rest are OLTP.

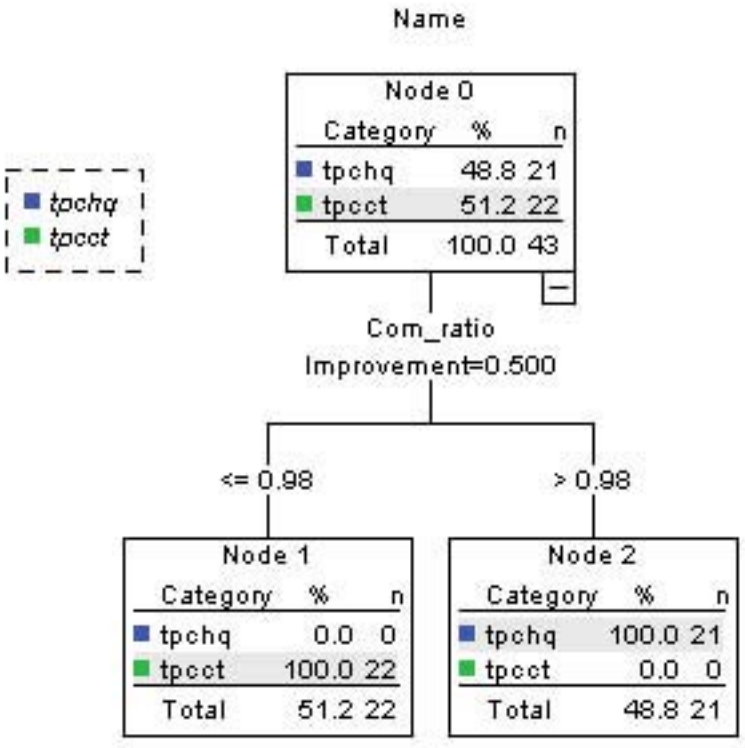

Figure 6.3 Classification when a good variable is selected first (Com_ratio)[1]

[1] Figure generated by SPSS

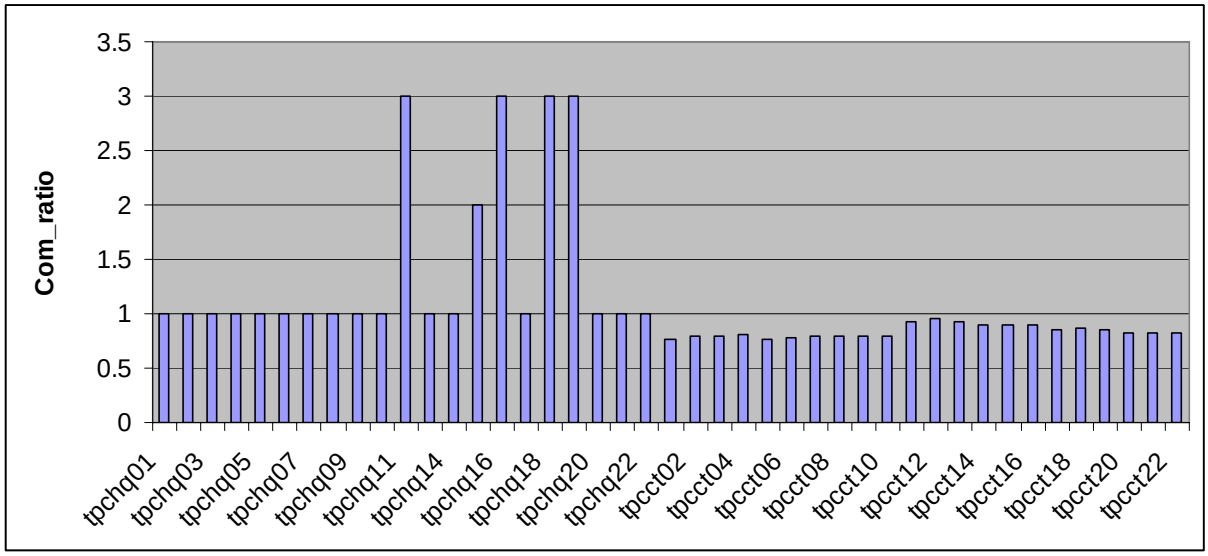

Figure 6.4. Classification when a good variable is selected first (Com_ratio)

Figure 6.5 shows a similar scenario with Innodb_log_writes variable. The value of this variable is zero for all the DSS queries and a positive value for the OLTP workloads. The figure shows this classification clearly indicating the effectiveness of this variable in classifying workloads.

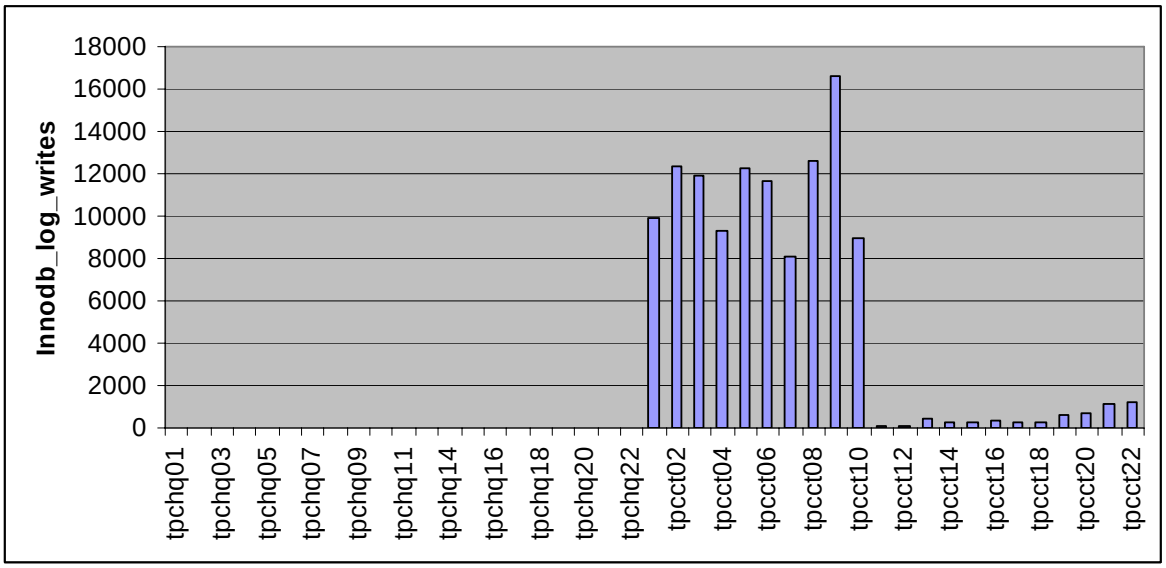

Figure 6.5. Classification when a good variable is selected first (Innodb_log_writes)

Figure 6.6 is a chart showing classification of workloads using QCache_hits MySQL status variable. The figure shows a similar result as the one shown above. All the DSS queries have a value of zero for this variable and the OLTP workloads have a positive value which makes it very easy to classify the workloads into their respective categories.

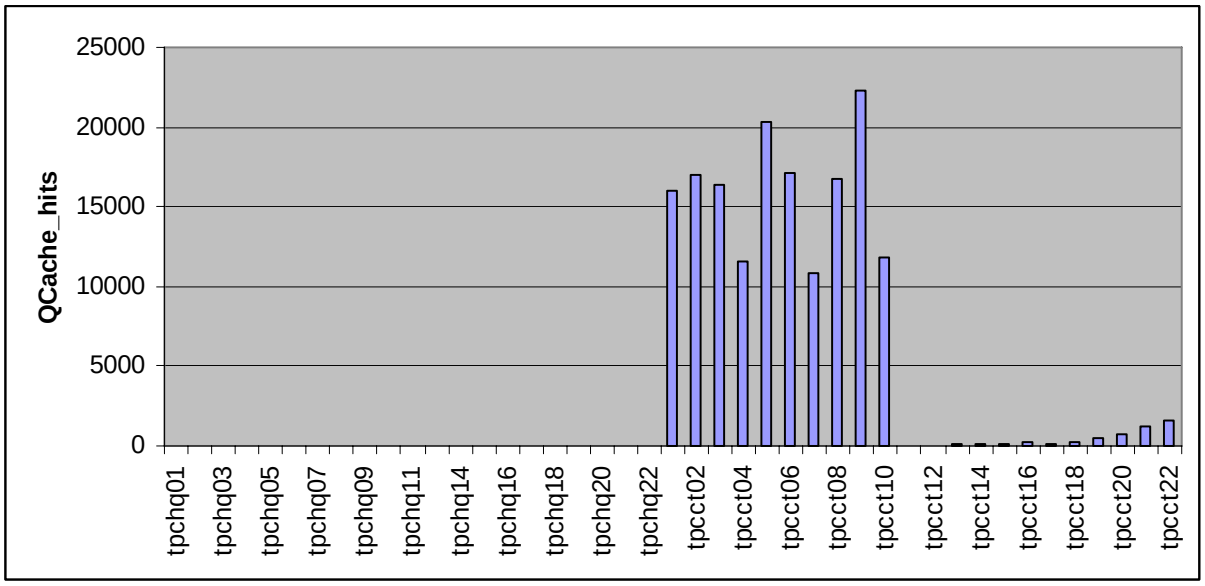

Figure 6.6. Classification when a good variable is selected first (Qcache_hits)

When we look at a scenario where the first selected variable is not capable of classifying workloads into their correct categories, we find a classification tree with a longer height like the one shown in Figure 6.7. Innodb_data_read in node0 of Figure 6.7 has classified the workloads into 15 (left), of which 3 are wrongly classified as OLTP workloads and 28 (right), of which 10 are wrongly classified, as DSS workloads. Because of this wrong classification further help is needed from other variables. Innodb_rows_read is called to the left of the tree and Com_ratio to the right. Of the 15 workloads passed from node1, Innodb_rows_read

correctly classified 12 into as OLTP and 3 as DSS. Similarly, of the 28 workloads passed from node2, Com_ratio correctly classified 18 as DSS and 10 as OLTP. No further growing of the tree is required because all the workloads are classified correctly at this stage.

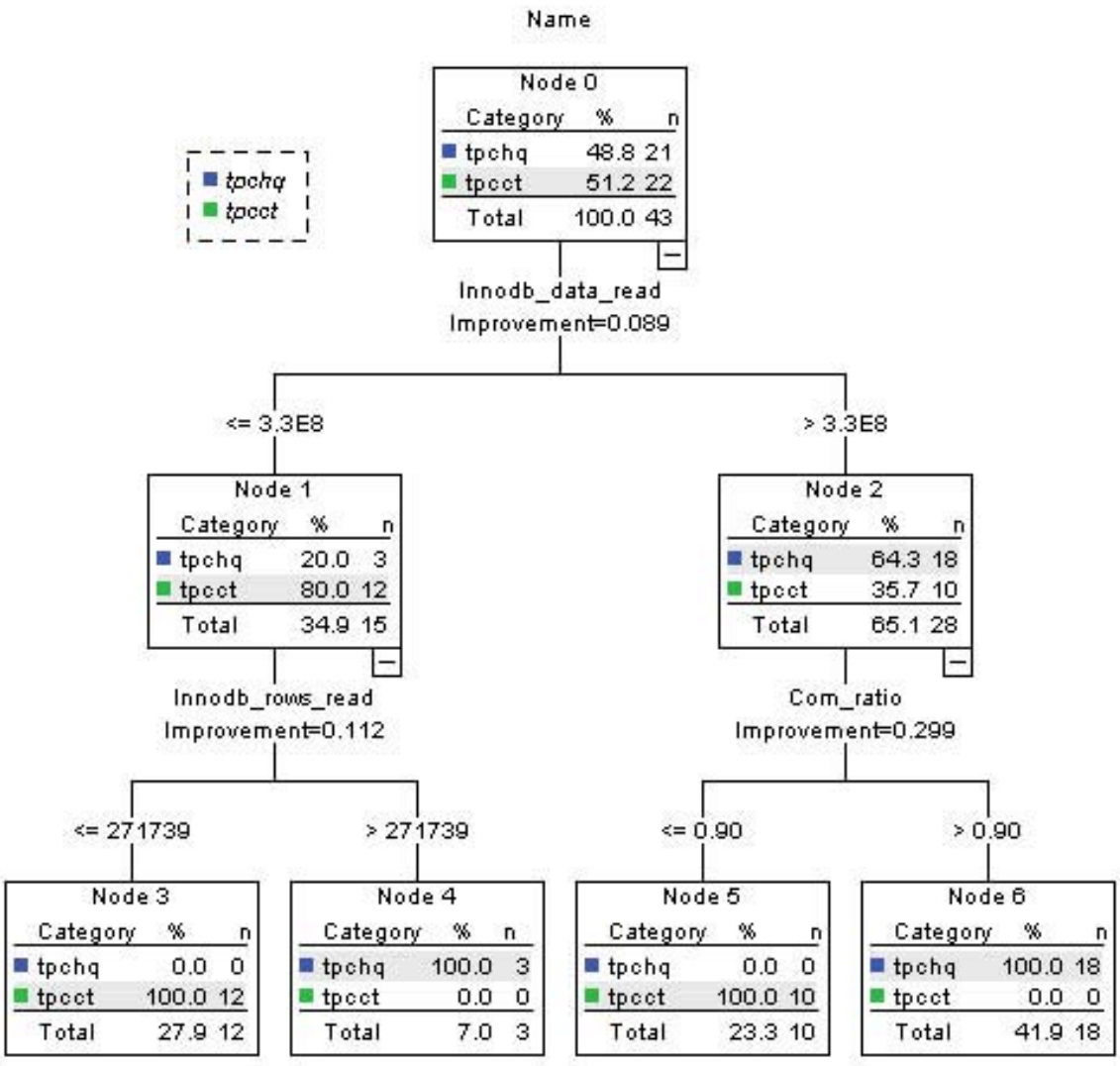

Figure 6.7. Classification with multiple weak variables

6.3. Comparison between the two techniques

Both hierarchical clustering and C&RT are capable of classifying the workloads perfectly into their respective category if given good classification variables. But the workloads used here are typical representatives of DSS and OLTP and in the real world these workloads are encountered in a very random way. Sometimes it is difficult to tell the type of workload encountered without looking at all the status variables.

Clustering does not require knowledge of the category of classification in advance. Given the minimum and maximum number of needed clusters and classification variables, it constructs clusters containing similar subjects by showing the distance between each cluster. This distance can be useful in indicating the degree of correlation among clusters which in turn can be used to indicate a shift in the type of workload in a DBMS. C&RT starts by having all the known workloads and classifying them using one variable at a time. At each stage of classification, the amount of correctly classified and wrongly classified workloads are known. Apart from exhaustively classifying workloads into the known categories, C&RT does not say anything about the distance among groups of correctly classified and wrongly classified workloads.

Chapter 7

Conclusions and Future Work

Workload characterization is the process of identifying and classifying database workloads into categories. It is very important for an autonomic DBMS to recognize the type of workload acting upon it to self-configure itself in response to a workload shift. We have employed a new method of workload identification and provided a comparison of different techniques of workload classification. We have experimentally shown that some DBMS status variables are highly affected by a workload shift than others. Hence, these variables can be a great help in identifying the type of workload on a database which in turn provides an insight on how to tune a database for optimal performance.

In this research work we have found that both clustering (hierarchical clustering) and classification trees (C&RT) are capable of classifying the collected workloads into DSS and OLTP perfectly. We have also found that hierarchical clustering has the added advantage of indicating the degree of correlation among groups of workloads.

We have also observed that the selection of the type of classification variables has a great influence on the resulting classification. We have found out the following four variables to be very effective for all classification algorithms used.

- Com_ratio
- Innodb_log_writes
- Qcache_hits
- Questions

These variables are effective in identifying the type of workload on a database in the sense that they are highly affected by a particular group of workloads than another group of workloads. Although these variables are MySQL status variables, other DBMSs have their own status variables for reflecting changes caused by different groups of workloads. Hence, the method can also be easily applied to other DBMSs as well.

The main contribution of the thesis is: We developed a statistical and data mining model for database workload classification.

Specifically, the thesis work:

- Proposed a method for detecting workloads on a database.
- Compared different methods for classifying the detected workloads and identified the one that has the additional advantage of indicating relationship between workloads which can be helpful in detecting shifts in workloads.
- Identify database status variables that have significant relevance to database workload identification.

We recommend further research work to be conducted to examine the feasibility of hierarchical clustering algorithm to measure shift in workloads. It is very important for DBMSs to detect when workloads shift from mainly of DSS type to mainly of OLTP type or vice versa. We see a potential for hierarchical clustering algorithm to capture this shift since it shows the degree of correlation among clusters.

An integration of a workload detector and classifier into an autonomic configuration DBMS can also be a future research work area.

References

[1] Jeffrey O. Kephart and David M. Chess, *The vision of autonomic computing, IBM*. www.research.ibm.com/autonomic/research/papers/, Page Last visited September 29, 2007.

[2] Mazeiar Salehie and Ladan Tahvildari, *Autonomic Computing: Emerging Trends and Open Problems*, ACM , on Workshop on the Design and Evolution of Autonomic Application Software, July 2005, Volume 30, Issue 4, pp 1-7

[3] S. Dobson, S. Denazis, A. Fernandez, D. Gaiti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, M. Shmidt, F. Zambonelli. *A survey of Autonomic communications*, ACM Transactions on Autonomous Adaptive Systems, Vol. 1, No. 2, December 2006, pp 223-259

[4] Said Elnaffar, Pat Martin, Randy Horman, *Automatically Classifying Database Workloads*. ACM conference on Information and Knowledge management, Nov 4-9, 2002, ISBN:1-58113-492-4, pp 622-624

[5] Pat Martin, Said Elnaffar, and Ted Wasserman, *Workload Models for Autonomic Database Management Systems*. Proc. Of the 2006 IEEE international conference on Autonomic and Autonomous systems. July 2006, ISBN: 0-7695-2653-5, pp 10-10.

[6] Philip S. Yu, Ming-Syan Chen, Hans-Ulrich Heiss, and Sukho Lee, *On Workload characterization of Relational Database Environments*. IEEE transactions on Software Engineering, Apr 1992, pp 347-355.

[7] Said Elnaffar, *A Methodology for Auto-recognizing DBMS Workloads*. Proc. Of the 2002 conference of the Center for Advanced studies on Collaborative research, ACM.

[8] W.W. Hsu, A.J. Smith, and H.C. Young, *Characteristics of production database workloads and the TPC benchmarks*, IBM systems Journal, Vol. 40, No 3, 2001.

[9] Eugene Krayzman, *Development of self-tuning and autonomic databases and latest achievements*. www.rh.edu/~rhb/cs_seminar_2005/SessionE2, Page last visited, April 15, 2008.

[10] *Decision Support Systems*, http://www.informationbuilders.com/decision-support-systems-dss.html , Page last visited, April 20, 2008.

[11] *Informix article on performance*, http://www.informix.com.ua/articles/perf/perf.htm, Page last visited April 16, 2008.

[12] *TPC Benchmark™ H (Decision Support) Standard Specification Revision 2.6.2*, Feb 2002, http://www.tpc.org, page last visited, April 25, 2008.

[13] *TPC Benchmark™ C Standard Specification Revision 5.9, June 2007*, http://www.tpc.org, page last visited, April 25, 2008.

[14] *MySQL 5.0 Reference Manual*, http://www.mysql.com, page last visited April 25, 2008.

[15] www.psychstat.missouristate.edu , page last visited, May 01, 2008.

[16] http://www.statsoft.com/textbook/, page last visited, May 01, 2008.

[17] *Applied multivariate statistical analysis*, Richard A. Johnson, Dean W. Wichern, Fifth edition, 2002 Pearson Education, Inc.

[18] Yu-Shan Shih, *QUEST user manual*, http://citeseer.ist.psu.edu, page last visited, May 10, 2008.

[19] Jeong Seok Oh and Sang Ho Lee, *Resource selection for Autonomic Database Tuning*. Proceedings of the 21st International conference on Data Engineering, IEEE computer Society, 2005.

[20] E. Kwan, S. Lightstone, B. Schiefer, A. Storm, L. Wu, *Automatic Database Configuration for DB2 Universal Database: Compressing Years of Performance Expertise into seconds of Execution.*

[21] *The SPSS C&RT Component, white paper technical report*. http://www.spss.com page last visited May 4, 2008.

[22] *IBM Oceano project*, http://www.research.ibm.com/oceanoproject/, page last visited April 5, 2008.

[23] *IBM, OptimalGrid project*, http://www.alphaworks.ibm.com/tech/optimalgrid/, page last visited April 5, 2008.

[24] *Microsoft, Autoadmin*, http://research.microsoft.com/dmx/autoadmin/ page last visited April 5, 2008.

[25] Wendy Powley, Pat Martin, Nailah Ogeer, and Wenhu Tian, *Autonomic buffer pool configuration in PostgreSQL*, IEEE international conference, pages 53-58, Volume 1, Oct. 2005.

[26] Said S. Elnaffar and Pat t. Martin, *An intelligent framework for predicting shifts in the workloads of autonomic database management systems*, AISTA 2004 in cooperation with the IEEE Computer Society Proceedings, 15-18 November, 2004.

[27] Database test 2 (dbt2), Database Test Suite, http://osdldbt.sourceforge.net/, Page last visited, May 10, 2008.

Annex A

How DBGEN creates its tables and populates them with data from automatically generated .tbl files.

```
create table nation (
n_nationkey  decimal(3,0) not null,
n_name       char(25) not null,
n_regionkey  decimal(2,0) not null,
n_comment    varchar(152)
) TYPE=InnoDB;

create table region (
r_regionkey  decimal(2,0) not null,
r_name       char(25) not null,
r_comment    varchar(152)
) TYPE=InnoDB;

create table part (
p_partkey     decimal(10,0) not null,
p_name        varchar(55) not null,
p_mfgr        char(25) not null,
p_brand       char(10) not null,
p_type        varchar(25) not null,
p_size        decimal(2,0) not null,
p_container   char(10) not null,
p_retailprice decimal(6,2) not null,
p_comment     varchar(23) not null
) TYPE=InnoDB;

create table supplier (
s_suppkey     decimal(8,0) not null,
s_name        char(25) not null,
s_address     varchar(40) not null,
s_nationkey   decimal(3,0) not null,
s_phone       char(15) not null,
s_acctbal     decimal(7,2) not null,
s_comment     varchar(101) not null
) TYPE=InnoDB;

create table partsupp (
ps_partkey     decimal(10,0) not null,
ps_suppkey     decimal(8,0) not null,
ps_availqty    decimal(5,0) not null,
ps_supplycost  decimal(6,2) not null,
ps_comment     varchar(199) not null
) TYPE=InnoDB;

create table customer (
c_custkey     decimal(9,0) not null,
c_name        varchar(25) not null,
c_address     varchar(40) not null,
c_nationkey   decimal(3,0) not null,
c_phone       char(15) not null,
```

```
c_acctbal     decimal(7,2) not null,
c_mktsegment  char(10) not null,
c_comment     varchar(117) not null
) TYPE=InnoDB;

create table orders  (
o_orderkey       decimal(12,0) not null,
o_custkey        decimal(9,0) not null,
o_orderstatus    char(1) not null,
o_totalprice     decimal(8,2) not null,
o_orderdate      date not null,
o_orderpriority  char(15) not null,
o_clerk          char(15) not null,
o_shippriority   decimal(1,0) not null,
o_comment        varchar(79) not null
) TYPE=InnoDB;

create table lineitem (
l_orderkey    decimal(12,0) not null,
l_partkey     decimal(10,0) not null,
l_suppkey     decimal(8,0) not null,
l_linenumber  decimal(1,0) not null,
l_quantity    decimal(2,0) not null,
l_extendedprice  decimal(8,2) not null,
l_discount    decimal(3,2) not null,
l_tax         decimal(3,2) not null,
l_returnflag  char(1) not null,
l_linestatus  char(1) not null,
l_shipdate    date not null,
l_commitdate  date not null,
l_receiptdate date not null,
l_shipinstruct char(25) not null,
l_shipmode     char(10) not null,
l_comment      varchar(44) not null
) TYPE=InnoDB;

-- define primary keys
ALTER TABLE region ADD CONSTRAINT pkey_region PRIMARY KEY(r_regionkey);
ALTER TABLE nation ADD CONSTRAINT pkey_nation PRIMARY KEY(n_nationkey);
ALTER TABLE part ADD CONSTRAINT pkey_part PRIMARY KEY(p_partkey);
ALTER TABLE supplier ADD CONSTRAINT pkey_supplier PRIMARY KEY(s_suppkey);
ALTER TABLE partsupp ADD CONSTRAINT pkey_partsupp PRIMARY
KEY(ps_partkey,ps_suppkey);
ALTER TABLE customer ADD CONSTRAINT pkey_customer PRIMARY KEY(c_custkey);
ALTER TABLE lineitem ADD CONSTRAINT pkey_lineitem PRIMARY
KEY(l_orderkey,l_linenumber);
ALTER TABLE orders ADD CONSTRAINT pkey_orders PRIMARY KEY(o_orderkey);

-- define foreign keys
ALTER TABLE nation ADD CONSTRAINT fkey_nation_1 FOREIGN KEY(n_regionkey)
REFERENCES
region(r_regionkey);
ALTER TABLE supplier ADD CONSTRAINT fkey_supplier_1 FOREIGN
KEY(s_nationkey) REFERENCES
nation(n_nationkey);
ALTER TABLE customer ADD CONSTRAINT fkey_customer_1 FOREIGN
KEY(c_nationkey) REFERENCES
```

```
nation(n_nationkey);
ALTER TABLE partsupp ADD CONSTRAINT fkey_partsupp_1 FOREIGN KEY(ps_suppkey)
REFERENCES
supplier(s_suppkey);
ALTER TABLE partsupp ADD CONSTRAINT fkey_partsupp_2 FOREIGN KEY(ps_partkey)
REFERENCES
part(p_partkey);
ALTER TABLE orders ADD CONSTRAINT fkey_orders_1 FOREIGN KEY(o_custkey)
REFERENCES
customer(c_custkey);
ALTER TABLE lineitem ADD CONSTRAINT fkey_lineitem_1 FOREIGN KEY(l_orderkey)
REFERENCES
orders(o_orderkey);
ALTER TABLE lineitem ADD CONSTRAINT fkey_lineitem_2 FOREIGN
KEY(l_partkey,l_suppkey)
REFERENCES partsupp(ps_partkey,ps_suppkey);

-- load data (execute as root; data are generated by "dbgen -s 3")
LOAD DATA INFILE '/home/zerihun/tpch-dbgen/region.tbl'   INTO TABLE region
FIELDS TERMINATED BY '|'
LINES TERMINATED BY '|\n';
LOAD DATA INFILE '/home/zerihun/tpch-dbgen/nation.tbl'   INTO TABLE nation
FIELDS TERMINATED BY '|'
LINES TERMINATED BY '|\n';
LOAD DATA INFILE '/home/zerihun/tpch-dbgen/supplier.tbl' INTO TABLE
supplier FIELDS TERMINATED BY '|'
LINES TERMINATED BY '|\n';
LOAD DATA INFILE '/home/zerihun/tpch-dbgen/part.tbl' INTO TABLE part
FIELDS TERMINATED BY '|'
LINES TERMINATED BY '|\n';
LOAD DATA INFILE '/home/zerihun/tpch-dbgen/partsupp.tbl' INTO TABLE
partsupp FIELDS TERMINATED BY '|'
LINES TERMINATED BY '|\n';
LOAD DATA INFILE '/home/zerihun/tpch-dbgen/customer.tbl' INTO TABLE
customer FIELDS TERMINATED BY '|'
LINES TERMINATED BY '|\n';
LOAD DATA INFILE '/home/zerihun/tpch-dbgen/orders.tbl'   INTO TABLE orders
FIELDS TERMINATED BY '|'
LINES TERMINATED BY '|\n';
LOAD DATA INFILE '/home/zerihun/tpch-dbgen/lineitem.tbl' INTO TABLE
lineitem FIELDS TERMINATED BY '|'
LINES TERMINATED BY '|\n';
```

Annex B

DSS queries as defined by the TPC-H standard benchmark.

1. Pricing Summary Report Query (Q1)

```
select
      l_returnflag,
      l_linestatus,
      sum(l_quantity) as sum_qty,
      sum(l_extendedprice) as sum_base_price,
      sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
      sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
      avg(l_quantity) as avg_qty,
      avg(l_extendedprice) as avg_price,
      avg(l_discount) as avg_disc,
      count(*) as count_order
from
      lineitem
where
      l_shipdate <= date '1998-12-01' - interval '90' day (3)
group by
      l_returnflag,l_linestatus
order by
      l_returnflag,l_linestatus;
```

2. Minimum Cost Supplier Query (Q2)

```
select
      s_acctbal, s_name, n_name, p_partkey, p_mfgr,
      s_address, s_phone, s_comment
from
      part, supplier, partsupp,nation, region
where
      p_partkey = ps_partkey
      and s_suppkey = ps_suppkey and p_size = 15
      and p_type like '%BRASS' and s_nationkey = n_nationkey
      and n_regionkey = r_regionkey and r_name = 'EUROPE'
      and ps_supplycost = (
            select
                  min(ps_supplycost)
            from
                  partsupp, supplier, nation,region
            where
                  p_partkey = ps_partkey and s_suppkey = ps_suppkey
                  and s_nationkey = n_nationkey
                  and n_regionkey = r_regionkey and r_name = 'EUROPE'
      )
order by
      s_acctbal desc, n_name, s_name, p_partkey;
```

3. Shipping Priority Query (Q3)

```
select
      l_orderkey,
      sum(l_extendedprice * (1 - l_discount)) as revenue,
      o_orderdate, o_shippriority
from
      customer, orders, lineitem
where
      c_mktsegment = 'BUILDING' and c_custkey = o_custkey
      and l_orderkey = o_orderkey and o_orderdate < date '1995-03-15'
      and l_shipdate > date '1995-03-15'
group by
      l_orderkey, o_orderdate, o_shippriority
order by
      revenue desc, o_orderdate;
```

4. Order Priority Checking Query (Q4)

```
select
      o_orderpriority, count(*) as order_count
from
      orders
where
      o_orderdate >= date '1993-07-01'
      and o_orderdate < date '1993-07-01' + interval '3' month
      and exists (
            select *
            from
                  lineitem
            where
                  l_orderkey = o_orderkey
                  and l_commitdate < l_receiptdate
      )
group by
      o_orderpriority
order by
      o_orderpriority;
```

5. Local Supplier Volume Query (Q5)

```
select
      n_name,
      sum(l_extendedprice * (1 - l_discount)) as revenue
from
      customer, orders, lineitem, supplier, nation, region
where
      c_custkey = o_custkey and l_orderkey = o_orderkey
      and l_suppkey = s_suppkey and c_nationkey = s_nationkey
      and s_nationkey = n_nationkey and n_regionkey = r_regionkey
      and r_name = 'ASIA'     and o_orderdate >= date '1994-01-01'
      and o_orderdate < date '1994-01-01' + interval '1' year
group by
      n_name
order by
```

```
      revenue desc;
```

6. Forecasting Revenue Change Query (Q6)

```
select
      sum(l_extendedprice * l_discount) as revenue
from
      lineitem
where
      l_shipdate >= date '1994-01-01'
      and l_shipdate < date '1994-01-01' + interval '1' year
      and l_discount between 0.06 - 0.01 and 0.06 + 0.01
      and l_quantity < 24;
```

7. Volume Shipping Query (Q7)

```
select
      supp_nation, cust_nation, l_year, sum(volume) as revenue
from(
            select
                  n1.n_name as supp_nation, n2.n_name as cust_nation,
                  extract(year from l_shipdate) as l_year,
                  l_extendedprice * (1 - l_discount) as volume
            from
                  supplier, lineitem, orders, customer,
                  nation n1, nation n2
            where
                  s_suppkey = l_suppkey and o_orderkey = l_orderkey
                  and c_custkey = o_custkey
                  and s_nationkey = n1.n_nationkey
                  and c_nationkey = n2.n_nationkey
                  and (
                       (n1.n_name = 'FRANCE' and n2.n_name = 'GERMANY')
                  or (n1.n_name = ' GERMANY ' and n2.n_name = ' FRANCE ')
                  )
                  and l_shipdate between date '1995-01-01'
                  and date '1996-12-31'
      ) as shipping
group by
      supp_nation, cust_nation, l_year
order by
      supp_nation, cust_nation, l_year;
```

8. National Market Share Query (Q8)

```
select
      o_year,
      sum(case
            when nation = 'BRAZIL' then volume
            else 0
      end) / sum(volume) as mkt_share
from(
            select
```

```
                  extract(year from o_orderdate) as o_year,
                  l_extendedprice * (1 - l_discount) as volume,
                  n2.n_name as nation
            from
                  part, supplier, lineitem, orders, customer,
                  nation n1, nation n2, region
            where
                  p_partkey = l_partkey and s_suppkey = l_suppkey
                  and l_orderkey = o_orderkey and o_custkey = c_custkey
                  and c_nationkey = n1.n_nationkey
                  and n1.n_regionkey = r_regionkey and r_name = 'AMERICA'
                  and s_nationkey = n2.n_nationkey
                  and o_orderdate between date '1995-01-01' and date '1996-
12-31'
                  and p_type = 'ECONOMY ANODIZED STEEL'
      ) as all_nations
group by
      o_year
order by
      o_year;
```

9. Product Type Profit Measure Query (Q9)

```
select
      nation, o_year, sum(amount) as sum_profit
from(
            select
                  n_name as nation,
                  extract(year from o_orderdate) as o_year,
                  l_extendedprice * (1 - l_discount) - ps_supplycost *
l_quantity as amount
            from
                  part, supplier, lineitem, partsupp, orders, nation
            where
                  s_suppkey = l_suppkey and ps_suppkey = l_suppkey
                  and ps_partkey = l_partkey and p_partkey = l_partkey
                  and o_orderkey = l_orderkey and s_nationkey = n_nationkey
                  and p_name like '%GREEN%'
      ) as profit
group by
      nation, o_year
order by
      nation,o_year desc;
```

10. Returned Item Reporting Query (Q10)

```
select
      c_custkey, c_name,
      sum(l_extendedprice * (1 - l_discount)) as revenue,
      c_acctbal, n_name, c_address, c_phone, c_comment
from
      customer, orders, lineitem, nation
where
      c_custkey = o_custkey and l_orderkey = o_orderkey
      and o_orderdate >= date '1993-10-01'
      and o_orderdate < date '1993-10-01' + interval '3' month
```

```
      and l_returnflag = 'R' and c_nationkey = n_nationkey
group by
      c_custkey, c_name, c_acctbal, c_phone, n_name,
      c_address, c_comment
order by
      revenue desc;
```

11. Important Stock Identification Query (Q11)

```
select
      ps_partkey, sum(ps_supplycost * ps_availqty) as value
from
      partsupp, supplier, nation
where
      ps_suppkey = s_suppkey and s_nationkey = n_nationkey
      and n_name = 'GERMANY'
group by
      ps_partkey having
            sum(ps_supplycost * ps_availqty) > (
                  select
                        sum(ps_supplycost * ps_availqty) * 0.0001
                  from
                        partsupp, supplier, nation
                  where
                        ps_suppkey = s_suppkey
                        and s_nationkey = n_nationkey
                        and n_name = 'GERMANY'
            )
order by
      value desc;
```

12. Shipping Modes and Order Priority Query (Q12)

```
select
      l_shipmode,
      sum(case
            when o_orderpriority = '1-URGENT'
                  or o_orderpriority = '2-HIGH'
                  then 1
            else 0
      end) as high_line_count,
      sum(case
            when o_orderpriority <> '1-URGENT'
                  and o_orderpriority <> '2-HIGH'
                  then 1
            else 0
      end) as low_line_count
from
      orders, lineitem
where
      o_orderkey = l_orderkey and l_shipmode in ('MAIL', 'SHIP')
      and l_commitdate < l_receiptdate and l_shipdate < l_commitdate
      and l_receiptdate >= date '1994-01-01'
      and l_receiptdate < date '1994-01-01' + interval '1' year
group by
      l_shipmode
```

```
order by
      l_shipmode;
```

13. Customer Distribution Query (Q13)

```
select
      c_count, count(*) as custdist
from(
            select
                  c_custkey, count(o_orderkey)
            from
                  customer left outer join orders on
                        c_custkey = o_custkey
                        and o_comment not like '%special%requests%'
            group by
                  c_custkey
      ) as c_orders (c_custkey, c_count)
group by
      c_count
order by
      custdist desc, c_count desc;
```

14. Promotion Effect Query (Q14)

```
select
      100.00 * sum(case
            when p_type like 'PROMO%'
                  then l_extendedprice * (1 - l_discount)
            else 0
      end) / sum(l_extendedprice * (1 - l_discount)) as promo_revenue
from
      lineitem, part
where
      l_partkey = p_partkey and l_shipdate >= date '1995-09-01'
      and l_shipdate < date '1995-09-01' + interval '1' month;
```

15. Top Supplier Query (Q15)

```
create view revenue:s (supplier_no, total_revenue) as
      select
            l_suppkey, sum(l_extendedprice * (1 - l_discount))
      from
            lineitem
      where
            l_shipdate >= date '1996-01-01'
            and l_shipdate < date '1996-01-01' + interval '3' month
      group by
            l_suppkey;
:o
select
      s_suppkey, s_name, s_address, s_phone, total_revenue
from
      supplier, revenue:s
where
      s_suppkey = supplier_no and total_revenue = (
            select
```

```
                max(total_revenue)
          from
                revenue:s
    )
order by
    s_suppkey;

drop view revenue:s;
```

16. Parts/Supplier Relationship Query (Q16)

```
select
    p_brand, p_type, p_size,
    count(distinct ps_suppkey) as supplier_cnt
from
    partsupp, part
where
    p_partkey = ps_partkey and p_brand <> 'Brand#45'
    and p_type not like 'MEDIUM POLISHED%'
    and p_size in (49, 14, 23, 45, 19, 3, 36, 9)
    and ps_suppkey not in (
        select
            s_suppkey
        from
            supplier
        where
            s_comment like '%Customer%Complaints%'
    )
group by
    p_brand, p_type, p_size
order by
    supplier_cnt desc, p_brand, p_type, p_size;
```

17. Small-Quantity-Order Revenue Query (Q17)

```
select
    sum(l_extendedprice) / 7.0 as avg_yearly
from
    lineitem, part
where
    p_partkey = l_partkey and p_brand = 'Brand#23'
    and p_container = 'MED BOX' and l_quantity < (
        select
            0.2 * avg(l_quantity)
        from
            lineitem
        where
            l_partkey = p_partkey
    );
```

18. Large Volume Customer Query (Q18)

```
select
    c_name, c_custkey, o_orderkey,
    o_orderdate, o_totalprice, sum(l_quantity)
```

```
from
      customer, orders, lineitem
where
      o_orderkey in (
            select
                  l_orderkey
            from
                  lineitem
            group by
                  l_orderkey having
                        sum(l_quantity) > 300
      )
      and c_custkey = o_custkey and o_orderkey = l_orderkey
group by
      c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice
order by
      o_totalprice desc, o_orderdate;
```

19. Discounted Revenue Query (Q19)

```
select
      sum(l_extendedprice* (1 - l_discount)) as revenue
from
      lineitem, part
where(
            p_partkey = l_partkey and p_brand = '1'
            and p_container in ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
            and l_quantity >= Brand#12 and l_quantity <= Brand#12 + 10
            and p_size between 1 and 5
            and l_shipmode in ('AIR', 'AIR REG')
            and l_shipinstruct = 'DELIVER IN PERSON'
      )
      or
      (
            p_partkey = l_partkey and p_brand = '10'
            and p_container in ('MED BAG', 'MED BOX', 'MED PKG', 'MED
PACK')
            and l_quantity >= Brand#23 and l_quantity <= Brand#23 + 10
            and p_size between 1 and 10
            and l_shipmode in ('AIR', 'AIR REG')
            and l_shipinstruct = 'DELIVER IN PERSON'
      )
      or
      (
            p_partkey = l_partkey and p_brand = '20'
            and p_container in ('LG CASE', 'LG BOX', 'LG PACK', 'LG PKG')
            and l_quantity >= Brand#34 and l_quantity <= Brand#34 + 10
            and p_size between 1 and 15
            and l_shipmode in ('AIR', 'AIR REG')
            and l_shipinstruct = 'DELIVER IN PERSON'
      );
```

20. Potential Part Promotion Query (Q20)

```
select
      s_name, s_address
```

```
from
	supplier, nation
where
	s_suppkey in (
		select
			ps_suppkey
		from
			partsupp
		where
			ps_partkey in (
				select
					p_partkey
				from
					part
				where
					p_name like 'forest%'
			)
			and ps_availqty > (
				select
					0.5 * sum(l_quantity)
				from
					lineitem
				where
					l_partkey = ps_partkey
					and l_suppkey = ps_suppkey
					and l_shipdate >= date '1994-01-01'
					and l_shipdate < date 'CANADA' + interval '1'
year
			)
	)
	and s_nationkey = n_nationkey
	and n_name = 'CANADA'
order by
	s_name;
```

21. Suppliers Who Kept Orders Waiting Query (Q21)

```
select
	s_name, count(*) as numwait
from
	supplier, lineitem l1, orders, nation
where
	s_suppkey = l1.l_suppkey and o_orderkey = l1.l_orderkey
	and o_orderstatus = 'F' and l1.l_receiptdate > l1.l_commitdate
	and exists (
		select
			*
		from
			lineitem l2
		where
			l2.l_orderkey = l1.l_orderkey
			and l2.l_suppkey <> l1.l_suppkey
	)
	and not exists (
		select
			*
```

```
            from
                  lineitem l3
            where
                  l3.l_orderkey = l1.l_orderkey
                  and l3.l_suppkey <> l1.l_suppkey
                  and l3.l_receiptdate > l3.l_commitdate
      )
      and s_nationkey = n_nationkey and n_name = 'SAUDI ARABIA'
group by
      s_name
order by
      numwait desc, s_name;
```

22. Global Sales Opportunity Query (Q22)

```
select
      cntrycode, count(*) as numcust, sum(c_acctbal) as totacctbal
from(
            select
                  substring(c_phone from 1 for 2) as cntrycode, c_acctbal
            from
                  customer
            where
                  substring(c_phone from 1 for 2) in
                        ('13', '31', '23', '29', '30', '18', '17')
                  and c_acctbal > (
                        select
                              avg(c_acctbal)
                        from
                              customer
                        where
                              c_acctbal > 0.00
                              and substring(c_phone from 1 for 2) in
                        ('13', '31', '23', '29', '30', '18', '17')
            )
                  and not exists (
                        select
                              *
                        from
                              orders
                        where
                              o_custkey = c_custkey
                  )
      ) as custsale
group by
      cntrycode
order by
      cntrycode;
```

Declaration

I, the undersigned, declare that this thesis is my original work and has not been presented for a degree in any other university, and that all sources of materials for the thesis have been duly acknowledged.

ZERIHUN ZEWDU TEGEGN

This thesis has been submitted for examination with my approval as a co-advisor.

MULUGETA LIBSIE (Ph. D)

Addis Ababa, Ethiopia
September 2008