Master's Thesis

Dynamical Modeling of MySQL Database Server

By

Shirish K.C.

Department of Electrical and Information Technology
Faculty of Engineering, LTH, Lund University
SE-221 00 Lund, Sweden

Abstract

This thesis basically consists of different experiments conducted at the Department of Electrical and Information Technology, Lund University, Sweden.

These experiments are conducted to quantify the dynamics of the mean response time of the requests in a MySQL database server. The dynamics are studied under various load (requests per second) and database characteristics such as relation size, database engine type and number of concurrent connections to the MySQL server.

Acknowledgement

I would like to express my deep gratitude to my professor Maria Kihl for providing me with the opportunity to work on such an interesting project. I am very thankful for her trust and flexibility that she provided me in this project.

Further, I would like to thank to my project supervisor Payam Amani for providing me necessary guidance and co-operation while accomplishing this project.

I also want to express my gratitude to Manfred Dellkrantz and Gustav Cedersjö for providing support and assistance during the entire project.

Finally I would like to thank all of my friends for helping me to achieve this goal.

Shirish K.C.

Table of Contents

1 Introduction

Resource management for computing systems nowadays is considered as the most important part for any computing system. The reason behind this is that poorly managed resources severely reduce the performance of the computing systems. The current trends allow enterprise services to be accessible through networks. Usually, communication between these services is done across IP-networks. As user traffic to the servers is unpredictable and gets surged more often, these servers experience external load disturbances. As a result, servers belonging to enterprises often become bottleneck while the network backbone may remain not fully utilized. Due to these facts, these servers must provide some kind of performance guarantees against uncertain external load which it experiences quite often. The guarantee to service performance may include agreement on services (on QoS, throughput etc) that the system provides on their clients. On overloaded condition, the system should able to maintain acceptable degradation on these server performances.

However, it is important for involved computing systems to have accurate performance models for finding better resource optimization methods. Usually, a high number of clients are involved in accessing database servers. Therefore, operating regions of these servers experience mainly the high traffic situations which also mean the computing systems follow non-linear dynamics. The accurate characteristic of this dynamics determine the performance of the servers. Here, load dynamics of the database servers show completely different characteristic in overloaded situation than in low load situation or in high load situation.

The common problem that database servers usually encounter is database overload. This happens when requests to the database server become temporarily peak. When the requests to the database server get peak, more requests arrive at the server than they are designed for. However, if this kind of requests peaks seldom occurs, then it is not economical to allocate additional server capacity to handle these peaks. This is due to the fact that many of the resources remain unused and just consumed for waiting for the situation where the load peaks become high. In addition, the need of more

processing power also increases the need of more hardware in the computing system.

Some kind of admission control can be implemented in order to control these peaks in the load to the server. The admission control basically rejects some requests in order to avoid server overload and thereby ensure that the server do not stop functioning because of the overload problem. Another solution is load balancing which is basically a kind of powerful resource management mechanism that should be implemented in order to distribute the need of resources uniformly among several resources sharing units.

The database servers are also considered as a major component in the future Internet systems especially in field of cloud computing and data centers. Considering these facts in mind, different models and characteristics of high load dynamics of databases is closely observed. This report illustrates models that are able to accurately represent the dynamic of mean response time of requests that arrive to the database servers.

In other words, the servers get overloaded when arrival rate of the requests from the clients exceed the server capacity. Therefore, the modeling of the database server is needed for providing admission control on user requests in order to avoid the server overload. Then, we can achieve the same performance of database server with no need to allocate additional server capacity.

2 Background and Related Work

2.1 Detailed architecture description

The whole system can be viewed as a multi-tier system with layered architecture.

The multi-tier architecture is often described in client-server architecture in which the data presentation, data application processing and managing database are logically different processes.

Each tier or layer is responsible for performing its own specific functions and forms a part of the system. In a layered architecture, processing of the requests starts from the topmost layer. The part which remains unprocessed is passed to the layer or tier below this layer. It can be viewed as each layer will get request from the tier just above and responds back for that request to that tier.

In case of a multi-tier web application, servers are distributed among different tiers. The number of tiers differs from system to system. However, there are some general principles that are common in most of the systems. The requests from the clients are usually handled by topmost tier which is client tier and the lowest tier is often a data tier with a database server.

Figure 2.1 Two-tier web server system

The client tier which is the web browser renders the web page and executes the client-side scripts of the web application. The generation and handling of dynamic content are performed in the application tier. Here, the programming can be developed for application servers to enable them to perform different functions for different requests. Some examples of application tier include Java EE, ASP.NET, PHP, ColdFusion platform.

When clients initiate requests for getting or updating some of the attribute values of the specified tuple inside the relation then those requests are sent to next tier, data tier.

The data tier comprises of a database management system where user requests are handled on the basis of need from the application server. One of the mostly used types of database server is the relational database server. In a relational database, the requests to the database are usually accompanied with SQL statement. The data in the relational database server is stored in a single relation or in several relations. Depending upon the types of requests from users, a request is actually delivered for getting data from relations or confirmation of updating or inserting or deleting of data in the relations.

2.2 Database Management System

2.2.1 Relational Model

The relational model is a famous model used for database management system. It was first introduced and proposed by Edgar F. Codd [1]. This model is a database model which is based on first-order predicate logic.

This model is intended to provide declarative methods to specify data and queries. It basically means that the users can directly state the information that database contains and information of the users is acquired from the database. The procedure behind this is relied on the database management system software which defines data structures for storing and retrieving user data.

2.2.1.1 Relational Database

The data in relational database is stored into distributed data sets or relations. However, data is matched in relational database using a common characteristic that lies within the data set.

The relational database uses some sort of software called relational database management system (RDBMS).

Some mathematical terms are implemented in the relational database theory. These terms are broadly equivalent to SQL database terminology.

Table 2.1 illustrates some of the mostly used relational database terms and their SQL database equivalents.

Relational terms	**SQL equivalents**
Relation, base relvar	Table
derived relvar	view, query result, result set
tuple	row
attribute	column

Table 2.1: Lists of terms and their equivalent used in relational database

A relation usually consists of a set of tuples where each of the tuples contains same attributes. A tuple generally denotes an object or information related to the object. Objects can be anything which may be something physical or concepts. A relation can also be described as a table consisting of rows and columns. The columns or attributes provides the list of data referenced to the same domain or heading and the type of data referenced which are conformed to same constraints.

There is no specific order how data is stored in the tuples or attributes. The applications can access data by different methods of querying of database. There are different queries types such as select query to acquire data from the database and insert, delete, update query to modify the relations of database. In any relations, there need to be some attribute to be uniquely defined in order to provide reference to each tuple of a relation. This can be carried out by defining the primary key.

2.2.2 Structural Query Language

SQL is an acronym for Structured Query Language and that is a standardized language designed for relational database management systems for retrieving, updating and managing data in relational databases. The design of original version called SEQUEL (*s*tructured *E*nglish query language) was accomplished in IBM research center in 1974 and 1975.The introduction of SQL as a commercial database system was started by Oracle Corporation in 1979 [2].

SQL is a type of declarative language where expected outcome or operation is performed without any details about how to perform the tasks.

The statements are used in order to command for certain type of operations or methods. This methods need to be carried out in certain instructions which contain specific SQL statement and additional parameters that are applied to that statement. All SQL statements and their modifiers should follow certain official SQL standards and certain extensions to those standards depending on the ways each specific database provider implements them. Commonly used statements are grouped into the following categories [2]:

Data Query Language (DQL)

- SELECT - Used to retrieve certain records from one or more tables.

Data Manipulation Language (DML)

- INSERT - Used to create a record.
- UPDATE - Used to change certain records.
- DELETE - Used to delete certain records.

Data Definition Language (DDL)

- CREATE - Used to create a new table, a view of a table, or other object in database.
- ALTER - Used to modify an existing database object, such as a table.
- DROP - Used to delete an entire table, a view of a table or other object in the database.

Data Control Language (DCL)

- GRANT - Used to give a privilege to someone.
- REVOKE - Used to take back privileges granted to someone.

2.2.3 Indexing

For the prompt searching of rows with specific column values, indexes are used. If there is no indexing, MySQL database need to read from the beginning starting from first row through the whole table to find relevant rows. This process is repeated for each search of a relevant row. This searching of the MySQL database takes long time if the size of table is large. However, if indexing is used for the columns, MySQL can quickly find the position of the data in the relevant rows without a requirement for search from the beginning to the end of the table. This indexing will significantly reduce the reading time compare to sequential read time without indexing.

The common MySQL indexes are defined by parameters like PRIMARY KEY, UNIQUE, INDEX, and FULLTEXT on certain columns.

2.2.4 MySQL

MySQL is an open source relational database management system (RDBMS) that runs as a server providing a very fast, robust SQL (Structured Query Language), multi-threaded, multi-user access to a number of databases. It was first developed by the Swedish company MySQL AB and now it is owned by Oracle Corporation [3]. MySQL is made available in both open source license and in commercial license.

The MySQL has a certain communication protocol which is used for authentication, querying and managing the server using a subset of the standard Structured Query Language (SQL) commands. The client libraries with other libraries which implement the protocol are written for JDBC (Java Database Connectivity) and for the .NET platforms. MySQL provides APIs for the C, C++, Eiffel, Java, Perl, PHP and Python languages. In addition, OLE DB and ODBC providers can also connect to MySQL in the Microsoft environment.

The MySQL database server contains several relations under each database and there can be many databases inside MySQL server. Nowadays, there are different types of database storage engines such as MyISAM Storage engine, InnoDB Storage engine, Merge Storage engine etc.

All relations contained inside MySQL database server in our experiment consists of InnoDB as default database storage engine.

2.2.4.1 InnoDB

InnoDB includes standard ACID-compliant transaction features which are capable of commit, rollback and crash-recovery. Therefore, user data is protected in InnoDB. The InnoDB is able to maintain the data integrity in entire query process. InnoDB involves row-level locking. With row-level locking features, it is possible for one query to update or read a row while, another query updates or reads a different row at the same time. Hence, multiuser concurrency and performance of database is enhanced. The user data is stored in clustered indexes so that I/O operations for common queries can be reduced. These queries are issued based on the primary keys.

InnoDB engine is selected in case of processing a large volume of data. The reason behind this is that it can be designed to provide maximum performance. InnoDB has been used in a famous Internet news site Slashdot.org, Mytrix, Inc. which stores more than 1TB of data.

The InnoDB became part of the Oracle Corporation in October 2005 after former company Innobase Oy has been acquired by Oracle Corporation [4].

In order to find whether the database server support InnoDB, SHOW ENGINES command can be used in mysql command prompt.

```
mysql>show engines;
```

If the database server which has been installed supports InnoDB, the window installer while configuring MySQL server makes InnoDB as the default engine in Windows.

2.2.4.2 MyISAM

MyISAM is the default storage engine for older versions of MySQL, generally before MySQL 5.5. This engine relies on the older ISAM code. However, many useful extensions have been added in this code. Each MyISAM table is split in three different files on disk. These three different files contain information about table format, data, and the indexes.

All these files have the same name as name of the specified table and the extension added to this name identify three different file types. The file with .frm denotes the format of table. The .MYD and .MYI extension describe the data file and index file respectively.

The maximum number of rows for server with MyISAM engine is limited to 2^32 (≈ 4.295E+09) rows. But, if the server is built with –with-big-tables option , the limit on maximum number of rows that MySQL server supports will be increased to (2^32)^2 (≈ 1.844E+19) rows. The MySQL server can have maximum 64 indexed fields per MyISAM table and they can have maximum 16 columns per index in a MyISAM table.

2.2.4.3 Basic Differences between InnoDB and MyISAM engines

MyISAM engine is faster than InnoDB in case of execution of queries if the number of queries is not enormously large. It means that the time taken for selecting, updating and inserting is relatively less under normal condition than the time required doing this operation for InnoDB engine.

The features like row-level locking, transaction-safe queries and relational table design in InnoDB engine make the InnoDB engine very attractive and popular engine to choose.

The recovery in case of a crash or any unexpected shutdown is possible in InnoDB. This recovery can be achieved by replaying its logs. In case of MyISAM, the engine need to carry out full scan and to repair any indexes or possibly tables if the insert or any update operation is performed on table but not completely flushed on disk.

The MySQL table ENGINE can be defined at the time of creating table inside defined database. This ENGINE definition can be performed by issuing command as shown below.

```
mysql>CREATE TABLE tablename (val1 INT, val2 INT,
val3 varchar(90),val4 varchar(90),PRIMARY
KEY(val2)) ENGINE=MyISAM;
```

If ENGINE definition is not stated, default InnoDB storage engine will be defined.

The following statement can be issued to determine storage ENGINE of the table.

```
mysql>show table status;
```

This command gives some additional useful information regarding the table under specified database to.

Also, many features of the table can be changed by using command ALTER TABLE. In order to change the storage engine of table defined under specified database, the following command can be issued;

```
mysql>ALTER TABLE tablename ENGINE = InnoDB;
```

This command basically changes the storage ENGINE of table tablename to InnoDB if this table's previous storage engine is MyISAM.

2.3 Database Performance Analysis

The measurement of performance of databases can be performed in different ways and performance requirements can be different for different applications. There are some applications especially designed to perform complex queries while others applications are focused on high concurrency demands. There are different distinct database benchmarks which are designed for comparing different database servers to each other and traffic experiments for investigating the dynamics of a working system.

2.3.1 Database Benchmarks

The first database benchmark developed by Bitton, DeWitt and Turbyfill [5] is called Wisconsin Benchmark which was widely spread. Mainly, this benchmark is used for the measurement of speed of different SELECT

queries. In this benchmark, the execution of all of the tests is carried out by single users in sequential order.

There is another well known benchmark developed by Turbyfill, Orji and Bit-ton [6].It is called as AS3AP (Ansi SQL Standard Scalable and Portable).This benchmark is basically created for the comparison of the different DBMS implementations. Both single user and multi-user tests are included in AS3AP.

The multi-user tests are carried out first to determine the number of users where maximum throughput is achieved. The real tests are conducted after finding the optimal number of users. The benchmark specifies a long series of tasks which are carried out by the database server. The defining of final performance measurement is specified as the maximum database size for which the server can execute all tasks in 12 hours.

There are new and updated database benchmarks provided by Transaction Processing Performance Council (TPC). Currently, three different benchmarks suites namely TPC-C, TPC-E and TPC-H are proposed. All these benchmarks are intended to simulate the complete complex system which reflects real world applications.

2.3.2 Traffic Experiments

The benchmarks that are mentioned above are intended to provide the general measure of the overall performance of the server. However, this thesis is more focused on the investigation of behavior of the database for different changed scenarios. This investigation explores the real dynamics of a running system.

A traffic generator is required in order to load test the database server. The traffic generator generates the requests to the database server. This generation of requests can be random or replayed from a recording of a real system. The response time from the requests are logged and are further analyzed.

There are a number of traffic generators available as a open source software for web servers. One example of traffic generator for web servers is CRIS[7]. However, these traffic generators are not easily available even though they exist. One traffic generator that can generate requests to both web servers and database servers is called Apache JMeter [8].

2.4 The Java Programming Language

2.4.1 Java Platform

Java is described as a programming language which is developed by James Gosling at Sun Microsystems (now part of Oracle Corporation) and is released as a core component of Sun Microsystems' Java platform in 1995. Java language derives many of its syntax from C and C++ languages. However, it has its own simpler object model and fewer low-level facilities. In the preface of the Java language specification, the language is described as follows.

"*The Java programming language is a general-purpose concurrent class-based object-oriented programming language, specifically designed to have as few implementation dependencies as possible. The main aim of usage of Java language is to facilitate the application developers to write a program once and then be able to run it everywhere on the Internet. Java language nowadays is considered to be one of the most popular programming languages in use, especially for client-server web applications.*" [9]

To be able to run Java applications 'anywhere' in the network, the java applications are compiled to what Java calls bytecode (class file).The Java Virtual Machine (JVM) interprets the bytecodes into code which can run on Java Virtual Machine (JVM) everywhere with no regard of computer architecture. The JVM ensures that the application is unable to do anything that it is not intended to do.

The JVM uses a mechanism called garbage collection for the memory management for the application. This implies that old objects which do not need memory are treated as 'garbage' and this garbage are collected by garbage collector and deleted. This freed memory space is available for

subsequent new objects. This garbage collector should have a mechanism to determine which objects are no longer referenced by the application and free the memory heap used by these unused objects. In general, the current garbage collector operates in parallel with the application. However, in some case, it sometimes needs to pause the application in order to accumulate some of the oldest objects. The other advantage of garbage collector is that it provides assurance of program integrity which is an important part of Java security strategy.

Java Platform, Standard Edition or Java SE is the mostly used platform for developing Java applications. The Java SE contains Java Virtual Machine along with a set of standard libraries. Java Platform, Enterprise Edition (Java EE) extends the features of Java SE with specifications and libraries needed for building of enterprise server applications.

2.4.2 Java Database Connectivity (JDBC)

Java Database Connectivity (JDBC) is an API which defines interfaces and classes for Java SE for connecting and accessing relational databases. This JDBC enables connecting and interacting with different relational databases. Many database manufactures and vendors provide drivers which when loaded allow the users and DBMS to interact with database and Java applications.

3 Problem Descriptions and Methods

3.1 Problem Description

3.1.1 Why do we need this model and where are we going to use it?

The MySQL database server basically executes the queries that arrive at the server through any of the predefined number of concurrent connections. If the arrival rate of the queries increases, the throughput of server for queries will also increase. However, every database server has its own capacity of handling queries arriving to the server at certain rate. If the arrival rate of these queries exceeds beyond that certain rate, the server gets overloaded and rejects some of the queries. We need a model that can provide detailed analysis of this behavior of the server. In our case, the model showing dynamics of the mean response time of request as a function of the mean arrival rate is used for the analysis of this behavior of the server.

The model generally shows the mean response time for low traffic load, high traffic load and maximum load beyond which the server gets overloaded. Therefore, this model is necessary to design the response time estimator and controller logic inside the application server to keep database server operating in normal condition. Here, normal condition implies the condition where server is running without getting overloaded.

Many computing systems require that their server should operate in high load traffic without being overloaded. In this case, we need to determine the ranges of the values of queries mean arrival rate over which the server remains in high load traffic. With the help of such a model, we can design a response time estimator to determine ranges of the values of requests mean arrival rates for which the mean response time lie within high load traffic below the maximum value of queries mean arrival rate beyond which the server gets overloaded. After this, the controller is implemented inside the application server that controls the amount of traffic to ensure that the queries mean arrival rate lies within the high traffic loads below the overload region of database server.

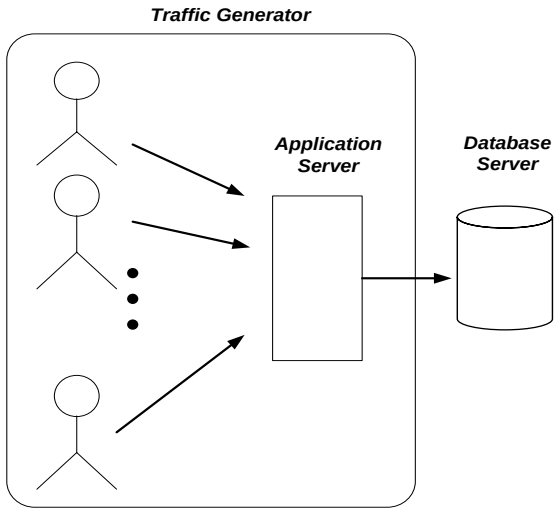


Figure 3.1 Experiment Setup

Also, there are many systems that may require fixed requests' mean response time delay. This means that the mean response time for those systems should be below a predefined desired value. Similarly, the model can be used to design the response time estimator and controller which will help to determine the mean value of response time and control the request arrival rate so that the mean response time is always below the desired value.

3.1.2 What types of models do we use?

When the traffic generator generates requests with a certain mean arrival rate for predefined period of time then we can achieve the measurement of samples of response time for that arrival rate. Since the instantaneous measurements of all samples of response time for each input arrival rate is very noisy, the long term averaging of all the samples of response time are performed in order to achieve model for steady state measurement.

3.2 Methods

The behavior of the database server has been analyzed by iterative experiments on the database server. All experiments are conducted by

exposing the database server with different mean arrival rates (requests per second) to investigate how the server mean response time behavior changes.

Besides this, experiments are also conducted to investigate dynamics of the server mean response time of requests for different scenarios. These scenarios basically involves change in request type which means either Select or Update or mixture of Update and Select, change in the database storage engine, change in the relations sizes for several relations and change in the number of concurrent connections to server.

For a certain arrival rate, the traffic generator is run for a certain period of time so that enough requests are generated which in turn, delivered to the database server to provide enough samples of response time. The longer the traffic generator runs, the larger the amount of the requests traffic would be and hence the larger the amount of samples of response time would be. Generally, if we have large samples of response time, we can measure response time more exactly. The optimum time for how long time the traffic generator needs to run for the measurements of response time depends upon how much accuracy we want in our measurements of response time. In our case, the traffic generator is run for 5 minutes for a certain arrival rate.

The inter arrival waiting time between two consecutive queries follow an exponential distribution. These times generally are of the order of some milliseconds. After getting enough samples of response times for a certain mean arrival rate, we take the mean of all the samples for that mean arrival rate. Similarly, the experiment is repeated for another value of mean arrival rate and corresponding mean is taken. This process is iterated for a range of mean arrival rates so that we can measure the mean response time for that range.

The whole process of measuring the mean response time for a range of mean arrival rates is repeated for at least five times and the mean of all five mean response time for each arrival rate is taken in order to get a more exact mean value. Thus, we can achieve the range of mean response times for the range of request's mean arrival rate. The number of times the whole process is repeated depends upon the how exactly we need to measure the mean response time value. If we need more exact value of the mean response time, the whole process of measurement of the mean response time of a

request is repeated for more number of times. The plot of mean response time against arrival rate is shown in Figure 3.2.

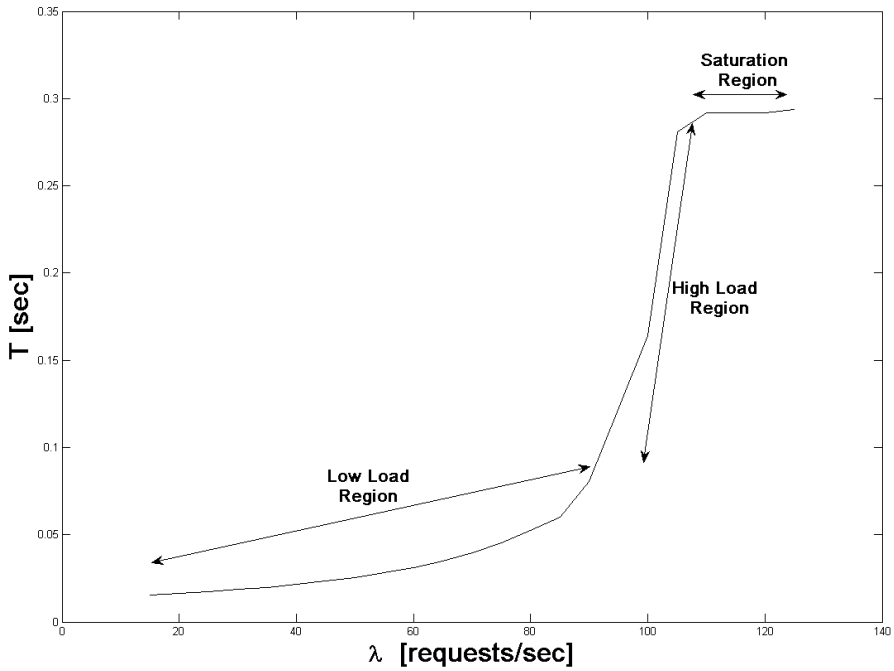


Figure 3.2: General output mean response time plot specifying three distinct load regions

Each experiment is performed for ranges of mean arrival rate of requests enough to be able to show mean response time of database server for low load region, high load region and saturation region.

The region on the Figure 3.2 that shows the less steep part in the dynamic of mean response time of requests is considered to be low load region.

The region on the Figure 3.2 that shows the more steep part in the dynamic of mean response time of requests is considered to be high load region.

The region on the Figure 3.2 where output mean response time curve of request delivered to the database server gets saturated and gives flat

response is considered to be saturation region. In other words, saturation region is the region where database server for a particular request gets overloaded for the defined number of connections.

4 Lab Setup

4.1 Database Server

The data tier in a multi-tier server system consists of a database server. Most of the experiments is performed on the database server together with the application server.

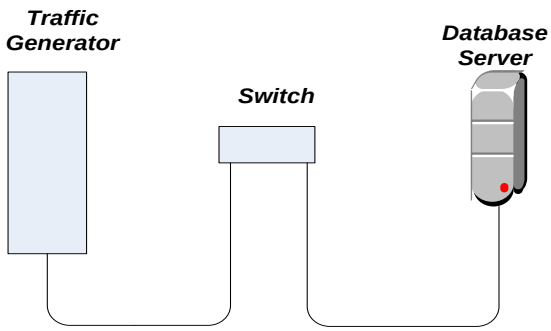


Figure 4.1: Lab Hardware Setup

4.1.1 Server Hardware and Software

The experiments are performed in the lab of Lund university consisting of two ordinary computers with some open source software installed. The hardware of computer involved in the experiment is a DELL OptiPlex GX270. The other hardware is equipped with a computer that constitutes an 1.86 GHz Intel(R) Core(TM)2 CPUs processor, 2022MB RAM main memory, a single hard-drive and a Gigabit Ethernet network card. The hard drive has SATA interface with capacity of 160 GB. The rotational speed of the hard drive is 7200 RPM and the hard drive has buffer size of 2048kB. The computers are connected with a Fast Ethernet switch. Here, one out of two computers is acting as traffic generator while the other computer has the default version of the MySQL server 5.1 installed on it. The Windows XP is running on all computers.

The MySQL server is further configured such that it is opened for external connections and the maximum number of concurrent connections has been increased to 800 connections. Although there are many different database

engines that MySQL can support, the default INNOdB storage engine and MyISAM storage engine have been selected for the experiments.

4.1.2 Database Structure

In this experiment, only one database is used which contains several relations. All relations are made up with same scheme however these relations may have different relations sizes. The concept of basic structure of the relation comes from the database benchmark developed by Bitton, DeWitt and Turbyfill called the "Wisconsin Benchmark" [4].The actual structure comes from a newer version of this benchmark that is more scalable with respect to relation size [10].The limit on the maximum number of tuples is determined by the number of unique values of a 32 bit integer which is $2^{32} \approx 4.3 \times 10^9$ tuples. The relation consists of 16 attributes. Two relations of InnoDB storage engine with size of five million and ten million tuples and three relations of MyISAM storage engine with size of two million, five million and ten million tuples are used in these experiments.

The two attributes in the relations namely unique1 and unique2 are integers in the range between zero to the number of tuples. Here, unique1 is random while unique2 is sequential while other remaining thirteen attributes are derived from either of them. Based on the specification of the benchmark, a Huskell program is utilized to create the data for the relation. Haskell is a lazy functional programming language [11]. In order to create the sequences for unique1 and unique2, a linear feedback shift register which is used to create a maximum length sequence is used. The program not just only generates the sequences, instead the whole relation is completed with INSERT-statements with this program.

4.2 Traffic Generator

4.2.1 Introduction

To determine how the behavior of the server changes with load, the load needs to be introduced in some way. However, the whole aim of the experiment is not only analysis of the behavior of the database server, but

also comparing that behavior for different scenarios. Therefore, the traffic directed to the database server is controlled in the application server. All this behavioral changes and control mechanisms should be included in the traffic generator.

The major function of the traffic generator is to generate desired traffic and measure the response time accurately. In order to make the traffic generator effectively, there are some conditions that need to be considered carefully.

1. Ability to measure the mean response time correctly.
2. Ability to produce traffic that follows desirable arrival distribution.

The generator is supposed to perform some concurrent activities. These activities include generating of requests, processing of requests and logging the response time accurately. All this activities are demonstrated with the Java threads.

4.2.2 Concurrent Activities

Basically the traffic generator performs three concurrent activities. It should firstly generate query requests, process the requests secondly and finally log the response times of the requests. All these activities are carried out by Java threads. The generation of query requests follows a Poisson process and fixed numbers of workers are Java threads that execute these query requests directed to the database server. All threads and connection are set up before the actual traffic generation and execution starts.

The traffic generator sends query requests to the database server. Inside the traffic generator, the interval between generations of request is controlled on the signaling thread. Each worker thread has built a connection with the database and waits for the requests to come for execution. Whenever a request arrives then the signaling thread signals the worker threads, the worker threads first establish connection to the database and take that request. Before executing that request, it starts a timer and when response is received, it stops the timer. By doing this, the actual response time is logged which is the difference of start time and stop time.

4.2.3 Arrival Distribution

The traffic generator generates the queries after waiting a random exponentially distributed time so that the generation of queries follows the Poisson process. However, the queries that arrive to the database server may not be able to follow the Poisson process all the time. The reason behind this is that the Java threads which actually are worker threads are responsible to fetch these queries to the database server. If all the workers threads are busy processing the requests then the incoming requests need to wait for processing. Due to this reason, the arrival distribution of requests to the database server (or logged response time) no longer follows the Poisson process.

This arrival distribution has been achieved based on the concept of Counting Semaphore. The basic of Counting Semaphore will be discussed here.

4.2.3.1 Concept of Semaphore

Semaphore is a variable or abstract data type which is most often used in a parallel programming environment for providing controlling access by multiple processes or threads to common resources. This concept of semaphore emerges when several threads frequently need to access a limited number of resources.

4.2.3.2 Description

A counting semaphore is simply a synchronization object. This counting semaphore can consist of an arbitrarily large number of states where an integer variable called counter defines this internal state.
The value of the counter defines a particular meaning:

Counter	Value
zero	All waiting worker threads
positive	No waiting worker threads

Table 4.1: Counter value and their meaning

There are two operation defined for counting semaphores:

- **Wait**, this operation decreases the semaphore counter, if the result is zero, the invoking worker threads waits until this result is positive.
- **Signal**, this operation increases the semaphore counter, if the result is positive, then the signaling thread signals the calling worker threads.

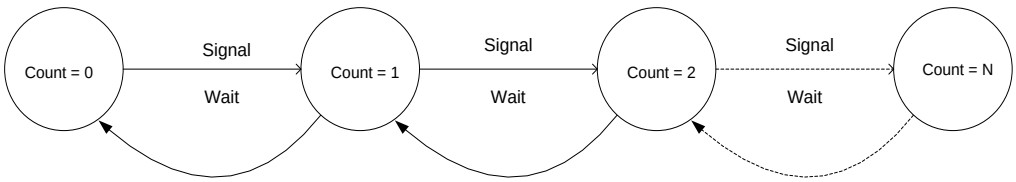


Figure 4.2: Concept of states on counting semaphore

Counting Semaphores are used mostly for synchronization purposes which allow several calling worker threads to wait until an event has occurred. Usually the event is generated by a signaling thread. An integer counter (usually initiate to 0) is used to keep tract of each associated semaphore. The signaling thread is responsible to increase the integer counter by one at a time which in turn, automatically decreases by associated calling worker thread in order to acquire an instance of a resource. It means that if the value of the counter is greater than 0, it announces the availability of resources and the waiting worker thread can consume one event and decrease the counter and return immediately. If the value is 0, the calling worker thread needs to wait until the signaling threads increase the counter value by posting new event. The signaling thread used for posting to a semaphore is responsible for waking up the first worker thread that is currently waiting, which in turn resume semaphore wait operation and decrease the counter again.

Semaphore can also be used for a certain type of resource management. The counter indicate the amount of the currently availability of a certain type of resource for which the calling worker threads are waiting on the semaphore. These resources become available again when the signaling thread posts these resources again.

4.2.4 Query Generator

In this experiment, requests are sent to the database server through multiple connections where each request is sent per connection basis. Each query is denoted by a String. In different experiments, query generator generates different queries that are sent to the database. The general structure of the query statement is the same where the only difference is on different parameters. The sample example representing the basic structure is shown below.

```
SELECT attribute FROM relation WHERE otherAttribute=?
```

where the question-mark is replaced by a number between 0 and total number of tuples. In another experiment the query looks like

```
UPDATE otherRelation SET attribute=? WHERE attribute=?
```

where both question-marks are replaced by different numbers between 0 and total number of tuples.

5 Experiments

5.1 Introduction

To be able to investigate a performance model of the database server, several experiments are carried out. All these experiments are performed with different loads to identify how the dynamics of the mean response time of queries sent to the database server changes with different loads.

5.1.1 Basic Experiment

The methods for carrying out these experiments are basically similar. The computer running as database server is connected to another computer acting as a traffic generator which measures the response time it takes for the database server to execute the requests. The major functionality of the traffic generator includes generation of requests to the database server. Before the generation of requests in the traffic generator starts, multiple concurrent connections to the database server are established and verified. After confirming that the connections are perfectly working, requests are transmitted to the database server on these connections. The transmission of requests follows the Poisson process with a specified rate and all the response times are logged. The whole experiment is carried out for some predefined time, usually some minutes. After this time, the connections are closed and the process for generation of requests is stopped. All the log files are saved in a file with a filename that corresponds to the mean arrival rate. Similar experiments are repeated with different mean arrival rate and all the log files are saved in the folder. Later, the results are processed and necessary graphs are plotted with the help of Matlab.

5.1.2 SQL

The requests generated from the traffic generator to the database server consists of either SELECT or UPDATE or a mixture of SELECT and UPDATE statements in which the probability of choosing UPDATE statement is 75 percent and of SELECT statement is 25 percent. The general structure of requests is explained as follows.

```
SELECT * FROM tablename WHERE unique2=?;

UPDATE tablename SET val6='some random string'
WHERE unique2=?;
```

The relation tablename is generated based on Wisconsin Scalable (wiscs) Benchmark with a predefined size. The attribute unique2 was indexed in sequential order. In place of question marks shown above, the randomly generated integer number is inserted. The insertion of these numbers follows a uniform distribution that lies between zero and the total number of tuples.

Two MySQL relations of InnoDB storage engine are used with size of tuples equal to 5x10^6, and 10x10^6 tuples respectively and three MySQL relations of MyISAM storage engine are used with size of tuples equal to 20x10^6, 50x10^6, and 100x10^6 tuples respectively.

5.1.3 Verification

All the response times are logged in a file. All the queries from the traffic generator arrive at the database server through the predefined concurrent connections. If all connections are occupied at the same time in an experiment, the database server is considered to be saturated and then that experiment is excluded.

5.2 Experiment: Select only

5.2.1 Description

The Select request is based on read only operations on the single or multiple columns of the single row of MySQL relations. Therefore, the locking of a particular section of database is not required.

The experiment involves forwarding Select requests from the traffic generator to the MySQL database with the structure as below.

```
mysql>SELECT number FROM tablename WHERE val2=?;
```

Here, val2 is the primary key of the relation tablename. The primary key is used for unique indexing of number of rows in relation tablename.

The purpose of the experiment is to determine dynamics of the mean response time of Select requests when data are read from a particular row of a relation. In order to compare the dynamics of mean response time for Select requests, the experiment is iterated for several relations with various relation sizes.

The mean response time of the MySQL server for Select requests depends on the mean arrival rate, relation size and storage engine of database server. Therefore, the experiment is conducted to investigate mean response time for 5MSelectInnoDB, 10MSelectInnoDB, 20MSelectMyISAM, 50MSelectMyISAM and 100MSelectMyISAM.

The mean response time of Select request in MySQL relations increases with the increase in size of relations. The input request mean arrival rate is increased with step sizes of 5.

5.2.2 Result

Figure 5.1 compares the dynamics of mean response times of 5MSelectInnoDB, 10MSelectInnoDB, 20MSelectMyISAM, 50MSelectMyISAM and 100MSelectMyISAM.

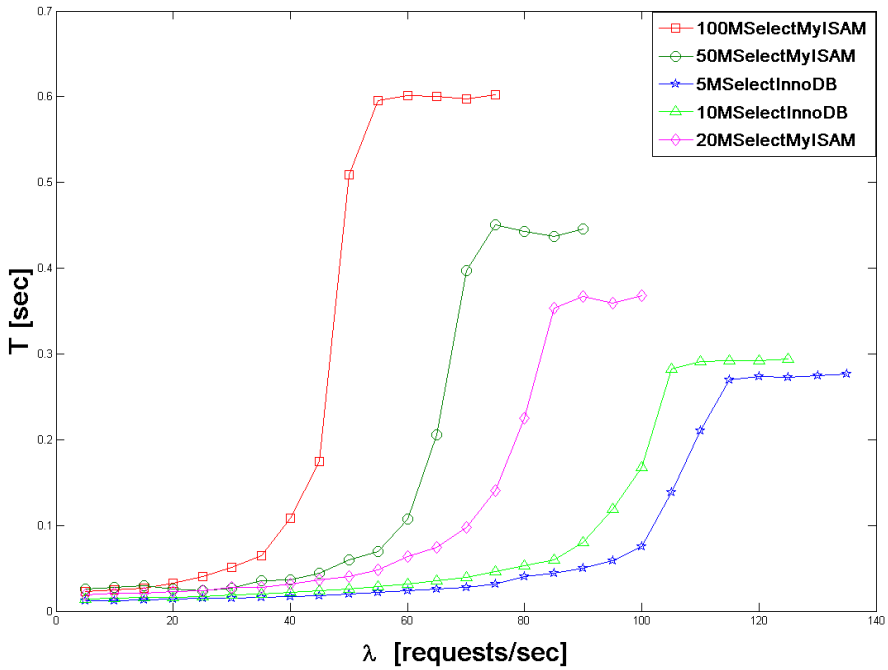


Figure 5.1: Comparison of mean response time of 5MSelectInnoDB, 10MSelectInnoDB, 20MSelectMyISAM, 50MSelectMyISAM and 100MSelectMyISAM.

The 5MSelectInnoDB remains in the low load situation for the highest value of request mean arrival rate while the 100MSelectMyISAM remains in the low load situation for the smallest value of request mean arrival rate. This is because 5MSelectInnoDB involves relations with size of 5 million and 100MSelectMyISAM involves relations with size of 100 million. The mean response time curve of Select request for relations with large relation size (100M) shows high value at lower input arrival rate than for relations with low relation size (10M) at high input arrival rate.

The rising trend of server mean response time for Select requests in all scenarios is more or less similar even if relations with different engine types

are involved. These similarities in rising trend are because of the fact that the Select request basically performs read only operations which doesn't require locking of part of database. Since the reading operation on MySQL relations is same for both InnoDB storage engine and MyISAM storage engine, the mean response time of Select requests do not depend on the MySQL relations engine type. The 100MSelectMyISAM saturates at the smallest value of mean arrival rate (around 50) with the highest value of mean response time (around 0.6 sec) while 5MSelectInnoDB saturates at highest value of mean arrival rate (around 110) with smallest value of mean response time (around 0.26 sec). The reason behind this trend of saturation of mean response time is that the Select requests in these experiments utilize different MySQL relations with different relation sizes.

5.3 Experiment: Update only

5.3.1 Description

The Update request is based on write operations which require locking of the part of the database. The locking method of the database depends upon on the storage engine type of the database. The InnoDB performs row-level locking while MyISAM allows table locking when updating.

The request from the traffic generator consists of an UPDATE-query which is structured as follows.

```
mysql>UPDATE tablename SET val5='some random string' WHERE val2=?;
```

Here, val2 is the primary key of the relation tablename.

Similar to the Experiment 5.2, the mean response time of the MySQL server for the Update requests depends on factors such as request mean arrival rate, different relation size and storage engine of database server. Therefore, the experiment is conducted to measure the mean response time for

5MUpdateInnoDB, 10MUpdateInnoDB, 20MUpdateMyISAM, 50MUpdateMyISAM and 100MUpdateMyISAM and to compare these mean response time to investigate the effects of above mentioned factors on it. The input request mean arrival rate is increased with step sizes of 1.

5.3.2 Result

Figure 5.2 shows the dynamics of the mean response times of 5MUpdateInnoDB, 10MUpdateInnoDB, 20MUpdateMyISAM, 50MUpdateMyISAM and 100MUpdateMyISAM.

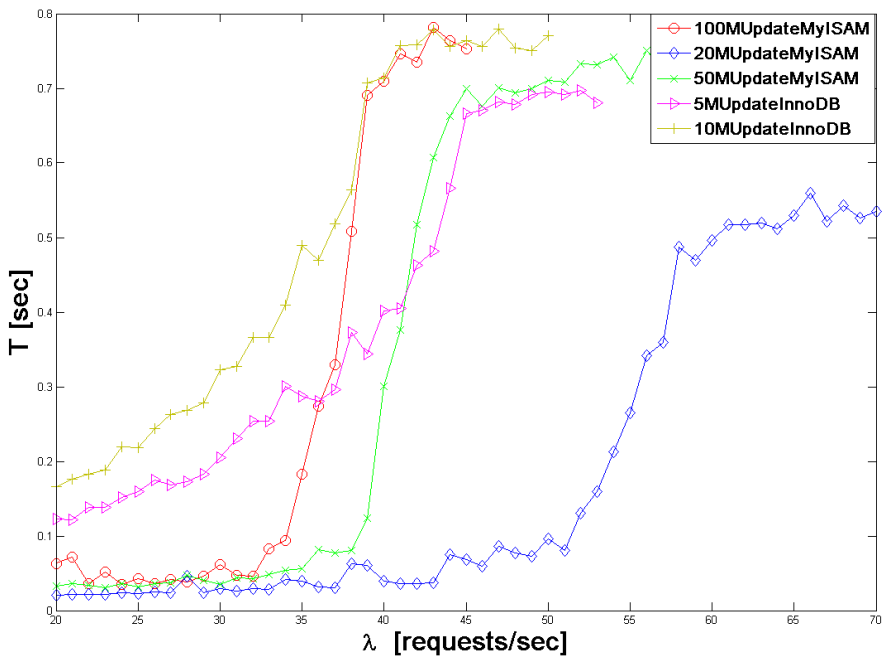


Figure 5.2: Comparison of mean response time of 5MUpdateInnoDB, 10MUpdateInnoDB, 20MUpdateMyISAM, 50MUpdateMyISAM and 100MUpdateMyISAM.

The mean response times for 20MUpdateMyISAM, 50MUpdateMyISAM and 100MUpdateMyISAM show almost similar flat response for low load situations. This low load situation can be measured when the values of mean arrival rate for 20MUpdateMyISAM, 50MUpdateMyISAM and 100MUpdateMyISAM are below 34, 38 and 50 respectively. However, the mean response time of 5MUpdateInnoDB, 10MUpdateInnoDB shows almost linear response at low load situations while 20MUpdateMyISAM, 50MUpdateMyISAM and 100MUpdateMyISAM shows almost flat response at low load situations. This change in trend of graph in Figure 5.2 is due to the fact that 20MUpdateMyISAM, 50MUpdateMyISAM and 100MUpdateMyISAM involves relations with MyISAM storage engine while 5MUpdateInnoDB and 10MUpdateInnoDB involves relations with InnoDB storage engine. In case of low load situation, the mean response time of 5MUpdateInnoDB and 10MUpdateInnoDB which utilize InnoDB engine is relatively higher than that of 20MUpdateMyISAM, 50MUpdateMyISAM and 100MUpdateMyISAM which utilize MyISAM engine.

The rising trend of mean response time of Update request for relations with MyISAM engine type is steeper than that for relations with InnoDB engine type. It can be seen from Figure 5.2 that 5MUpdateInnoDB and 50MUpdateMyISAM saturate for similar values of mean arrival rate (around 45). Also, the mean response time for saturation shows a similar value (around 0.7 sec) for both 5MUpdateInnoDB and 50MUpdateMyISAM.

Similarly, the mean response time of 10MUpdateInnoDB and 100MUpdateMyISAM saturates at similar value (around 0.78 sec) and corresponding mean arrival rates also show similar value (around 41).

The different trend of the graph in Figure 5.2 is due to the fact that 5MUpdateInnoDB and 10MUpdateInnoDB utilize InnoDB engine which involves row level locking while 20MUpdateMyISAM,

50MUpdateMyISAM and 100MUpdateMyISAM utilize the MyISAM engine which involves table level locking.

5.4 Experiment: Mixture of Update and Select

5.4.1 Description

This experiment is conducted to determine the mean response time of the mixed requests consisting mixture of Update and Select requests. The generation of mixed requests from the traffic generator is performed in such a way that the probability of generation of Update requests is 75 percent and probability of generation of Select requests is 25 percent.

The mean response time for mixed requests depends upon a number of factors such as the mean arrival rate, relation size and different storage engines of the database server (either InnoDB or MyISAM). The experiment is performed with 5MmixInnoDB, 10MmixInnoDB, 20MmixMyISAM, 50MmixMyISAM and 100MmixMyISAM to compare and investigate the effects of these factors on mean response time of mixed requests. The experiment is performed with input mean request arrival rate taken at step sizes of 1.

5.4.2 Result

Figure 5.3 compares the dynamics of the mean response time of mixed requests for five different scenarios (5MmixInnoDB, 10MmixInnoDB, 20MmixMyISAM, 50MmixMyISAM and 100MmixMyISAM).

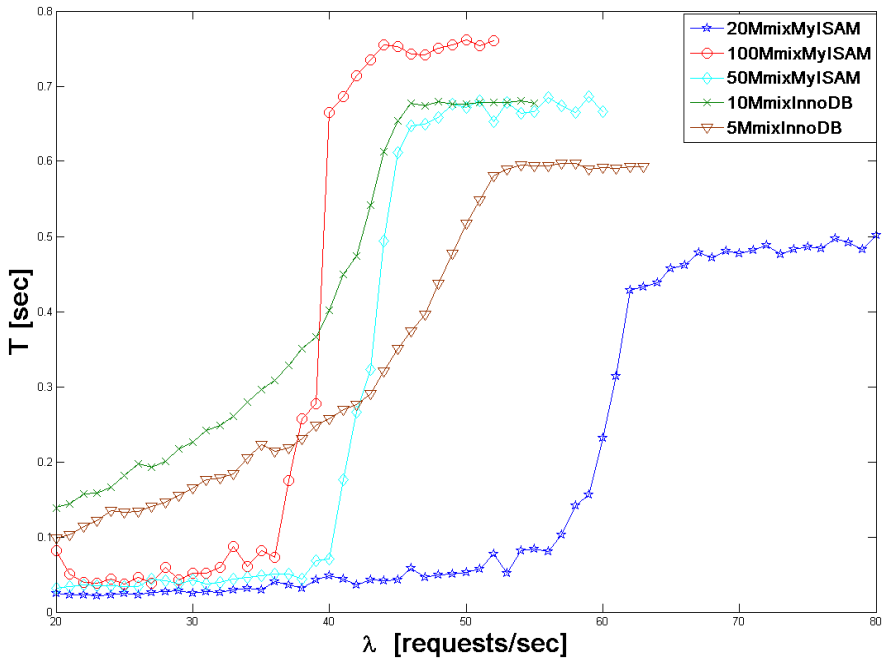


Figure 5.3: Comparison of mean response time of 5MmixInnoDB, 10MmixInnoDB, 20MmixMyISAM, 50MmixMyISAM and 100MmixMyISAM

Generally, the dynamics of mean response times for mixed requests for all five scenarios (5MmixInnoDB, 10MmixInnoDB, 20MmixMyISAM, 50MmixMyISAM and 100MmixMyISAM) is more or less similar to that for Update requests for five scenarios (5MUpdateInnoDB, 10MUpdateInnoDB, 20MUpdateMyISAM, 50MUpdateMyISAM and 100MUpdateMyISAM) of Figure 5.2.

However, the mean arrival rate required for saturation of 100MmixMyISAM (around 43) is less than that of 10MmixMyISAM (around 46). The mean response time for saturation of 100MmixMyISAM (around 0.76 sec) is greater than the mean response time for saturation of

10MmixInnoDB (around 0.66 sec). Instead, both the 50MmixMyISAM and 10MmixInnoDB at saturation show similar values of mean response time (around 0.66 sec). Also, the input request mean arrival rate required for saturation of both 50MmixMyISAM and 10MmixInnoDB are of almost similar values (around 46).

The mean response time of 5MmixInnoDB, 10MmixInnoDB, 20MmixMyISAM, 50MmixMyISAM and 100MmixMyISAM at saturation show correspondingly smaller value than that of 5MUpdateInnoDB, 10MUpdateInnoDB, 20MUpdateMyISAM, 50MUpdateMyISAM and 100MUpdateMyISAM of Figure 5.2. However, the mean arrival rate needed for saturation of all 5MmixInnoDB, 10MmixInnoDB, 20MmixMyISAM, 50MmixMyISAM and 100MmixMyISAM is correspondingly higher than that of 5MUpdateInnoDB, 10MUpdateInnoDB, 20MUpdateMyISAM, 50MUpdateMyISAM and 100MUpdateMyISAM of Figure 5.2.

One reason for this change in trend of graph in Figure 5.3 compared to the graph in Figure 5.2 is the selection of requests type. The experiment 5.3 involves mixed requests which can be either Update or Select request from the traffic generator for five different scenarios. However, experiment 5.2 is conducted only with the Update queries for five different scenarios.

5.5 Experiment: Select and Update

5.5.1 Experiment : Comparison of 5M Select and 5M Update with InnoDB storage engine

5.5.1.1 Description

This experiment is firstly performed for Select queries on MySQL relations with size of 5million tuples and then the experiment is repeated for Update queries on the same MySQL relations. The purpose of this experiment is to determine and then compare the dynamics of mean response times for Select request and Update request. Here, default InnoDB database storage engine

type for MySQL relation is considered. The input mean request arrival rate is increased with step sizes of 1.

5.5.1.2 Result

Figure 5.4 compares the dynamics of the mean response times of 5MUpdateInnoDB and 5MSelectInnoDB .

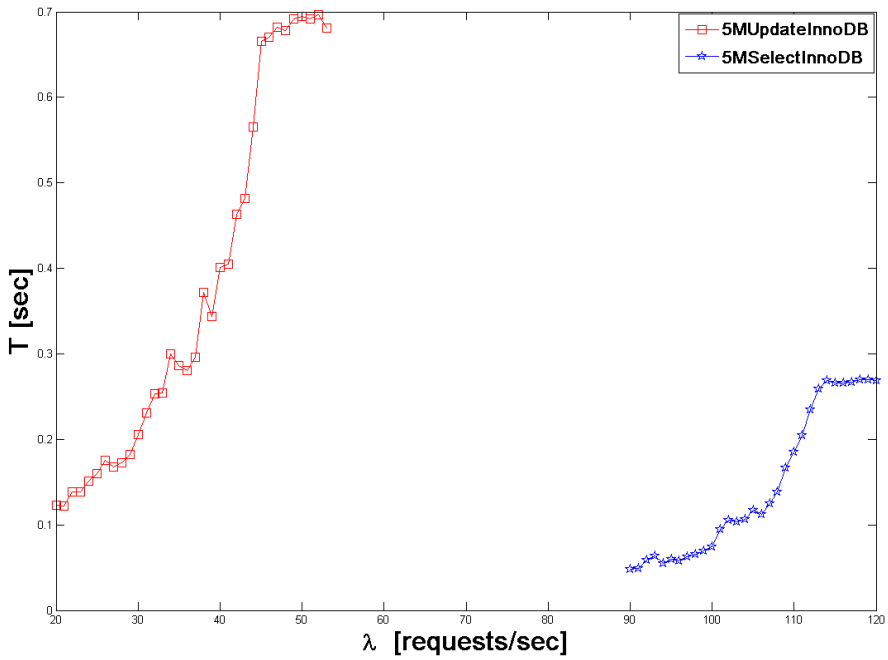


Figure 5.4: Comparison of mean response time for 5MUpdateInnoDB and 5MSelectInnoDB

The dynamics of the mean response times of 5MUpdateInnoDB are quite different than that of 5MSelectInnoDB.

The 5MUpdateInnoDB remains in a low load situation for request mean arrival rates less than approximately 20 while the 5MSelectInnoDB remains in a low load situation for request mean arrival rate less than approximately 105. This means that 5MUpdateInnoDB remains in a low load situation for

much smaller values of request mean arrival rate compared to 5MSelectInnoDB.

The reason behind this trend in case of a low load situation is that the server needs to wait for the requests from the traffic generator to arrive at the server in order to execute the requests. The mean arrival rate of requests on different concurrent connections is far lower than the capacity of the server to handle requests that arrive on different concurrent connections.

The rising rate of dynamics of mean response times for both 5MSelectInnoDB and 5MUpdateInnoDB in case of a high load situation is comparatively steeper compared to that of a low load situation. This is because the database server remains always busy executing requests arriving at the server through different concurrent connections.

The 5MUpdateInnoDB curve saturates when the mean arrival rate becomes higher than 48. This value of mean arrival rate is much smaller compared to the mean arrival rate needed for saturation of 5MSelectInnoDB. 5MSelectInnoDB saturates when the mean arrival rate becomes greater than 115.

The mean response time of Update for the overloaded condition (around 0.7 sec) is far higher than that of Select curve (around 0.28 sec).

5.5.2 Experiment : Comparison of 50M Update and 50M Select with MyISAM storage engine

5.5.2.1 Description

The experiment is carried out to determine and compare the dynamics of mean response times of both 50MUpdateMyISAM and 50MSelectMyISAM at different values of input request mean arrival rate. The input mean arrival rate is increased with step sizes of 1.

5.5.2.2 Result

The graph showing the comparison of dynamics of the mean response time of 50MUpdateMyISAM and 50MSelectMyISAM is plotted in Figure 5.5.

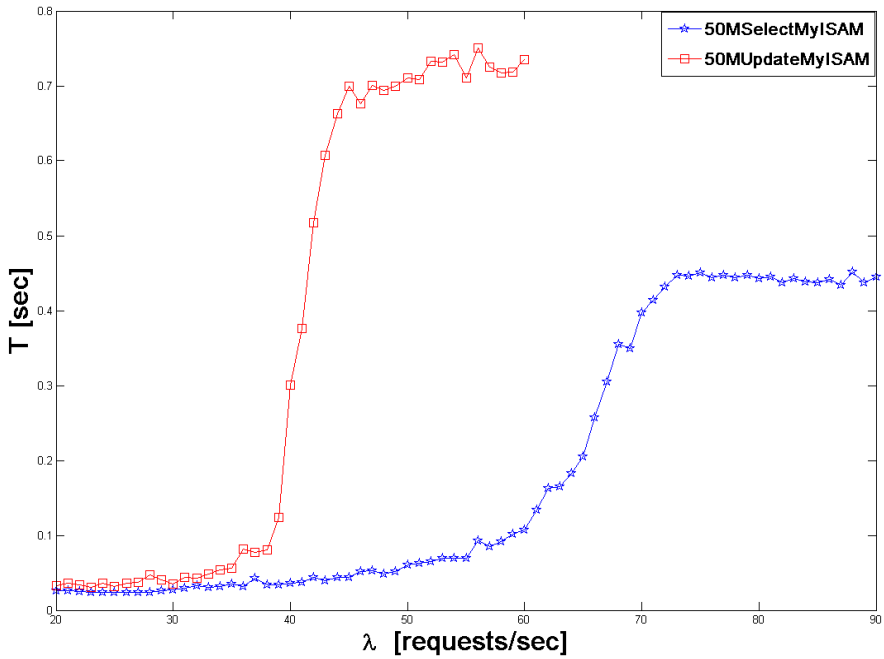


Figure 5.5: Comparison of mean response time for 50MUpdateMyISAM and 50MSelectMyISAM

The dynamics of mean response time of both 50MUpdateMyISAM and 50MSelectMyISAM show an almost flat response time at low load situations. The low load situations for 50MUpdateMyISAM and 50MSelectMyISAM are observed when the request mean arrival rates for both 50MUpdateMyISAM and 50MSelectMyISAM are less than 37. The flat nature of the mean response time for low load situation is due to the reason that the experiment is conducted for MySQL relations with MyISAM database storage engine. However, the mean response of

50MSelectMyISAM remains in a low load situation for a higher range of mean arrival rates than that of 50MUpdateMyISAM.

The high load situation for both 50MUpdateMyISAM and 50MSelectMyISAM shows a sharp rising when the mean arrival rate is greater than around 37 and less than the value where mean response times for these requests saturate. However, the rising rate of 50MUpdateMyISAM is steeper than for 50MSelectMyISAM.

The mean response time curve for 50MUpdateMyISAM saturates when the value of mean arrival rate becomes greater than around 44. The mean response time of 50MSelectMyISAM saturates when the mean arrival rate becomes greater than around 74. This shows that the mean arrival rate needed for saturation of 50MSelectMyISAM is much higher than for 50MUpdateMyISAM.

The mean response time for saturation of database server for 50MUpdateMyISAM is around 0.7 sec. This value is much higher than the mean response time for saturation of 50MSelectMyISAM (around 0.46 sec).

5.6 Experiment: Change in number of connections to the MySQL database server

5.6.1 Experiment: Effects of change in number of connections to the server on 10MSelect with InnoDB engine

5.6.1.1 Description

Several clients can access MySQL server at the same time. This can be done through multiple concurrent connections to the MySQL database server. It is possible to manually set the number of the connections to the MySQL server that can be built until the number do not exceed maximum connection limit.

The experiment is carried out to determine and compare the dynamics of the mean response time of 10MSelectInnoDB when number of concurrent connections to the database server is changed. Here, at first, the experiment

is conducted for the default number of concurrent connections. Afterward, the same experiment is repeated separately when the number of concurrent connections is manually changed to 15, 10 and 5 respectively. If the number of connections is manually reduced then the server rejects the requests that arrive at the database server through concurrent connections depending upon the number of connections reduced from the default connections. In all experiments, the default number of concurrent connections is assumed to be 30. The database server with InnoDB engine type is used. The input mean request arrival rate is increased with step sizes of 5.

5.6.1.2 Result

Figure 5.6 shows the dynamics of the mean response times of 10MSelectInnoDB5, 10MSelectInnoDB10, 10MSelectInnoDB15 and default 10MSelectInnoDB.

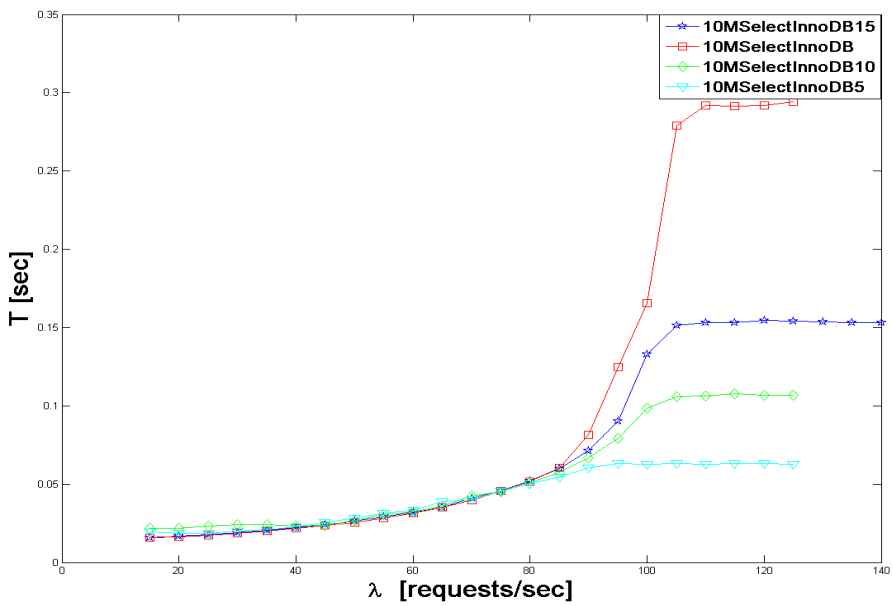


Figure 5.6: Effect of change in number of concurrent connections on 10MSelectInnoDB

When we increase the number of concurrent connections from 5 to the default value, the rising rate of mean response time of 10MSelectInnoDB for high traffic situations will also increase and trend of curves becomes correspondingly steeper. However, the mean response time for the low load region does not show any significant changes for different number of concurrent connections.

The saturation of the mean response time of 10MSelectInnoDB requires less value of mean arrival rate if the number of concurrent connections is decreased from the default number. Similarly, the mean response times for saturation of 10MSelectInnoDB show smaller values if the number of concurrent connections decreases from default number.

The reason behind this trend is that if we increase the number of concurrent connections to the database server then rate of number of concurrent execution of queries (throughput) will also be increased corresponding to the increased number of concurrent connections.

However, if we reduce number of concurrent connections without reducing the default number of worker threads, fewer queries will reach to the database server through these reduced connections. Therefore, the throughput of the database server will be also reduced.

Hence, if we reduce the number of concurrent connections, the ability of the server to execute queries in parallel will also be reduced. Therefore, the server becomes saturated for a smaller value of the mean arrival rate in compared to a server with an increased number of concurrent connections.

5.6.2 Experiment: Effects of change in number of connections to the server Decreased on 50M Update with MyISAM engine

5.6.2.1 Description

This experiment is conducted to investigate the effect of changing the number of concurrent connections to the database server on the dynamics of the mean response time for Update requests. The numbers of concurrent connections considered in this experiment are default 30, 20 and 15. The

database server with MyISAM storage engine type is used. The input mean request arrival rate is increased with step sizes of 1.

5.6.2.2 Results

Figure 5.7 shows the mean response time of Update request for different number of concurrent connections to the database server.

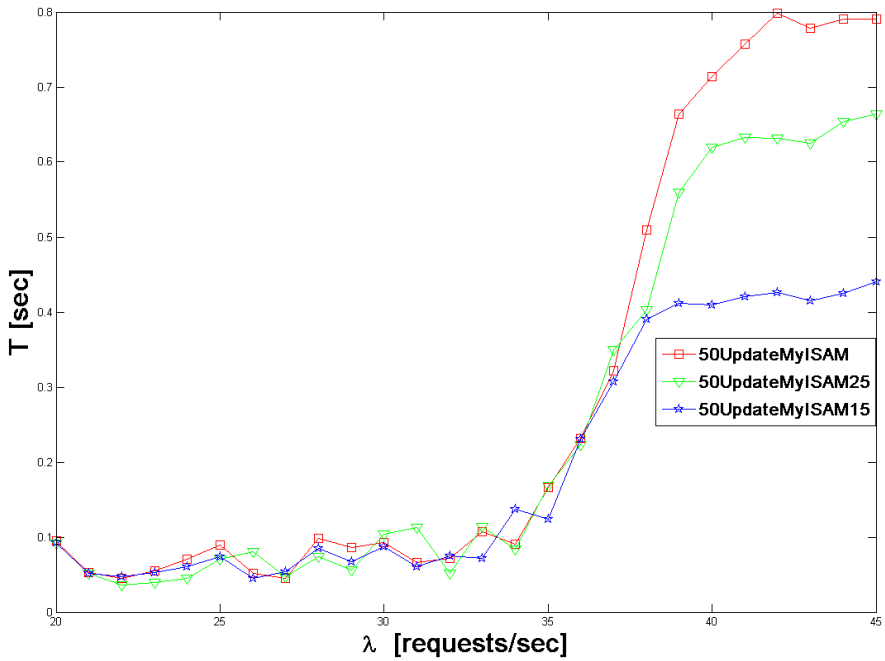


Figure 5.7: Effect of change in number of concurrent connections on 50MUpdateMyISAM

Similar to the Select query in Figure 5.6, the mean response time of Update requests for all three concurrent connections to the server (default 30, 20 and 15 connections) do not show any significant difference in low load situations. However, the rising trends of these mean response times for high load situations are quite different. The mean response times of Update for all three concurrent connections saturates for a smaller value of mean arrival rate in compared to the mean response time for Select request shown in

Figure 5.6. Also, the corresponding saturation value for the mean response time is higher for Update than that for Select shown in Figure 5.6.

The reason behind this trend of graphs which are measured after the number of connections is manually reduced is same as the trend of graphs in experiment 5.1. However, the trend of graph in Figure 5.7 shows an almost flat response at low load situations due to the nature of MyISAM storage engine involved in this experiment.

5.7 Experiment: Effect of choosing different Step Size for Input Request Arrival Rate

5.7.1 Description

This experiment is conducted to observe the effect of the selection of step sizes for the input request arrival rate on the dynamics of the mean response time of requests delivered to the database server. If we choose a large step size of input request mean arrival rate, we cannot figure out the dynamics of output mean response time of requests accurately and exactly. In this experiment, we choose step sizes for the input request mean arrival rate of 1 and 5.

5.7.2 Results

Figure 5.8 shows the comparison of the dynamics of mean response time of Update request for input mean arrival rate increased with step sizes of 1 and 5.

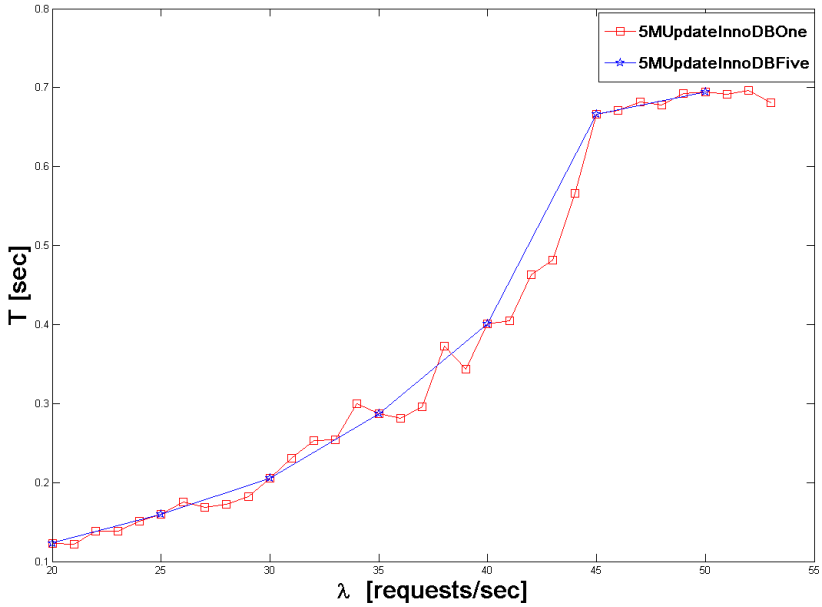


Figure 5.8: Effect of choosing different step size for input request arrival rate on 5MUpdateInnoDB

The dynamics of the mean response time of Update request in high load situation can be more exactly measured if we choose a small step size for the input request mean arrival rate. If we choose a large value of step size, then we may not be able to measure the dynamics of the mean response time of Update request more accurately. Also, the rising rate of the trend in the graph can be determined exactly if we choose a small value of input step size.

In the above Figure 5.8, the mean response time of Update request for the low load region and the saturation region is not so much affected by the selection of input step size than that for the high load region.

Hence, the selection of step size of input request mean arrival rate is also depend on the need of how accurately we want to determine the optimum performance of the database server for high load traffic and therefore it plays a significant role in achieving a good model of the database server.

Thus, the selection of step size of input mean arrival rate also affects the efficient designing of a response time estimator and controller logic used for controlling the requests traffic.

6 Discussion on Limitations

Section 5 (Experiment section) provides a discussion on a set of experiments performed on the data tier that contains a database server. The whole idea is to measure the mean response time of the requests sent to the database server. This section provides discussion on the validity of the measured responses times.

The measurement of response time starts when the request has left the traffic generator and stops when the response of request is returned from the server. The time measured does not only include the response time in the database server but also the time in the network and also the time to process the network packets in the operating system.

The generation of traffic and measurement of the response time are performed on the same computer. The traffic generation needs processing power. If the processing power of the computer that measure the response time is not enough, then the measured response time might be larger than the actual response time.

The methods used for performing all the experiments also have some limitations. All the experiments assume that the default number of worker threads is equal to 30 and that the default number of external concurrent connections to the database server is equal to 30. However, the maximum number of concurrent connections available for the worker threads is configured to be 800.

The experiment results in this thesis do not contain any graphs for the number of worker threads more or less than 30 and the number of external concurrent connections greater than 30.

The worker threads which are actually Java threads are responsible for fetching the requests to the database server. If, in case, high traffic of the requests are generated from traffic generator and sent to the database server and at the same time, all the worker threads are busy in executing the requests at every predefined connections then the worker threads are unable to perform simultaneous fetching of those requests. Therefore, the requests arriving at the database server may not follow exponential distribution all the time even if the generation of requests from traffic generator follows exponential distribution.

In all the experiments, relations with smaller relation sizes (5 millions and 10 millions) are considered for InnoDB engines while relations having larger relation sizes (20 millions, 50 millions and 100 millions) are used for MyISAM engine. The reason behind this is that we cannot figure saturation regions exactly in practice and, the results look something unreasonable and not promising if we use relations having smaller relation sizes for MyISAM engine.

7 Conclusion and Future Works

The modeling of the MySQL database server is performed according to the dynamics of the mean response time for different loads (request mean arrival rates). The dynamic modeling of MySQL database server for given particular requests shows different behavior for different loads and also for different changed scenarios. These changed scenarios include change in any of the metrics such as types of requests, engine storage type, number of concurrent connections and relations sizes.

Hence, the model showing this behavior of the database server need to be considered when designing the database server for optimum performance. With this, we are able to determine the mechanism for maximum utilization of system's capacity while meeting all the SLAs (service level agreement). In this way, we are able to perform resources management in efficient way.

Future Works

This thesis does not discuss anything about the overload protection mechanism. However, there are many papers and reports related to mechanism for overloading protection.

Since we achieve a good model for the optimum performance of the database server, we can use this model in the future to determine appropriate methods for overload protection.

The step-controllers and PI-controllers (Proportional-Integral controllers) are the simpler controllers which have been used in application servers for control purposes. The implementation of these types of controllers has been discussed in a previously published thesis report written by Gustav Cedersjö [12]. However, there exist many sophisticated controllers applied on application servers for controlling database overload.

List of References

[1] E.F. Codd. A relational model of data for large shared data banks. Communications of the ACM, 13(6):377–387, 1970.

[2] Oracle9*i* SQL Reference Release 2(9.2), Part Number A96540-02, October 2002.

[3] "Sun Microsystems Announces Completion of MySQL Acquisition; Paves Way for Secure, Open Source Platform to Power the Network Economy" . *Sun Microsystems*. 26 February 2008.

[4] "Oracle Announces the Acquisition of Open Source Software Company, Innobase" . Oracle. Retrieved 2012-01-30.

[5] D. Bitton, D.J. DeWitt, and C. Turbyfill. Benchmarking database systems a systematic approach. In Proceedings of the 9th International Conference on Very Large Data Bases, pages 8–19. Citeseer, 1983.

[6] C. Turbyfill, C. Orji, and D. Bitton. AS3AP – a comparative relational database benchmark. In COMPCON Spring '89. Thirty-Fourth IEEE Computer Society International Conference: Intellectual Leverage, Digest of Papers., pages 560–564, 1989.

[7] A. Hagsten and F. Neis. Crisis request generator for internet servers. Master's thesis, Dept. of Communication Systems, Lund University, 2006.

[8] Apache Software Foundation. Jmeter. http://jakarta.apache.org/jmeter.

[9] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. Java™ Language Specification, The (3rd Edition) (Java (Addison-Wesley)). Addison-Wesley Professional, 2005.

[10] D.J. DeWitt. The Wisconsin benchmark: Past, present, and future. The Benchmark Handbook for Database and Transaction Processing Systems, 1, 1991.

[11] S. Marlow. Haskell 2010 Language Report, 2009.

[12] G. Cedersjö. Overload Protection for the Data Tier in a Multi-Tier Web Server System. M.Sc. thesis, EIT, LTH, Sweden 2010.

List of Figures

List of Tables

List of Acronyms

T : mean response time taken by MySQL database server for particular SQL request

λ : request mean arrival rate

5MSelectInnoDB: SELECT request to the relation of InnoDB engine type with size of 5 millions tuples

5MUpdateInnoDB: Update request to the relation of InnoDB engine type with size of 5 millions tuples

5MUpdateInnoDBOne: same as 5MUpdateInnoDB but the input step size is choosen as 1

5MUpdateInnoDBFive: same as 5MUpdateInnoDB but the input step size is choosen as 5

10MSelectInnoDB: SELECT request to the relation of InnoDB engine type with size of 10 millions tuples

10MSelectInnoDB15: same as 10MSelectInnoDB but the number of connections to the MySQL server is reduced to 15

10MSelectInnoDB10: same as 10MSelectInnoDB but the number of connections to the MySQL server is reduced to 10

10MSelectInnoDB5: same as 10MSelectInnoDB but the number of connections to the MySQL server is reduced to 5

10MUpdateInnoDB: Update request to the relation of InnoDB engine type with size of 10 millions tuples

10MSelectMyISAM: SELECT request to the relation of MyISAM engine type with size of 10 millions tuples

10MUpdateMyISAM: Update request to the relation of MyISAM engine type with size of 10 millions tuples

20MSelectMyISAM: SELECT request to the relation of MyISAM engine type with size of 20 millions tuples

20MUpdateMyISAM: Update request to the relation of MyISAM engine type with size of 20 millions tuples

50MSelectMyISAM: SELECT request to the relation of MyISAM engine type with size of 50 millions tuples

50MUpdateMyISAM: Update request to the relation of MyISAM engine type with size of 50 millions tuples

50MUpdateMyISAM25: same as 50MUpdateMyISAM but the number of connections to the MySQL server is reduced to 25

50MUpdateMyISAM15: same as 50MUpdateMyISAM but the number of connections to the MySQL server is reduced to 15

100MSelectMyISAM: SELECT request to the relation of MyISAM engine type with size of 100 millions tuples

100MUpdateMyISAM: Update request to relation of MyISAM engine type with size of 100 millions tuples

Appendix A

A.1 Caching

In MySQL, many clients may send requests for fetching the values from the single or multiple columns of the same tuple of certain relations of the database. Meanwhile, default caching time of results of requests depends on the type of database server used. The caching help to get prompt reply of the requested query which has been previously requested. This is because MySQL's query cache stores the results of previously executed requests until some valid time before it gets expired.

However, this caching of requests may affect our motive of getting exact response time needed for the execution of the requests since the response result set of requests are delivered directly from cache instead from the database server.

Hence, it is necessary to ensure whether caching of the requests is enabled or not. This can be performed by issuing following command in mysql command prompt,

```
mysql> SHOW VARIABLES LIKE 'query_cache_size';
```

It will return the following result

```
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| query_cache_size | 0 |
+------------------+-------+
```

It means that the query cache size is 0. This value ensures that the caching has not been enabled. By default, the query cache size is 0 so caching is disabled by default.

Furthermore, setting the query cache type variable equal to 0 also disable the query caching.

```
mysql> SET SESSION query_cache_type = 0;
```

A.2 Remote MySQL Server Connection

The remote accessing of the MySQL database server from user or client server is disabled by default due to security reasons. However, in many cases, the remotely accessing of MySQL database server from another client server or user is required. Also, to be able to connect the MySQL server remotely, IP based account is also needed and proper connection parameters that includes name of the host where server is running and username and password of the mysql server account is needed. There is default value for each connection parameters but these values can be overrided if necessary by using program options specified either on the command line or option file.

For doing this, following configuration is needed to be performed.

The connection to mysql server should be made:

```
$ mysql -u root -p mysql
```

The next step is to create a database or use the database if that database already exists. If new database called mindata needs to be associated with user root and remote IP 202.54.10.20, then, the following commands can be issued at mysql command prompt;

```
mysql> CREATE DATABASE foo;
mysql> GRANT ALL ON foo.* TO bar@'202.54.10.20'
IDENTIFIED BY 'PASSWORD';
```

Then, the next step is to logout from mysql command prompt and make sure that the port 3306 is opened.

By default, port 3306 is not opened and firewalled.

After making sure that the port 3306 is opened for mysql server the following command can be typed to check whether the remote connection of mysql server is established or not.

```
$ mysql -u webadmin -h 65.55.55.2 -p
```

Another method is the use of telnet command to test the connection to remote MySQL server and port 3306.

```
$ telnet 65.55.55.2 3306
```

After issuing these commands, we can check and verify whether the specific user has been granted with a privilege for accessing the MySQL server remotely. The following command is issued in mysql command prompt for this purpose.

```
mysql> SHOW GRANTS FOR 'root'@'localhost';

mysql> SHOW GRANTS FOR 'root'@'202.54.10.20';
```

Other statements that are used for verifying user privileges are shown below.

```
SHOW GRANTS;

SHOW GRANTS FOR CURRENT_USER;

SHOW GRANTS FOR CURRENT_USER();
```

Problem Encountered:

The port 3306 is blocked by default because of firewall which prevents the client program to access the MySQL server remotely. Therefore, the windows firewall needs to be configured through control panel and make exception for the port 3306. This port is used for connecting MySQL server.

The following steps are carried out before remotely connecting to the MySQL server.

1. Ping the remote ip address of server.

2. Show grant for specific user.

3. Check whether exception is made for port 3306 since this port is used by MySQL server.

A.3 Changing The concurrent connection limit

While installing MySQL, the default configuration of the MySQL server which is also part of installation process sets number of the maximum concurrent connections to the MySQL server equal to 100.

In Windows, this could be verified by looking the my.ini file .This file can be found in the default location mentioned below.

```
--defaults-file="C:\Program Files\MySQL\MySQL
Server 5.1\my.ini"
```

Alternatively, the maximum concurrent connections to the MySQL server can be achieved by issuing the following command in mysql command prompt.

```
mysql>show variables like 'max_connections';
```

The default output gives number of maximum concurrent connections equal to 100.

The maximum number of concurrent connections to the MySQL server can be increased or decreased whenever necessary. However, the maximum number of concurrent connections cannot be exceeded beyond 1400.

The following command can be issued in mysql command prompt to set the maximum concurrent connections to different value apart from default value.

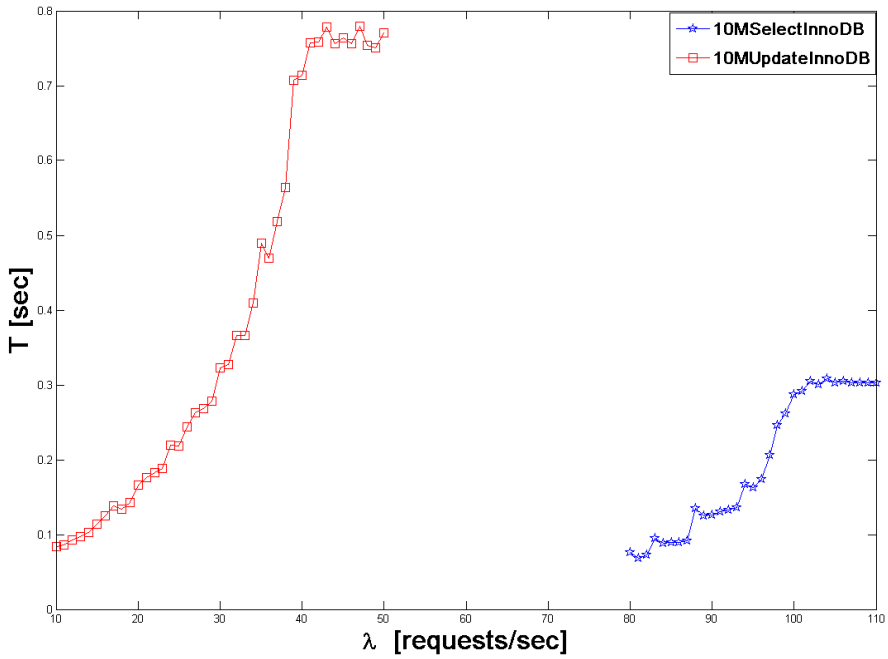
```
mysql>Set global max_connections=15;

mysql>Set global max_connections=250;
```

Alternatively, it is possible to navigate to the bin directly of MySQL installation and use the MySQLInstanceConfig.exe file to reconfigure the MySQL server directly. This will allow opening the MySQL Server Instance Configuration Wizard which when further proceeding allows to manually set the concurrent connections limit.

In many cases, it may be significant to limit the maximum number of concurrent connections to MySQL server that can be built. The reason behind this is that the MySQL server may run out of resources when there involves large number of concurrent connections to the server.

Appendix B

B.1 Comparison of 10M Select and 10M Update for InnoDB engine



B.2 Comparison of 20M Select and 20M Update for MyISAM engine