COMPARISON ON VARIOUS CONCURRENCY PLATFORMS WITH IO BOUND PROBLEM (CONCURRENT DATABASE INSERTION ON MULTI-CORE MACHINE)

BY

LOW BOON WEE

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF INFORMATION SYSTEM (HONS)

BUSINESS INFORMATION SYSTEM

Faculty of Information and Communication Technology

(Perak Campus)

APRIL 2011

LOW BOON WEE April 2010

UNIVERSITI TUNKU ABDUL RAHMAN

REPORT STATUS DECLARATION FORM

Title: **COMPARISON ON VARIOUS CONCURRENCY PLATFORMS WITH IO BOUND PROBLEM (CONCURRENT DATABASE INSERTION ON MULTI-CORE MACHINE)**

Academic Session: **JAN 2011**

I **LOW BOON WEE**
(CAPITAL LETTER)

declare that I allow this Final Year Project Report to be kept in
Universiti Tunku Abdul Rahman Library subject to the regulations as follows:
1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.

Verified by,

______________________	______________________
(Author's signature)	(Supervisor's signature)

Address:
489, TAMAN ASEAN
JALAN MALIM
75250 MELAKA

OOI BOON YAIK
Supervisor's name:

Date: 8Th APRIL 2010

Date: ____________________

COMPARISON ON VARIOUS CONCURRENCY PLATFORMS WITH IO BOUND PROBLEM (CONCURRENT DATABASE INSERTION ON MULTI-CORE MACHINE)

BY

LOW BOON WEE

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF INFORMATION SYSTEM (HONS)

BUSINESS INFORMATION SYSTEM

Faculty of Information and Communication Technology

(Perak Campus)

APRIL 2011

DECLARATION OF ORIGINALITY

I declare that this report entitled “COMPARISON ON VARIOUS CONCURRENCY PLATFORMS WITH IO BOUND PROBLEM (CONCURRENT DATABASE INSERTION ON MULTI-CORE MACHINE)” is my own work except cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidate for any degree or other award.

Signature	:	___________________________
Name	:	LOW BOON WEE
Date	:	8Th APRIL 2011

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisor, Mr. Ooi Boon Yaik whom has given me this bright opportunity to engage in the research field of Using Multi-Core to-do Multithreading on Database. I would also like to thank Mr. Wong Chee Siang as well for his guidance. A million thanks to both of them.

Special thank to those around me, my family members, and Wong Yoot Wen for their unconditional support and love, in standing by my side during hard and easy times. Finally, I must say thanks to for everyone's encouragement throughout the course of my study.

I would also thank all my friends whom had been there for me. Without them much of the work won't be possible.

When I asked for strength, He lifts my burden from my shoulders.

When I asked for knowledge, He provides me with what I need.

When I asked for wisdom, He is there constantly with me.

Without Him nothing is possible, I would like to thank God for His faithfulness and for being with me at all times and providing all my needs. Thank God.

ABSTRACT

The advancement of multi-core processors and database technologies have enable database insertion to be implemented concurrently via multithreading programming. In this work, we evaluate the performance of using multithreading technique to perform database insertion of large data set with known size. The performance evaluation includes techniques such as using single database connection, multithreads the insertion process with respective database connections, single threaded bulk insertion and multithreaded bulk insertion. MySQL 5.2 and SQL Server 2008 were used and the experimental results show that for larger datasets bulk insertion of both databases can drastically be improved with multithreading.

TABLE OF CONTENTS

LIST OF FIGURES

Figure Number **Title** **Page**

LIST OF TABLES

Table Number **Title** **Page**

CHAPTER 1 INTRODUCTION

As the amount of data increase and so goes with the processing power, this created a need for a better solution into INSERTING huge amount of data into the database. Therefore in this project a research is being done on what is an efficient way into doing bulk insertion on different database engine. Multi-threading is being used as a tool to help improve the performance.

1.1 MOTIVATION

Most CPU today comes with more than 1 core, for example in an Intel Xeon processor has 12 logical cores in a single micro-processor with multi-threading enabled. In a high end machine it has over 64 cores (Ryan Johnson, Shore-MT: A Scalable Storage Manager for the Multicore Era, 2009). With such amount of computational power, processing time is being reduced tremendously with the use of multi-threading. The issue we looking here is, does multi-threading the Data Access Layer (DAL) helps reduce bulk insertion cycle time? With size of data is ever growing, and this has made data storage a challenge. Random Access Memory (RAM) was once use to be a major issues, but now with RAM in access of over 128 GB (PowerEdge R905, 2010) there is more than sufficient.

With the fall in price of multi-core processors, it has made high-performance processors affordable to all. This has spark the question if multi-threading would help improve database insertion performance. It is known that database insertion is slow compared to how data size is growing. From my research, there are a number of factors that contribute to this limitation; they are magnetic disk bottleneck, processing power, RAM, and database architecture. With so much available resources available, one server-side processing that could take advantage of this is database engine. From past research, it is possible to improve database insertion speed by using multi-threading therefore looking for the efficient point in doing bulk insertion on different database engines would give a clearer picture on performance improvement. The problem I would like to answer is does multi-threading improves database insertion.

Let’s take for exam the Large Hadron Collider (LHC) in Switzerland is able to generate 100 Mbps of data and it has to be stored into a database constantly. The ‘raw’ event data thus emerge from the detector’s electronic data acquisition (DAQ) system would have to be stored in a database. By the time the project reaches maturity, its projected data size would be in excess of 100 PetaBytes (Julian J. Bunn, 2007). Such event have encourage me into looking in the option of multi-threading. Standard SQL insertion would not able to insert those data fast enough and would create a backlog.

Many has proven that multi-threading does improve performance but how much is the question? By some calculations based on MySQL help directory on speed of the INSERT statement, the factors that contribute to performance speed are shown. The following are the factors that are involved in an INSERT statement. The numbers in brackets are approximate proposition of time required (Speed of INSERT Statement, 2010):

- Connection (3)
- Sending query to server (2)
- Parsing query (2)
- Inserting row (1 multiply size of row)
- Inserting indexes (1 multiply number of indexes)
- Closing (1)

Single Threaded

Time = Connection + Sending Query to Server + Parsing Query + Inserting Row + Inserting Indexes + Closing

= 3 + 2 + 2 + (1 X 5) + (1 X 5) + 1

= 18

Double Threaded

Time = Connection + Sending Query to Server + Parsing Query + Inserting Row + Inserting Indexes + Closing

= 3 + 2 + 2 + (1 X 5) + (1 X 10) + 1

= 23

Figure 1-1: Calculation of INSERT statement time

From the calculation in Figure 1, we can see the improvement in terms of time when the INSERT statement is being threaded. There is an improvement of 28% when it’s double threaded. This has motivated me into looking at different database engines to see what multi-threading does to it then building a DAL to would select the insertion method and number of threads to create according to the amount of data and also database engine.

1.2 PROJECT INTRODUCTION

In this project, the aim is to discover if multi-threading does help improve the efficiency of database bulk insertion. Then it would be is searching for an optimal insertion method for MySQL 5.2 and SQL Server 2008 Enterprise according to the number of data to be inserted. With all the data gathered, an intelligent DAL would be developed. The DAL would select the insertion method to be used according to the number of data to be inserted and also database engine that's it will connect to.

Tests were done on a single database in each engine which contains a table with two columns. Data inserted are 302 characters on one column and the other is an auto-generated numerical index. Characters include symbols, space, commas, numeric and alphabets. This would show if there is a relation between database connections, threads, SQL command, database engine, system utilization and many other factors that we would be encountering during the analysis of experiment results.

To benchmark the performance, sequential SQL (Structured Query Language) with transaction and without transaction are being used. The sequential method would then be compared to bulk insertion module that has been developed by individual vendors. All methods would be threaded in the range of 1 to 8 threads. Tests were done on the same machine in the same environment to maintain consistency. Each test is repeated 5 cycles to maintain consistency. Some are repeated more to reduce anomalies in the results due to certain factors.

In this project, to the goal is to prove that multi-threading does improve bulk insertion efficiency. Here we would also like to answer that the theory does not apply to all database engines. Past research has shown that multi-threading implementation has solved processor bound tasks (Broberg, 2000), but it is not confirm that it would solve I/O bounds. From reviews from multiple forums, multi-threading does not solve I/O bound problems but to what extent this is true? Due to the how the database engines are designed and developed, the implementation of multi-threading on them would be different. The emphasis on this project is not to prove which database engine is the fastest and what method is best to use. Instead it is to prove that multi-threading does help in improving database bulk insertion performance.

1.3.1 ADVANCEMENT IN DATABASE ENGINE

The advancement in database engine technology has made it possible to inject a huge amount of data into a database in a relatively short period of time. This could be done concurrently to reduce the time taken and the engine today supports multiple concurrent connections. Let's take for example Microsoft SQL Server 2008 is able to handle over thirty thousand connections at the single time. It's also able to support 4096 INSERT statement per columns at a single time. Database engine are able to support eight concurrent applications running (Maximum Capacity Specifications for SQL Server, 2010) (Rubbelke, 2008). This shows the amount of payload the database engines are able to support and its capabilities in handling concurrent bulk insertion.

1.3.2 WHAT IS I/O BOUND?

Most of the modern operating system (OS) is capable of supporting multi-core processor which has the capability to process a huge amount of data concurrently. This allows more work to be done per cycle time at all level of platform (Moore, 2008). This is another issue to this; the amount of data which such processing power generates has to be stored somewhere quickly. But another major issue will arise is the input/output bound which machines today are facing. Such I/O bound could be on the magnetic disk, memory or the data flow from the application to the database.

This is a brief technology update on magnetic disk technology that has because of the disk I/O bound. Magnetic disk or flash memory is fast expanding but the transfer rate of such storage medium has almost reached its maximum performance level. The development for storage medium is not growing as fast compared to processing power. Many developers believe that disk technology has changed dramatically. But sad to say, that is false. Only three things have actually changed since 1970's:

- **Large data buffer size: -** Today DBA can cache a much larger portion of data in a block and reducing disk I/O.
- **Disks with on-board cache**: - Almost all of today's hard disks have an on-board RAM cache to hold data going in and out of the disk which is normally known as data blocks.
- **RAID (Redundant Array Independent Disk)**: - The randomizing of data block using RAID 0 + 1 has eliminate the need of hard disks load balancing

system, by scrambling data blocks across the disk spindles, for example in Oracle 10g, the Automatic Storage Management (ASM) feature requires SAME disk “Stripe And Mirror Everywhere”, which is essentially RAID 1 + 0 and RAID – 10.

However with all this advancement in hand to spare, DBA must still remember that I/O issue is still the major concern and many are trying to understand how to maximize the performance of their I/O issues (Zukowski, 2005). This problem has brought much research to be done on how to improve multithreading processing methods and new methods are being implemented on databases to increase its data injection to a much higher I/O rate. What we are trying to solve here is something that is not widely researched as I was not able to find many journal paper on the internet which research on this area.

When you multi-thread an application, the other problem would arise is where the processor is not capable of processing the amount of data and this creates something called CPU-bound. The CPU (Central Processing Unit) has reached its limit. This might also cause a bottleneck at the CPU cache where the data is going out of the processor. This will subsequently cause a major drop in performance level. The other problem is where a huge amount of data is sitting in the RAM (Random Access Memory) waiting to enter the database, this would then clog up the RAM and cause a huge decrease in the machines performance level. If the application has a distributed processing system then network I/O throughput would also be the problem. This is few of the main side effects of not threading properly and when I/O is in the way.

1.3.3 WHAT IS MULTITHREADING (MT)

A thread is a sequence of control within a process. A single threaded process follows a single line of controls while process is being executed. A multi-threading process has several lines of control; therefore it’s capable of processing several independent processes at one time. When multiple CPU is available to be used, those independent actions could be executed parallel according to the number of threads at your disposal (Java on Solaris 7 Developer’s Guide, 1998).

Multithreading is running multiple processes concurrently. This allows processes to overlap each other and being process concurrently. This improves the efficiency substantially and this is proven in a journal by Scott R. Taylor 2008 where they used the Fibonacci function across multiple threading method and also different machines (Scott R. Taylor,, 2008). One processor would have multiple threads and this threads act as trunk in the processor where data flows thru and being process. One thread can handle one or multiple processes concurrently. One thread can also hold multiple database connection but this depends on the database engine. Current multi-core processes have multiple cores and would have multiple threads on a single core. This would then allow a huge of information flowing thru at a single point of time and able to support multiple processes concurrently. Processors or I/O are able to overlap and need not wait for processes to be completed before the next process can enter the CPU to be process. Multiple processes are able to run parallel and this would then reduce processing time.

These are the advantages of threading an application:

- Improves the response time needed to complete a process.
- Reduce the processing time as multiple processes are being processed simultaneously.
- Making full use of available resources on the CPU.
- Using lesser resources as process are being completed in a shorter time.
- Less overhead is being required to manage the queuing processes or inter-processing items.

1.3.4 DATA ACCESS LAYER (DAL)

DAL is a set of classes with functions for reading and writing to a database or other data storage medium. It does not contain any business logic for the application or user interface element. It’s a background worker that interacts between the application and the database. It’s a part of a multi-layer application design that would normally include items as below (J.D. Meier, Chapter 12: Data Access Layer Guidelines, 2009):

- A **User Interface Layer** (UI) which contains screens and user interface components.
- A **Business Logic Layer (**BLL) which contains the business rules for the application.

- **Data access logic components** are abstract logic that are needed to access underlying data stores. Doing so would then centralize the data access function. It would then help make the application easier to configure and maintain.
- Data helpers/utilities are functions and utilities assist in data transformation and data access in the layer. It contains specialized API and routines that is designed to improve data access efficiency and reduce the need to develop logic components and service agents in the layer.
- Service agent is where business component uses functions that are exposed by external services. The service agent isolates your application from the idiosyncrasies of calling diverse services and additional services can be provided.

Features on a DAL which would be implanted in this project would just be the INSERT operation. The DAL here would be multithreaded and able to support multiple connections to the database engine. The DAL that we would be implementing would be portable across different database engines. It would be an API to be imported into applications. According to codefeatures.com, DAL is the most efficient and reliable technique for database connectivity.

Figure 1-2 shows where the DAL sits in an application.

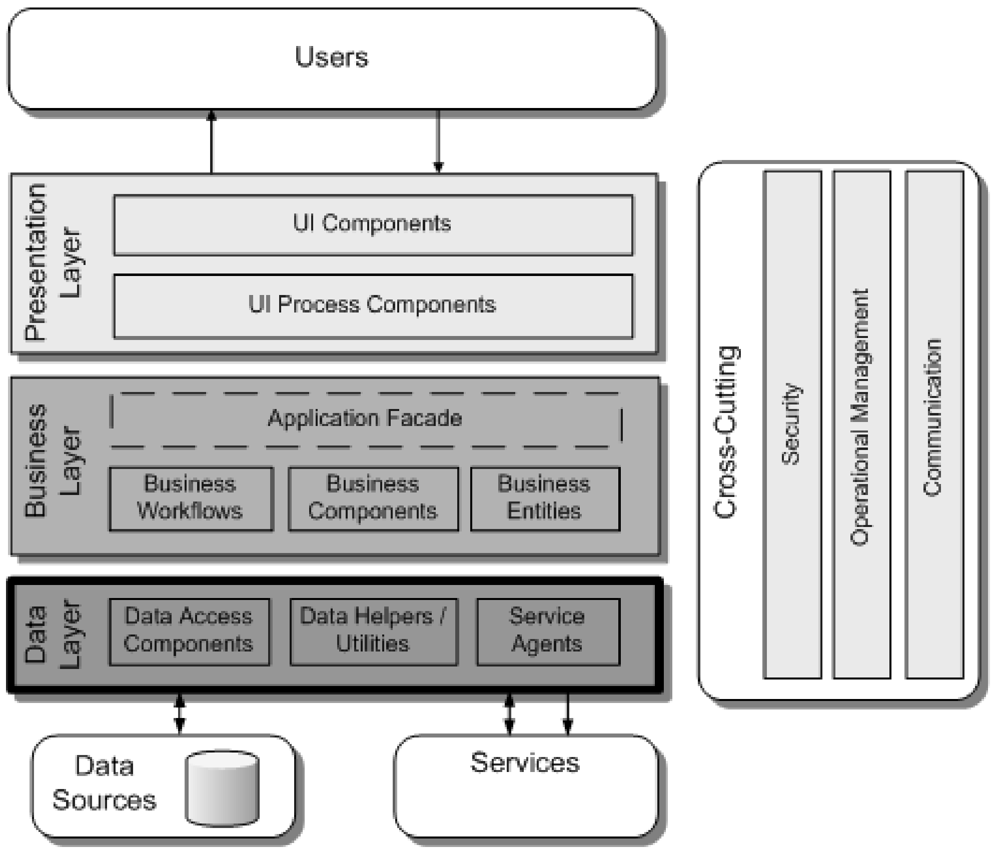


Figure 1-2: A typical application showing where the DAL and other components (J.D. Meier, Chapter 12: Data Access Layer Guidelines, 2009).

DAL is a plug-in that connects the database that the application supposed to connect to and perform CURD (create, read, update, and delete) operations. Its task is also to start and close connections to databases, it can also be known as database connection manager where connection pooling would be implemented or other connection management algorithms. We are planning to create multiple connections and threading the DAL to improve its performance. This then would allow it to inject data asynchronously into the database. The number of connection to be created is would also be researched. We would be looking for the most efficient number of connection to be set into our DAL.

1.4 METHODOLOGY IN BRIEF

Tests were conducted on sequential insertion with and without transaction on 1, 2, 4, and 8 threads. Then it would come compared with bulk insertion modules from the respective database vendors. Tests on different number of threads would be conducted as well. The database engine used would be MySQL 5.2 and Microsoft SQL Server

2008 Enterprise. The test data used would range from 1 to 80,000 rows with 302 characters on each row. Each test would be repeated five cycles and time were taken as a benchmark. The same data is being used on both database engines and it would be done on the same machine to control the environment. From the elapsed processing time, an average it is taken. A test data builder is being developed to generate the 302 character test data in each row.

The next test was on CPU (Central Processing Unit), RAM, and hard-disk I/O. The same data, database and program is being used to insert and the CPU, RAM and hard-disk I/O is being captured. Sampling method is being used to analyze the data. Each sample is taken at every half a second interval. The average is taken and used for analysis. Samples would be saved in a flat file and then processed by a program to get the average and records it. All tests were being done on the same machine. For this test, two types of methods were begin used and for further details refer to section 3.2.5.

1.5 OBJECTIVE

The objective of this project is to find a better solution into storing a huge amount of data into a database in the shortest time possible with the use of multithreading. In this project, we aim to make full use of the database engine's capabilities by running multiple insertions simultaneously.

Another objective here is also to see the relationship between CPU, RAM and hard-disk I/O utilization. This would show the role each component plays in multi-threading an INSERT function. From here a relationship between the database engine and the three factors plays towards multi-threading is concluded.

1.6 PROJECT SCOPE

The following are the platforms, programming languages and database engines that we would be doing our research on:

- OS platform: Windows XP SP3 (32-bit)
- Programming Languages: C#
- Database Engine: MySQL by Sun Microsystems, Microsoft SQL Server 2008
- Threading Methods: would be discuss in the Methodology chapter.

1.7 PROBLEM STATEMENT

In this project, we would be looking for the best method to insert a huge amount of data into the database in the shortest time possible. Many applications with database engine works in different ways, but no one knows what combination would suit best the environment that that we would be experimenting on. Many currently ask the question "What's the best method to thread the DAL and which database is the most efficient?" Another question that arises is weather to leave connections to the database open throughout the process or each process has its own connection?" the other big question would be "What threading method suits which database engine and what the best design and implementation?" The question on what relationship does CPU, RAM and hard-disk I/O utilization has on database bulk insertion speed would be answered here. Currently the community doesn't exactly know what are the consequences multi-threading the insertion process when the machine is processing some task.

The search for an optimal method to thread each database engine is being done to find what they work best with. The problem is all databases are built in different architecture and will multi-threading suit its environment. There is not much information on such topic. Therefore a study on which threading and insertion method suits a particular database is being done.

Many DAL today are not multithreaded and we would like to create our own DAL which is multithreaded and to fix the database engine we would be experimenting on and also our task to insert a huge amount of data into the database in the shortest time possible. A search for a better way to do a DAL for bulk insertion is being done and in the end of the project a smart DAL was being developed.

1.8 CONTRIBUTION

The contribution of this project is information about which database matches which threading method to be used for an application which needs to insert a huge amount of data concurrently or in the shortest time possible. A comparison on different insertion and threading methods were being tested to find an optimal solution. It would act as a guideline to database developers when they are dealing with database bulk insertion.

The other contribution would be tocreate a multithreaded DAL to insert a huge amount of data into the databases we are experimenting on and if possible, an API for this DAL will be created. The DAL would support bulk insertion on two different database engines and accept text files to process the insertion.

In the end of the project, a table is being generated to act as a guideline to database developers into threading database bulk insertion and methods to use according to the number of rows to be inserted.

1.9 TECHNOLOGY INVOLVED

1.9.1 Technology Involved

In this section technology involved in this research is being discussed. it would be covering database engine used, threading methods, programming language and operating systems (OS) being used.

1.9.2 Database

I would be using Microsoft SQL Server 2008. The SQL Server is a very advanced database which is being widely in industry. This comprise of SAP ERP (Enterprise Resource Planning), Camstar which uses it for their MES (Manufacturing Executing System) (SQL Server 2008 Product Information), this are the few examples of implementation and scalability of the database engine. Finally we would be using MySQL which is owned by Oracle. Why we are using this database engine is because it’s the most popular database and this is proven by the number of download it has per day (MySQL Overview). It has all the feature at a fraction of the cost and this makes it a good platform to test on (MySQL Overview).

1.9.3 Programming Language

For the programming languages, C# is being used on Visual Studio 2010. which is one of the most common languages used today.

1.9.4 Threading Method

Manual threading is being used in this project. Creation and spawning of threads are being done manually. For more information refer to the methodology section. Due to time constrain, only this threading method is being used.

1.9.5 Operating System

For the operating system, Microsoft Windows platform is being used in this project. It is one of the most common platforms for PC (Personal Computers) users these days.

The OS used is Windows XP thru out the research process to maintain a controlled environment. Due to time and resources constrain, only one operating system is being used.

CHAPTER 2 LITRETURE REVIEW

2.1 Introduction

Businesses today generate a huge amount of data which has to be stored as quickly as possible. This challenges database developers to look into ways of storing all this data. Let's take for example, in a stock exchange environment. Stock price changes constantly with hundreds or thousands of trading being done every minute, stock price has to be updated constantly and all this data has to be stored. Let's assume all this data has to be stored every 3 to 5 seconds and this has to be multiplied with the number of stocks that's being traded. This number could be running into the thousands or not hundreds of thousands every second. In-order to store all this data efficiently, a quick and efficient way has to be used and multi-threading is one of the many options. The other example of the use of such database insertion architecture is being used is in a scientific research environment as mentioned in Chapter 1 (Introduction). With the advancement of multi-core processors and database engine, which is capable of handling huge amount of data processing, connections, transactions and etc. this pose a great potential in developing a DAL that's able to cater to such environment needs. With the advancement in database engine insertion modules by vendors, it has minimized all the trouble into developing a bulk insertion DAL. Examples of such modules are .Net 4 Framework by Microsoft and MySQL Connector .Net 3.2.1 by Oracle. Such modules supports bulk insertion and it could be optimize to give better performance.

Threading is a meticulous task where creation and killing of threads has to be planned and done correctly. Managing threads is a very important task in multi-threading, if not done properly it will cause a lot of overhead and wastage.

From findings, current multi-threaded DAL's are being developed to application bound, hardware bound, database bound and still in the research stage. It also does not take into consideration the number of threads to be used according to the number of data to be inserted. In this project, I would like to answer this few questions. Would multi-threaded DAL bring a single core machine to a standstill if implemented? What is the performance difference when being implemented on a dual core machine compared to a quad core machine? Does magnetic-disk IO and RAM play a role in insertion performance? The DAL is being designed to take into consideration of how many rows of data are to be inserted as well as what database engine is being used. From there it would decide the insertion method and number of threads to be used to perform the task.

Much research had been done on almost similar topic, for example a group of researcher from Camegie Mellon University United States done similar test with four open source database engines (Shore, KerkeleyDB, MySQL, and PostgreSQL) and found that all of them are highly scalable (Ryan Johnson, Shore-MT: A Scalable Storage Manager for the Multicore Era, 2009). From the data that was presented, two database engines (Prostgres, MySQL) hit its optimal data throughput performance at 24 concurrent threads.

From forums and blogs, it is known that this project has a great potential in this field of searching for a more effective or efficient way of doing bulk insertion with the use of multi-threading. From findings, it is found that there is not much research being done in this particular field but it does show great potential in its related capability. The big question that most would ask today is which method works well with which database engine on how many threads? There is no clear answer for this question, from my research it’s related to hardware, magnetic disk IO and also the software. Previous works have been using all kinds of method but there are no detailed specifications on how it’s being done.

Physical disk I/O is one of the main issues in bulk insertion. With magnetic hard disk rotating at 15,000 rpm, with I/O transfer rate of approximately 78 Mbps (Seagate Barracude 7200.10 SATA 3.0Gb/s 320-GB Hard Drive, 2010). From the claim by the manufacturer, it is know that the hard disk is able to support a maximum transfer rate of 31 Mbps. This makes the physical disk I/O not an issue at this point of research. RAM has not been the main issue as it’s expandable up to 128 GB (PowerEdge R905, 2010). These allow a huge amount of RAM at disposal when needed.

2.2 WHAT IS MULTI-CORE PROCESSOR?

In recent years, multi-core processors based their chip on multi-threading/processing architecture, which has multiple CPU on a single chip. The fact that multi-core processors are able to give better performance to cost ratio, it has become the building blocks for high performance applications to be developed in recent years. Application with requires high computations power has benefited from this innovation, this allows tasks to be running parallel instead of serial (Verenkar, 2010).

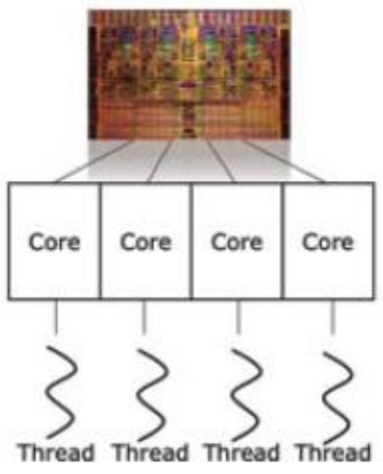


Figure 2-1:One processor can have multiple cores and usually each core has one thread. (Granatir, 2009)

Central Processing Unit (CPU) has continuously advanced in silicon process technology from 108nm to 65nm which is on Intel Pentium 4 processor. For the Pentium 4 processor starts the multi-threading era with hyper-trading technology. Then it advanced from 65nm to 45nm which is on the Intel Core 2 Duo processor. This is where the multi-core technology comes into the picture. For the core-I series processor from Intel they are using the latest 32nm technology. This technology can hold up to 6 cores on one die. To illustrate, 6 die on a processor is equivalent to 6 individual processors with the same clock speed running concurrently (Intel Processor Roadmap, 2010) .

The graph below shows the resource imbalance effect. Multi-core processing is limited to its inability of existing technology to make full use of the allocated system resources. Multi-core processing also has its own problems, for example if a single system resource is being scaled in an imbalance manner, it may exhaust resources and cause application to take the consequences. This may result in sub-optimal performance and lower user application performance. The graph below is being benchmarked using VMWare’s VMmark clearly illustrate how scaling disproportionately can affect the performance of the processor. Referring to the graph, we can see that a single 16 core system yields approximately 30% less aggregate processing power compared to a 4 core processer running the same world load (Marchand, 2008). This comes to proves that even with multi-core, we have to thread our application carefully can taking into consideration background processes that is running on the machine or other sub processes that is running concurrently. Threading overhead is also an important factor to take into consideration when you thread an application to maintain optimal performance.

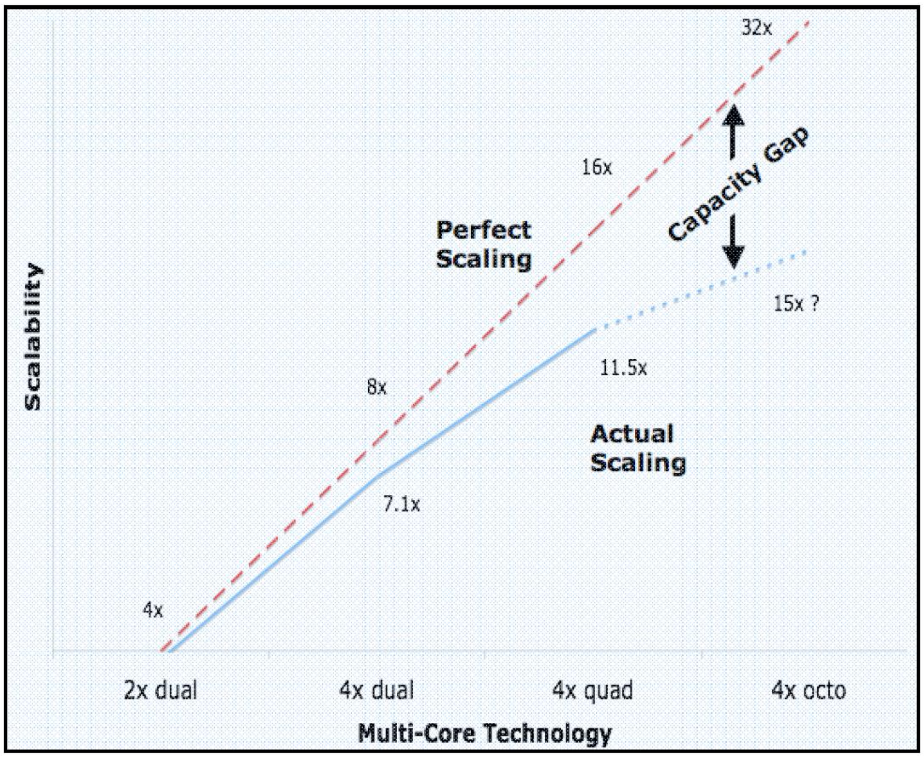


Figure 2-2: Performance increase as numbers of cores increases (Marchand, 2008).

2.3 WHAT IS MULTITHREADING?

Multithreading is running multiple tasks concurrently or parallel, this allow task to be completed faster compared to traditional method of arranging the task in a serial format. The number of threads available depends on the processor and also the Operating System (OS) to allow the threads to be called into service. Most OS today supports multitasking or multithreading very well. With the advancement in multi-core processors and the reduction in price of the product has allow more and more research to be done in the field. Everyone wants to multithread their application but if it's not done correctly, your users would be there asking "Isn't this application supposed to perform better compare dot the previous version?" or "Why has this application cause my other application to be on stand still?" and many other question as their machine start to stand still. This is because the application has not been threaded properly and is consuming too much processor power and is not allowing the other applications to run smoothly. There is a things to multithreading which is, if it's done right, it will work great but if not it will kill your application. If threading is overdone, it might just cause the application performance to drop.

Before threading, it is important to look at what are the application's environment and its usage. Then from there manage some thread overhead as it is proven costly to the performance (referring to section 1.1). Thread overhead will cause a bottleneck and that is where the performance will drop drastically. You can have the application performing at breakneck speed at the start but once it hits the bottleneck, it will jam the whole system down. Example of bottleneck is, for example you have 4 threads and the I/O only allows 2 connections. The other 2 threads has to wait for the task to be completed before being allow to access the I/O and till will cause a bottleneck.

	Single Processor Machine	**Multiprocessor Machine**
Processing Throughput	Low	High
Concurrent Task	One	Depends on processor
When call ThreadStart()	Does not start until there is available thread	May or may not start immediately depending on processor activity
Race Condition	Occurs when allowing another thread to reach a code block first	Occurs when task on different thread race to reach a code block first

Table 2-1: Comparison on single and multicore processors (MSDN Library (Threading), 2010)

There are two very famous threading strategies which are called OpenMP and Posix Thread (Pthread). OpenMP was built to be portable and has a compiler in it. It practices fork-join programming model. Meanwhile PThread is a library itself and crafted to provide optimum run-time library which support fork-join programming model (Verenkar, 2010).

Multithreading is divided into two categories which are explicit threading and implicit threading. Explicit threading is where the threading libraries require the programmer to control all aspects of threading, includes creating threads, assigning tasks, synchronizing or controlling intervention between threads, managing shared resources and other threading works. Examples of explicit threading are Windows thread, Pthread, C# thread, and Java threads. Implicit threading is where the library would control everything from creating threads, assigning tasks, managing threads, synchronizing threads and all other threading works. The library will do all the work for the programmer. Examples are threadPool, asynchronous threading method, Intel Treading Building Block (Intel TBB) and OpenMP.

2.4 PARALLELISM IN MULTITHREADING

Parallelism is running multiple tasks concurrently. This involves multiple threads or just a single thread. Usually multithreading is done on multi-core processors which has multiple threads at its disposal. Creating new task on a thread is called ‘spawning’. Usually each thread would perform a task. With 4 threads available, we are able to perform 4 task concurrently. The diagrams below would illustrate the scenario that would allow you to better understand what parallelism is in multithreading.

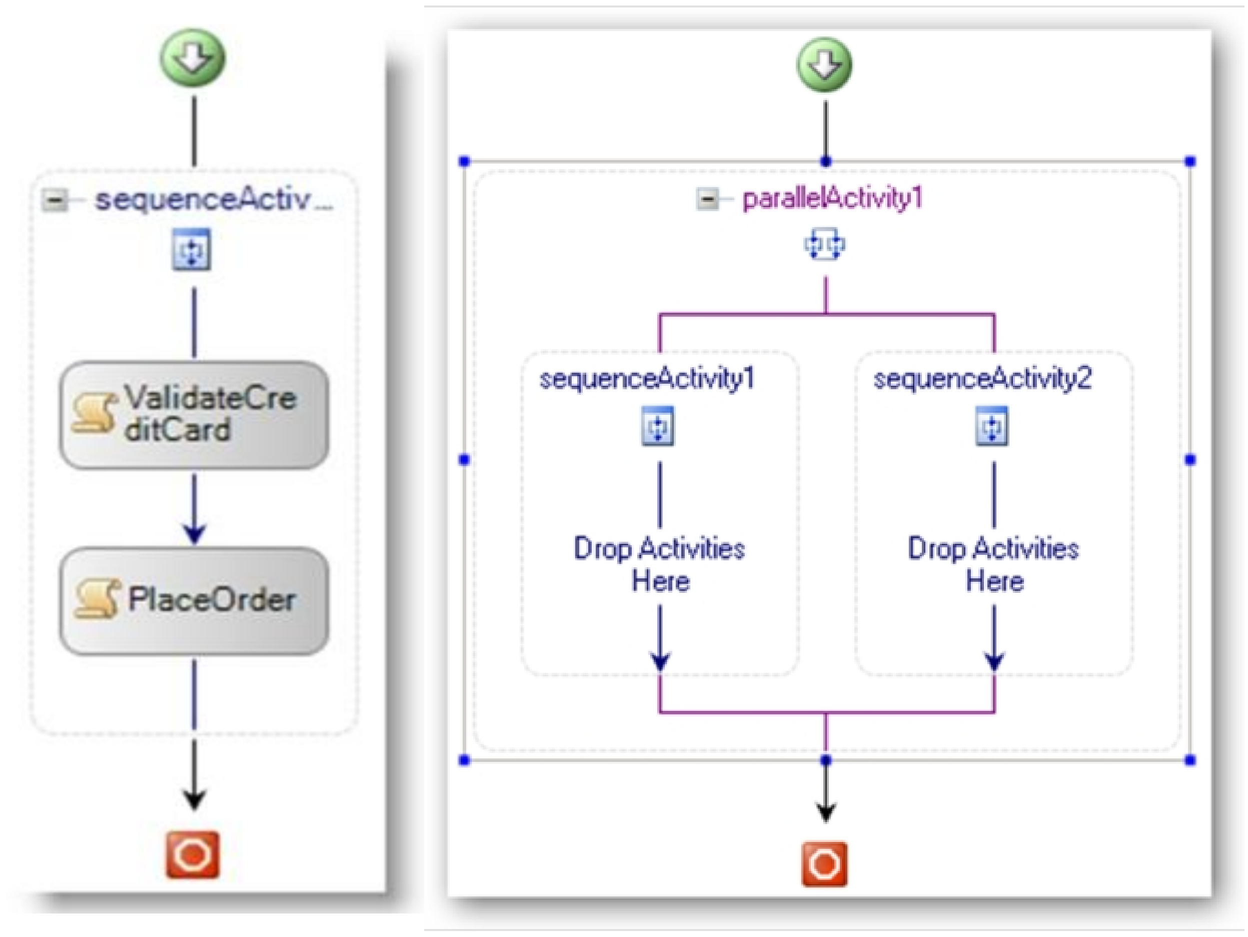


Figure 2-3: Left image illustrates serial programming. Right on the right illustrates parallel programming. (Vadaparty, 2008)

Threading is being divided into 2 categories which are implicit threads and explicit threads. Implicit threads are where the already available threading method would take care of everything from thread creation, thread killing, connections and thread management. While explicit threading method is where the developer has to control over when to create a thread, when to kill it, manage the thread and also other threading involvement. Figure 2-4 shows the examples of available technologies for both threading methods.

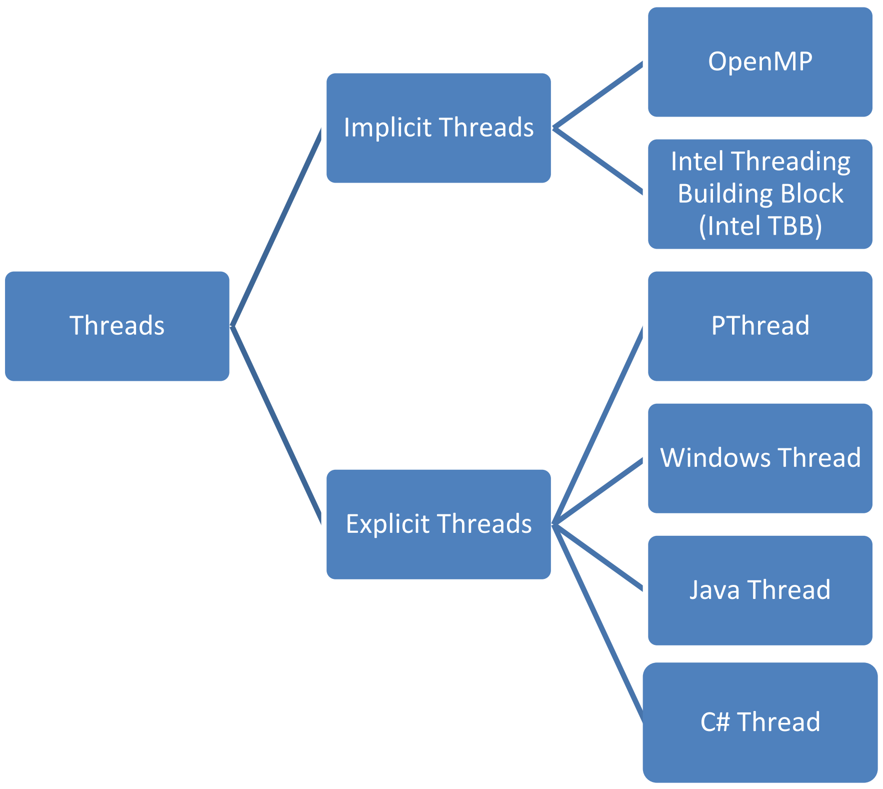


Figure 2-4: Thread Classification.

Following are the implementation of threading comparisons.

Challenges for parallel programming	**Windows* threads**	**OpenMP***	**Intel® Threading Build Blocks**
Task level		**x**	**x**
Cross-platform support		**x**	**x**
Scalable runtime libraries			**x**
Threads' Control	**x**		
Pre-tested and validated		**x**	**x**
C Development support	**x**	**x**	
Intel® Threading Tools support	**x**	**x**	**x**
Maintenance for tomorrow		**x**	**x**
Scalable memory allocator			**x**
"light" mutex			**x**
Processor affinity	**x**		Thread affinity

Figure 2-5: Threading Comparisons

Effect Multithreading on Database

To prove that multithreading would improve the performance of the database and also data insertion, following are a few test results which had been done over the years showing the improvement multithreading has contributed to the performance increase in database.

This test is being done in Microsoft by Akash Verenkar to introduce the .Net Framework 4 which support multithreading with database. This test compares sequential processing and data parallelism. The machine the test was being done has an Intel® Core™ 2 Duo CPU TZ7500 @ 2.20 GHz with 4.00 GB of RAM (Random Access Memory) running on Windows 7 Enterprise (64-bit) OS (Operating System). (Verenkar, 2010)

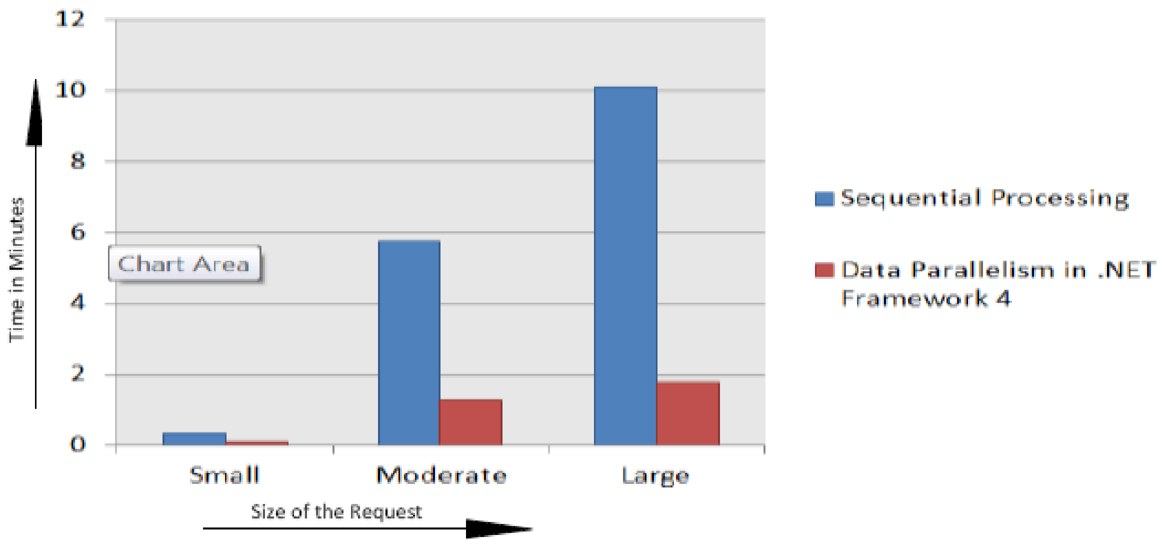


Figure 2-6: Shows the results from above mentioned two approaches (Verenkar, 2010).

From Diagram 2, we are able to see that there is a major improvement in processing time when the application is being multithreaded. Developers should look into where they are able to implement parallel programming and make sure your concurrency does improve the performance and has dependency amount tasks. Threading has to be done in a proper manner and much testing has to be done before the application is being rolled out to avoid bottleneck. PLinq, TPL and .Net 4 is being used to perform parallel programming and to reduce the complexity of code. (Verenkar, 2010)

In this paper they didn’t describe the best number of threads to create with the number of connections. It also didn’t answer weather how portable the application is. What if the application is being implemented on a single core machine? Would it perform as in a multi-core machine what is was being tested on. The method that was being presented only focuses on SQL Server 2008 and what if the user’s environment is using Microsoft Access 2010 or MySQL 5.0. Then such methods might not be feasible in such environment.

The next research is being done in Carnegie Mellon University by Ryan Johnson, Ippokratis Pandis, Nikos Hardavellas, Anastasia Ailamaki, and Babak Falsafi. The title of the paper is called Shore-Multithreading: A scalable Storage Management for Multicore Era. In this paper, they discuss that database storage management have long been efficient in handling multiple concurrent request. With the arrival of multicore chips in the market, more threads could be created and being executed concurrently. The benchmark is being done on four popular open source storage managers which are Shore, BerkeleyDB, MySQL and PostgreSQL. The examination of bottlenecks in various storage engines and the conclusion was Shore-MT has superior scalability. Experiment was done based on data throughput going into the database against concurrent threads. Diagram below illustrates the experiment result. (Ryan Johnson, Shore-MT: A Scalable Storage Manager for the Multicore Era, 2001)

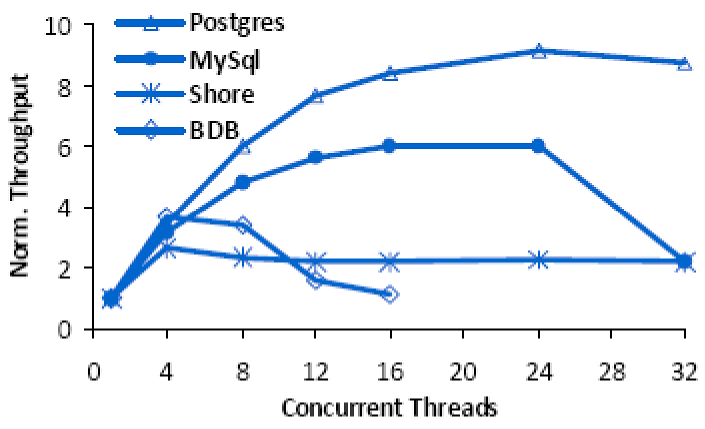


Figure 2-7: Scalability as a function of available hardware contexts. (Ryan Johnson, 2001)

The conclusion of this paper goes to show that database storage manager has to offer multithreading to achieve high performance user demands. From the experiment results, we can conclude by saying there is still room to improve the system and identifying bottlenecks that inhibit scalability would then improve the multithreading performance. It also points out that there is much more research to be done in the field of multithreading with database for there is much more bottlenecks and this paper would act as a reference to future development. (Ryan Johnson, Shore-MT: A Scalable Storage Manager for the Multicore Era, 2001)

In the paper above, they didn’t compare the optimal number of connections to be created and they didn’t use bulk insertion. In this paper it also didn’t specify what type of data is being inserted and also the machine used to generate the experiment results. Since 2001 when the paper was being published, there is newer and much more efficient threading methods and database being developed and this stand to have even better chance of creating are even more efficient way to thread a database to improve its performance.

From the research of Özsu (1991), distributed and parallel DBMSs are a reality today. It enables natural growth and expansion of database on simple machines. Parallel DBMSs are one of the most realistic ways into meeting the performance requirements of application which demands significant throughput on the DBMS (Valduriez 1. M., 1991).

Based on David J. DeWitt (1992) research, parallel processing is a cheap and fast way to gain signification gain in performance in database system. Software techniques such as data partitioning, dataflow, and intra-operator parallelism are needed to be employed to have an easy migration to parallel processing (Gray, 1992). The availability of fast processors and inexpensive disk packages is an ideal platform for parallel database systems.

According to P. Valduriez (1993), parallel database system is the way forward into making full use of multiprocessor architectures using software-oriented solutions. This method promises high-performance, high-availability and extensibility power price compared to mainframes servers (Valduriez P. , 1993). Parallelism is the most efficient solution into supporting huge databases on a single machine.

In a research to speedup database performance, Daniel Haggander (1999) shows that by multi-threading the database application it would increase the performance by 4.4 times than of a single threaded engine (Lundberg, Multiprocessor Performance Evaluation of a Telecommunication Fraud Detection Application, 1999). This research was done to support a fraud detection application which requires high performance read and write processes. Therefore they found that by increasing the number of simultaneous request would speed up the process (Lundberg, Multiprocessor Performance Evaluation of a Telecommunication Fraud Detection Application, 1999).

In 2005, Jingren Zhou shows that there is moderate performance increase when database is being multithreaded. He evaluated its performance, implementation complexity, and other measures and provides a guideline on how to make use of various threading method. From the experiment results, multi-threading improves database performance by 30% to 70% over single threaded implementation (Jingren Zhou, 2005) (Jingren Zhou, 2005). In this research, it is also found that Naïve parallelism is the easiest to implement but only gives a modest performance improvement.

From all previous research, we can see great potential in multi-threading database systems. It is proven that by parallelizing the system, it would have a moderate to significant gain in performance at a lower cost. Therefore with the right threading method and insertion method, we are able to improve database performance. This proves the potential in multi-threading database systems.

From all previous research, we can see great potential in multi-threading database systems. It is proven that by parallelizing the system, it would have a moderate to significant gain in performance at a lower cost. Therefore with the right threading method and insertion method, we are able to improve database performance. This proves the potential in multi-threading database systems.

2.5 DESCRIPTION OF BULK INSERTION

2.5.1 WHAT IS BULK INSERTION?

Bulk insertion is an efficient way to insert large amount of data into the database. Bulk insertion is being used to ensure high speed data transfer from the memory into the database. For example, referring to the example above about a stock market which has to store thousands of rows of data into the database and let’s assume that they are using SQL Server. You will have to insert all those data in the shortest time possible because it’s currently generating more than you can store. In this case use bulk insertion.

To ensure high speed data insertion, all data transformation process is being done before it enters into the insertion layer (MSDN Library (Bulk Insert), 2010). It’s being done in the business logic layer. Bulk insertion is transporting data stored in a flat file onto a database and data are being separated by either RowDelimiter or ColumnDelimiter. Bulk insert can only transfer data from a flat file into a database (MSDN Library (Bulk Insert), 2010).

```
BULK INSERT
   [ database_name. [ schema_name ] . | schema_name. ] [ table_name | view_name ]
   FROM 'data_file'
  [ WITH
 (
[ [ , ] BATCHSIZE =batch_size ]
[ [ , ] CHECK_CONSTRAINTS ]
[ [ , ] CODEPAGE = { 'ACP' | 'OEM' | 'RAW' | 'code_page' } ]
[ [ , ] DATAFILETYPE =
   { 'char' | 'native'| 'widechar' | 'widenative' } ]
[ [ , ] FIELDTERMINATOR = 'field_terminator' ]
[ [ , ] FIRSTROW = first_row ]
[ [ , ] FIRE_TRIGGERS ]
[ [ , ] FORMATFILE ='format_file_path' ]
[ [ , ] KEEPIDENTITY ]
[ [ , ] KEEPNULLS ]
[ [ , ] KILOBYTES_PER_BATCH =kilobytes_per_batch ]
[ [ , ] LASTROW =last_row ]
[ [ , ] MAXERRORS =max_errors ]
[ [ , ] ORDER ( { column [ ASC | DESC ] } [ ,...n ] ) ]
[ [ , ] ROWS_PER_BATCH =rows_per_batch ]
[ [ , ] ROWTERMINATOR ='row_terminator' ]
[ [ , ] TABLOCK ]
[ [ , ] ERRORFILE ='file_name' ]
 )]
```

Figure 2-8: Bulk insert in Trasact-SQL format. (MSND Library (Bulk Insert), 2010)

```
SET ANSI_WARNINGS OFF
DECLARE @str_command nvarchar(150)
SET @str_command = 'BULK INSERT [Customer_Sample] FROM ''' +
@SourceFilePath
+ ''' WITH (formatfile = ''' + @FormatFilePath + ''', firstrow =' +
cast(@RowNumber as nvarchar) + ')'
EXEC SP_EXECUTESQL @str_command
```

Figure 2-9: Bulk insertion from a flat file. (Harinath, 2006)

Referring to figure above, you will have to specify the batch size of the data you want to insert type of file or file format and file path. This is configuring the bulk insert task. The codes above is being used to bulk insert a flat file into a database.

Figure below shows another example of how to do bulk insertion in Transact-SQL from a flat file.

```
insert into stu_table values
                        (1, 'Komal',10),
                        (2, 'Ajay',10),
                        (3, 'Santosh',10),
                        (4, 'Rakesh',10),
```

Figure 2-10: Bulk insertion in standard SQL format.

If you cannot imagine how bulk insertion works, Source Code 2 shows a simple example on how bulk insertion is being done in a standard SQL format. You can imagine it as a line of SQL code that insert multiple rows into the table at one time.

Bulk insertion only supports OLE (Object Linking and Embedding) DB (Database) connection to database engine. To manage the connection to the database, a connection manager is being used to manage the connections and locate the database.

Bulk insertion is very important because it allows a reasonable amount of data to be inserted into a database on one connection. Let take for example, one bulk can hold 5000 rows and we have 4 connections to the database on 4 different threads. Next we assume that the insertion takes each bulk insert takes 4 seconds excluding thread overhead. Then this would allow us to insert 20,000 rows every 4 seconds. Each connection would sit on one thread and more connections could be created with it being queued, waiting for an available thread (using the thread-pool concept). Let’s now compare the above example to if we insert row-by-row and multithreading the row-by-row function. Let’s say each row insert requires 0.5 seconds excluding thread overhead and we have 4 threads. That is 32 rows for every four seconds on 4 threads.

The figure below would illustrate how bulk insert with multithreading works.

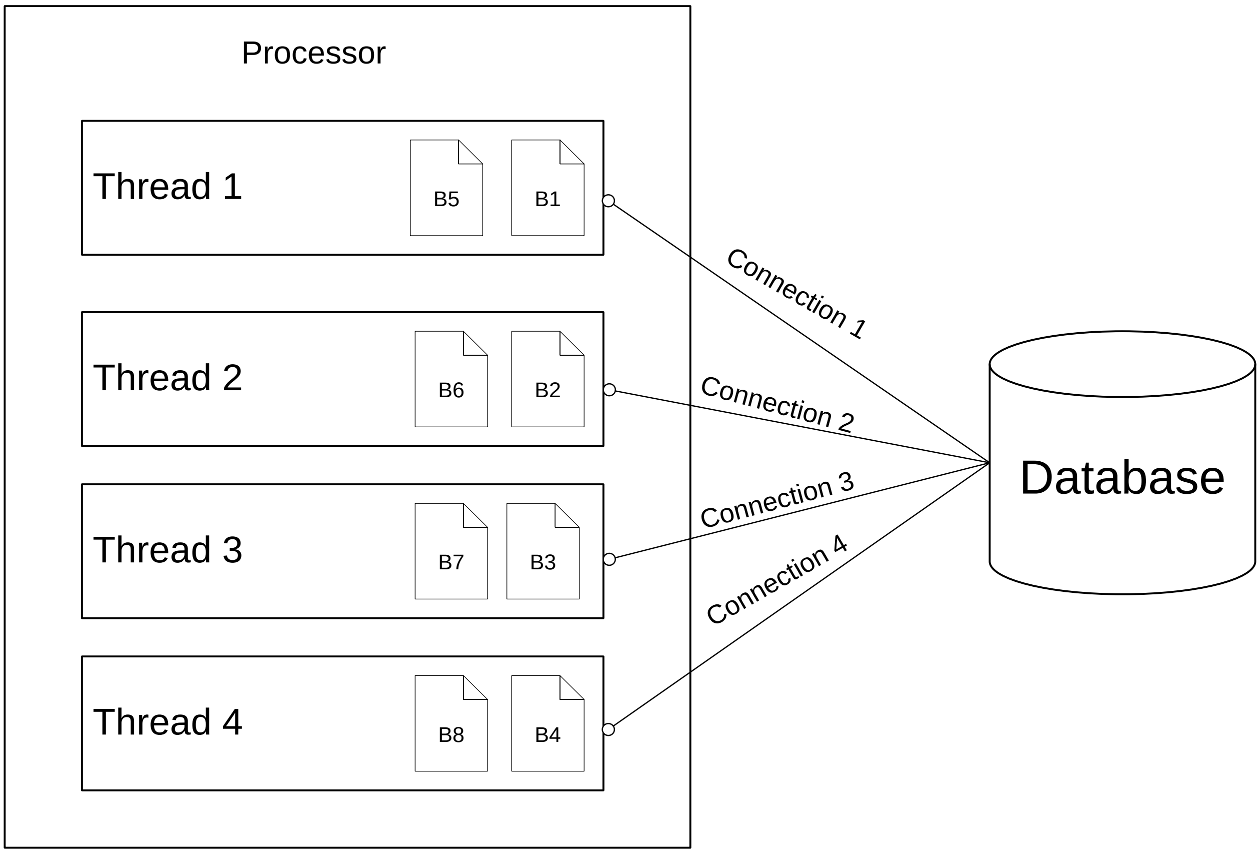


Figure 2-11: Illustration of bulk insert with multithreading

From the diagram above, we see that each B1 to B4 represents data in bulks. Referring back to the example above, each bulk is 5000 rows. B5 to B5 are bulk data which are in queue. Database connection may or may not maintain throughout the process. This is also one of the questions we are trying to answer in this research. This goes to show why bulk insert is important in our research to look for a better way to insert a huge amount of data into a database.

2.5.2 WHAT IS BULK COPY?

Bulk Copy is almost similar to bulk insertion; the difference here is that bulk copy has an imaginary table that sits in the memory before its bulk injected into the database. Bulk copy could be multithreaded and has been done before, so this proves it feasible in our project. Bulk copy is very fast, when run in a non-log mode, it's able to import/export thousands of rows per second. Bulk copy is also powerful; if you run a bcp.exe (bulk copy program) then you will be able to see that there are dozens of switches that can be used to specify how data is being threated during the import/export process (Harper, 2002). These switches can help control how and where the data is imported and exported from. BCP is users specify format data format for the imaginary table and also would be used when injecting the data into the database. You will have to understand the structure of the table before being able to perform bulk copy. The source code below illustrates how a BCP looks like.

```
bcp {[[database_name.][schema].]{table_name | view_name} | "query"}
{in | out | queryout | format} data_file
[-mmax_errors] [-fformat_file] [-x] [-eerr_file]
[-Ffirst_row] [-Llast_row] [-bbatch_size]
[-ddatabase_name] [-n] [-c] [-N] [-w] [-V (70 | 80 | 90 )]
[-q] [-C { ACP | OEM | RAW | code_page } ] [-tfield_term]
[-rrow_term] [-iinput_file] [-ooutput_file] [-apacket_size]
[-S [server_name[\instance_name]]] [-Ulogin_id] [-Ppassword]
[-T] [-v] [-R] [-k] [-E] [-h"hint [,...n]"]
```

Figure 2-12: Example of how BCP (Bulk Copy Program) looks like.

To run bulk copy, SQL Server has is it preinstalled and there is an API for it. The developer has to then specify the file format, batch size, and also data file. SQL Server and bulk copy can only work in two formats which are native format (SQL Server proprietary binary file format) or storing the data in a flat file in ASCII character set. Batch size is the amount of rows to be process at one batch it and be manipulated. Data files are the source and destination of the data. (Harper, 2002) BCP is preinstalled in SQL Server and is located at this directory “C:\Program Files\Microsoft SQL Server\80\Tools\Binn” and the file name is “Bcp.exe”. From here you’ll be able to specify the format of the data and all the other utilizes.

To successfully perform bulk copy operation, the system has to have enough credentials and they are as following. When importing rows from an exported file, you have to INSERT permission on the destination table (Harper, 2002).

From source, bulk copy is able to inject 300 million rows of data in just 10 minutes using 8 threads on a 64-bit machine fitted with 8 gigabytes of RAM (Random Access Memory). According to source, the CPU (Central Processing Unit) was running 100% usage. (Ltubia, 2008)

Bulk copy is able to perform multiple bulk copy operations using a single instance of a SqlBulkCopy class. The source code below illustrates how bulk copy works. The whole picture of bulk copy is to copy from a flat file to a database in a very short period of time with huge amount of data.

```
// Perform an initial count on the destination
//  table with matching columns.
SqlCommand countRowHeader = new SqlCommand("SELECT COUNT(*) FROM
dbo.BulkCopyDemoOrderHeader;", connection);
long countStartHeader = System.Convert.ToInt32(countRowHeader.ExecuteScalar());
Console.WriteLine("Starting row count for Header table = {0}",countStartHeader);

// Create the SqlBulkCopy object.
using (SqlBulkCopy bulkCopy = new SqlBulkCopy(connectionString))
{
   bulkCopy.DestinationTableName = "dbo.BulkCopyDemoOrderHeader";

   // Guarantee that columns are mapped correctly by
   // defining the column mappings for the order.
   bulkCopy.ColumnMappings.Add("SalesOrderID", "SalesOrderID");
   bulkCopy.ColumnMappings.Add("OrderDate", "OrderDate");
   bulkCopy.ColumnMappings.Add("AccountNumber", "AccountNumber");

   // Write readerHeader to the destination.
   bulkCopy.WriteToServer(readerHeader);
   readerHeader.Close();

   // Set up the order details destination.
   bulkCopy.DestinationTableName ="dbo.BulkCopyDemoOrderDetail";

   // Clear the ColumnMappingCollection.
   bulkCopy.ColumnMappings.Clear();

   // Add order detail column mappings.
```

```
    bulkCopy.ColumnMappings.Add("SalesOrderID", "SalesOrderID");
    bulkCopy.ColumnMappings.Add("SalesOrderDetailID", "SalesOrderDetailID");
    bulkCopy.ColumnMappings.Add("OrderQty", "OrderQty");
    bulkCopy.ColumnMappings.Add("ProductID", "ProductID");
    bulkCopy.ColumnMappings.Add("UnitPrice", "UnitPrice");

    bulkCopy.WriteToServer(readerDetail);
    readerDetail.Close();
}

//perform the same count again in the destination table to see if the bulk copy is
Perform accordingly.
```

Figure 2-13: Single-threaded Bulk copy operation from one database table to another.

```
//We open the connections
for (int m = 0; m < iNumThreads; m++)
{
      sqlSource[m] = new SqlConnection(sSourceConnectionString);
      sqlSource[m].Open();
      byConnectionsOpened++;
 }
 sqlDest.Open();

//perform copy process
oTc = new clsThreadCopy[iNumThreads];
for (int i = 0; i < iNumThreads; i++)
{
                //We initialize and configure with different source
connection,ranges,size..etc
                //as many clsThreadCopy´s instances as number of threads will execute in
parallel.
                 oTc[i] = new clsThreadCopy();
                 oTc[i].IdThread = i;
                 oTc[i].SqlSource = sqlSource[i];
                 oTc[i].SRangeLowKeys = sRangeLowKeys[i];
                 oTc[i].SRangeHighKeys = sRangeHighKeys[i];
                 oTc[i].SValuesFilter = sValuesFilter;
                 oTc[i].Size = iSize[i];
                 oTc[i].SDest = sDestinationConnectionString;
                 oTc[i].SSourceTable = sSourceTable;
                 oTc[i].SSourceTableScheme = sSourceTableScheme;
                 oTc[i].SDestTable = sDestTable;
                 oTc[i].SIndexField = sIndexField;
                 oTc[i].IBcOptions = iBcOptions;
                 oTc[i].INotifyAfter = iNotifyAfter;
                 oTc[i].IIndexFieldDataType = iIndexFieldDataType;
                 //Each time the clsThreadCopy fires the SqlRowsCopiedThreadEvent
event it will call to the OnSqlRowsCopied function.
                 oTc[i].SqlRowsCopiedThreadEvent += new
clsThreadCopy.SqlRowsCopiedEventDelegate(OnSqlRowsCopied);

                 WaitCallback w = new WaitCallback(CopyData);
                 //For each instance of clsThreadCopy we create one thread which call
CopyData function.
                 ThreadPool.QueueUserWorkItem(w, oTc[i]);
}

//We wait until all threads end.
lock (oThreadLocker)
{
        while (iNumThreads > 0)
        {
             Monitor.Wait(oThreadLocker);
         }
}
```

```
//We close connections.
for (int i = 0; i < byConnectionsOpened; i++)
       if (sqlSource[i].State == System.Data.ConnectionState.Open)
                    sqlSource[i].Close();
        if (sqlDest.State == System.Data.ConnectionState.Open)
                   sqlDest.Close();
```

Figure 2-14: Multi-threaded Bulk Copy operation from one database table to another.

```
private void WriteToDatabase()
{
    // get your connection string
    string connString = "";
    // connect to SQL
    using (SqlConnection connection = new SqlConnection(connString))
    {
        // make sure to enable triggers
        // more on triggers in next post
        SqlBulkCopy bulkCopy = new SqlBulkCopy
            (
            connection,
            SqlBulkCopyOptions.TableLock |
            SqlBulkCopyOptions.FireTriggers |
            SqlBulkCopyOptions.UseInternalTransaction,
            null
            );

        // set the destination table name
        bulkCopy.DestinationTableName = this.tableName;
        connection.Open();


        // write the data in the "dataTable"
        bulkCopy.WriteToServer(dataTable);
        connection.Close();
    }
    // reset
    this.dataTable.Clear();
    this.recordCount = 0;
}
```

Figure 2-15: Bulk Copy of flat-file to database

2.5.3 WHAT IS CONNECTION POOLING?

Connection pooling is a technique used to create and manage a pool of connections that are ready to be executed on a thread at any time. What it does is, there will be a pool of connections being created at the start of the thread. The amount of thread being created is user define. When the application needs a connection, the connection pool will provide a connection then when the transaction is done; the connection would be return to the pool. This is how connection pooling works. It has similar concept to thread pooling. (MySQL Connection Pooling, 2010)

When the connection is being called by the thread, it is exclusively for the calling thread only until its being returned to the pool. The benefits of connection pooling are: it’s able to reduce connection creation time. This is possible because the connections are being created once only and its then being recycled. Thus this is able to reduce the database driver overhead that is involved every time a connection is being created. By not closing the connection, we are able to reduce the thread overhead. Next would be simplified programming model. When using connection pooling, each thread can act as if it has created its own database connection and allow it to be used instantaneously. Finally it allow user to control resource usage. If connection pooling is not being used, resources will go to waste by creating and closing connections to a database. This also will lead to unpredictable behavior under extreme load. Each connection created to a database involves overhead which consist of memory, CPU, context switches and many more. Connection pooling can help to improve system performance by keeping resources utilization below the point where the application starts to slow down. (MySQL Connection Pooling, 2010)

Source Code below illustrates how to create a connection pooling method in C#.

```
SqlConnection conn = new SqlConnection();
conn.ConnectionString = "Integrated Security=SSPI;Initial; Catalog=school;
server=localhost; Min Pool Size = 10; Max Pool Size = 100";
conn.Open();
```

Figure 2-16: Creating Connection Pooling

The default number of connections created is 100. Users are able to enlist the pool by using Enlist function where it will automatically enlist the connection in the current transaction context of the creation thread if transaction context exists. To clear the pool, used .ClearAllPools function and it will close all connections. To connect to multiple databases, multiple pools are then created, one pool for one database.

2.5.3 WHAT IS DATA ACCESS LAYER (DAL)?

Data access layer is a back-end-code that links between the application and the database engine. DAL does not have any business logic nor does it have any function to manipulate the data that goes thru it. The task of a DAL is to manage the connection's to databases, executes queries and select the database to insert into if there is multiple database engine available. A DAL is capable of handling multiple database and we would like to be smart enough to know how many connections to be made to the database in order to get the optimal performance.

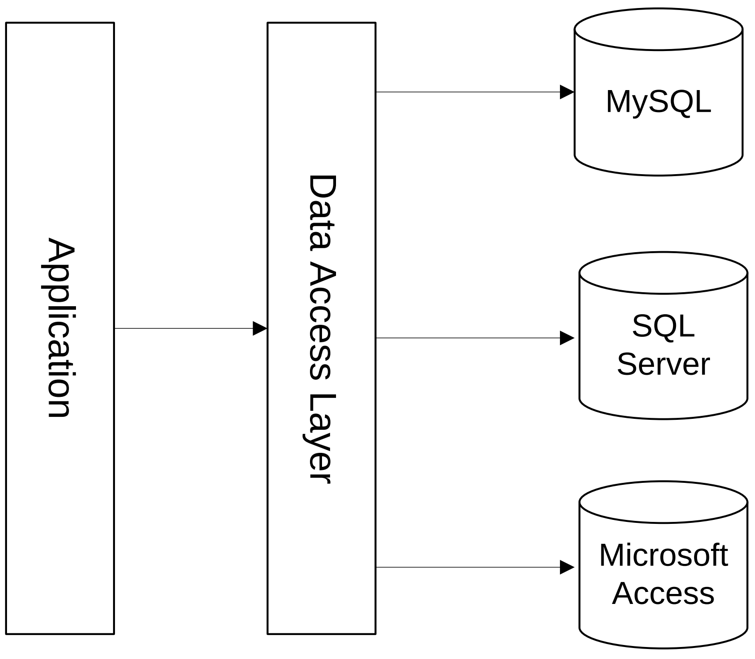


Figure 2-17: Data Access Layer (DAL)

There are three main components in a DAL, they are as following:

- **Data access logic components.** It’s a data access component abstract with all the logic necessary to access underlying data storage. Centralizing the data access functionality would make the DAL easier to manage and maintain (J.D. Meier, MSDN Library (DAL), 2009).
- **Data helper/utilities.** They are help functions that assist developers in data manipulation, data transformation and data access within the layer. This are specialized library or custom routines that are specially designed to maximize data access speed and reduce development of the logic and service agent (J.D. Meier, MSDN Library (DAL), 2009).
- **Service agents.** These are services used when business components must use functionality exposed by an external service. They are used to format the data to your application requirements (J.D. Meier, MSDN Library (DAL), 2009).

To improve the performance of the DAL, batching is being used to improve its performance. Batching is done by using bulk insertion method. It would reduce the overhead by increasing data throughput and reduce latency.

DAL is just a package that house different threading methods, connection management services and ways of how data is inserted, updated or deleted. Creating and closing connection can cause a lot of threading overhead, so a DAL has to be able to manage it efficiently. Data format are very important to ensure that the data is being stored properly. Your data has to be standardizing to ensure accuracy. DAL’s are scalable and also flexible. They are able to communicate with different database engine and performing multiple tasks but in this project we would be focusing on the INSERT statement only.

Performance consideration that we have to take is functionality of both DAL and database design. The DAL has to perform to its maximum data throughput. This means tuning the system and also the database engine. To manage the connections to the database, connection pooling could be used. Results from simulated load scenarios could be used to determine the optimal number of connections to different database engine. Batch command is important into improve the DAL performance as it reduce the number of trip it has to do to the database.

```
private void InsertData()
{
   IDBManager dbManager = new DBManager(DataProvider.SqlServer);
   dbManager.ConnectionString =ConfigurationSettings.AppSettings[
   "ConnectionString "].ToString();
   Try
   {
      dbManager.Open();
      dbManager.CreateParameters(2);
      dbManager.AddParameters(0, "@id",17);
      dbManager.AddParameters(1,"@name", "Joydip Kanjilal");
      dbManager.ExecuteNonQuery(CommandType.StoredProcedure,
      "Customer_Insert");
   }
   catch (Exception ce)
   {
     //Usual code
   }
   finally
   {
      dbManager.Dispose();
   }
}
```

Figure 2-18: Insert function using stored procedure in a DAL (Kanjilal, 2007).

```
IDbCommand InsertCommand =
            DataObjectFactory.CreateStoredProcedureCommand(_DatabaseConnection)
;
InsertCommand.CommandText = "up_Orders_Insert";
InsertCommand.UpdatedRowSource
            = UpdateRowSource.FirstReturnedRecord;
The INSERT stored procedure
INSERT INTO [Orders]
```

```
(
        [CustomerID]
)
VALUES
(
        @CustomerID
)
```

Figure 2-19: Insert Command in a DAL written in C# (Donald, 2010).

Connection plays a huge role in betting the best performance out of the DAL. All database connections should be managed by the DAL. This involves creating, closing and managing the connections using available resources. Retry logics are designed to manage situations where connection fails (J.D. Meier, MSDN Library (DAL), 2009).

2.6 TECHNIQUES USED IN DAL

2.6.1 TABLE LOCKING METHOD

To speed up insertion operation to a database with multiple statements for into a non-transactional database table, lock your table.

```
LOCK TABLE stud_tbl WRITE;
INSERT INTO stud_tbl VALUES (1,23),(2,34),(4,33);
INSERT INTO stud_tbl VALUES(8,26),(6,29);
….
UNLOCK TABLES;
```

Figure 2-20: Lock table for non-transactional tables

By locking your table, the extra performance comes from the buffer is flushed from the disk only once after all the INSERT statements have been completed. Normally the index buffer would be flush after every INSERT statement. Explicit locking is not needed if you insert all in one single INSERT statement. For transactional tables, START_TRANSACTION and COMMIT is being used instead of LOCK_TABLES to obtain optimal performance.

Locking also lowers the time for multiple-connection tests, although the idle time for individual connection may increase due to lock waits. Let take for example we have the following insert statement running concurrently:

Connection 1 does 1000 inserts

Connection 2, 3, and 4 does 1 insert

Connection 5 does 1000 inserts

If locking is not being used, connection 2,3, and 4 would finish before 1 and 5. But if locking is being used, connection 2m3 and 4 would most likely be done before 1 or 5 but the total time would is 40% faster compared to without locking. Locking would permits threads to access the table. This applies to MySql server database. (MySql Speed of INSERT Statement, 2003)

2.6.2 SEQUENTIAL PROCESSING

Sequential processing is a traditional and most common approach to input data into a database. This means that data is inserted row-by-row. Each command has one row of data inserted. In this method, you will create one connection and insert multiple rows of data into the database and each query hold s one row. If you have 1000 rows to insert, then you have to execute the query 1000 times. The next row cannot be executed until the current row has been inserted.

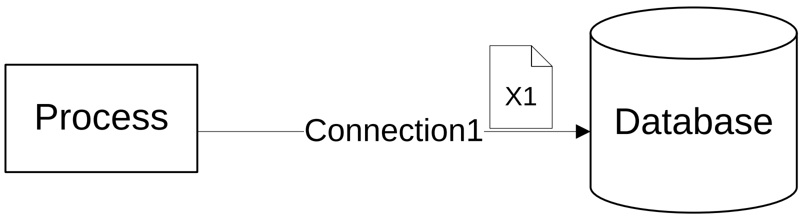


Figure 2-21: Sequential data insertion to a database

This approach has many limitations when it comes to huge amount insertion. The major limitations are it tends to cause bottleneck when there is other application sharing the same database. Furthermore if the database is being shared across a number of applications, it will cause substantial degrading of performance to the other application that is trying to access the database (Verenkar, 2010). The other limitation is that data is being processed sequentially. Even though input data is capable of being processed independently, stored procedure inherently process the data sequentially (Verenkar, 2010). It is very difficult to perform parallelism in stored procedure. The final limitation is that it has the high tendency to unevenly distribute load between system components and this is due to bad architecture (Verenkar, 2010). This procedure would be used as a benchmark for all the other methods to be used in the DAL.

2.6.3 PARALLELIZING DATABASE ACCESS

Most new database engines can support multiple calls simultaneously and support concurrent access (Verenkar, 2010). This has allow database developers to parallelize the data input process by increasing the number of database connections and also multithread the INSERT statement. The following diagram will illustrate how it works.

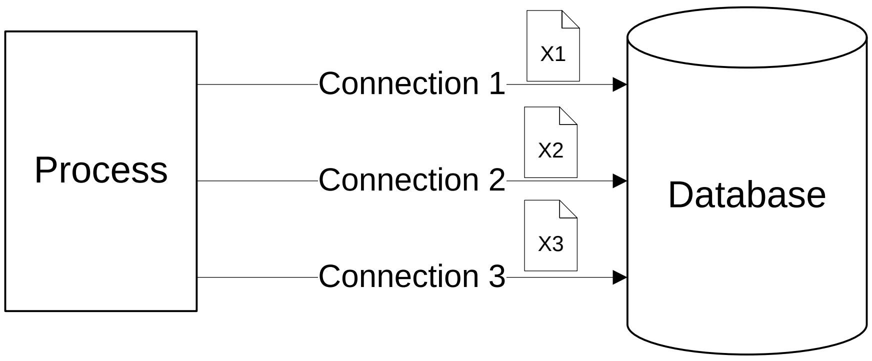


Figure 2-22: Parallel Database Access with 3 Connections and INSERT Statement

From the diagram above, there are three connections being made to the database and X1, X2 and X3 represents the INSERT statements. From the illustration above, if we assume that each insert statement has one row in hand, then it can insert 3 rows at a time. This would then speed up the process by 3 times. This process could be made even faster if we use bulk insertion on X1, X2 and X3. This process also has its own limitations, it also face the limitation of chances having performance bottleneck at the SQL Database. If there are too many connections being made to the database with the INSERT function taking too much usage from the processor this will definitely cause a bottleneck. Subsequently cause performance degradation. The optimal number of connections, threads and bulk size is very important to maintain an optimal performance.

2.6.4 USING DATA PARALLELISM IN .Net Framework 4

This method will solve all 3 limitations of the parallelizing database access and sequential database access. In this approach we used stored procedure to perform bulk insertion. It will then accept XML input and insert all the rows into the database. This way all the data in the XML can be processes in parallel and from theory, it would improve performance. In this method the .Net Framework 4 parallel programing construct would be used to perform this task. The entire question about how many threads to create, how many connections to create and application scalability would be solved by the .Net Framework 4. All the processing would be done in parallel without having to deal the intricacies of threading. (Verenkar, 2010)

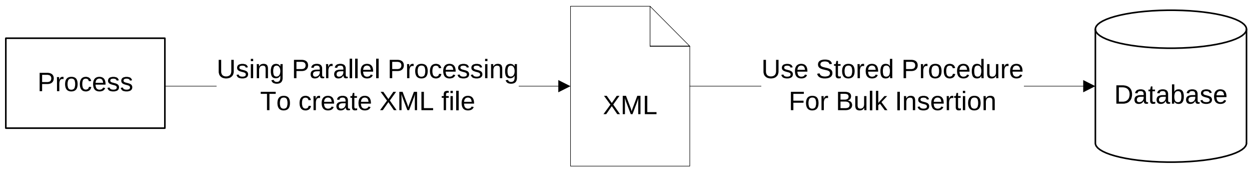


Figure 2-23: Creating an XML file using parallel programming and using stored procedure to perform bulk insertion.

In this process we will have to store all the data into a XML format file then from their use store procedure to do bulk insertion into the database. The stored procedure, we would be using Transact-SQL. The limitation here is that Transact-SQL only works with SQL Server or .Net Framework.

2.6.5 PARALLEL LINQ

PLinq performs INSERT, UPDATA, and DELETE operation in LINQ to SQL by updating the database content. By default, LINQ to SQL translates all queries and non-queries action to SQL and executes the changes. LINQ to SQL if flexible in manipulating and persisting changes that is made to the object you created. As soon as an entity object is available, your then allowed to change them as typical object in your application.

```
Northwnd db = new Northwnd(@"c:\Northwnd.mdf");

// Query for a specific customer.
var cust = (from c in db.Customers where c.CustomerID == "ALFKI" select c).First();

// Change the name of the contact.
cust.ContactName = "New Contact";

// Create and add a new Order to the Orders collection.
Order ord = new Order { OrderDate = DateTime.Now };
cust.Orders.Add(ord);

// Ask the DataContext to save all the changes.
db.SubmitChanges();
```

Figure 2-24: Example of how PLinq is being implemented.

2.6.6 CONCURRENT BAG

The new .Net Framework 4 come with many new namespace and one of the many new namespaces are located in System.Collections.Concurrent. This new namespace contains a handful of types which helps in implementing different types of thread safe collections. This new namespaces focus on multithreading which is the hot topic in today's application development. The name for this new namespace is called ConcurrentBag<T>. It's a typical bag data structure (multiset) which keeps data in an unordered format with duplicates allowed. (Etheredge, 2010)

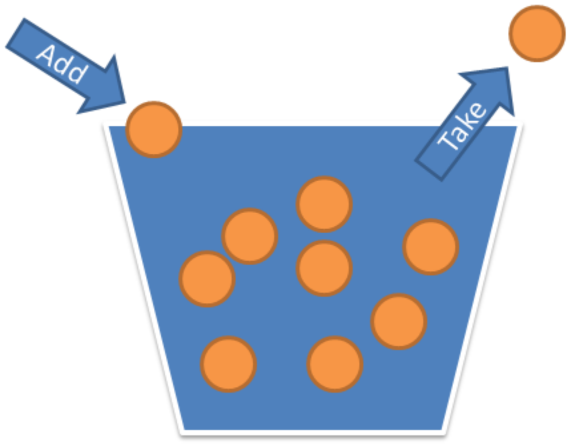


Figure 2-25: Illustrate the concept of concurrent bag in .Net 4 (Etheredge, 2010).

Following are source code's to illustrate how it works.

```
var cb = new ConcurrentBag<string>();
cb.Add("test");
```

Figure 2-26: Illustrate how to add item into a concurrent bag.

```
string val;
if(cb.TryTake(out val))
{
    Console.Writeline(val);
}
```

Figure 2-27: Illustrate how to remove item from a concurrent bag.

```
var cb = new ConcurrentBag<string>();
Task.Factory.StartNew(() =>
{
    for(int i = 0; i < 1000; i++)
    {
        cb.Add("test");
    }
    cb.Add("Last");
});

Task.Factory.StartNew(() => {
    foreach(string item in cb)
    {
          Console.WriteLine(item);
    }
}
```

Figure 2-28: Code illustrates concurrent programming in concurrent bag.

In Source Code 15, it illustrates how Concurrent Bag can support parallel programming. The first block of the code is adding items into the bag and the second block is being fired up to read what is in the bag. Data that is added after the enumeration started won't be read when the second block is fired. It will only have till what it was last added before the second block of code is fired.

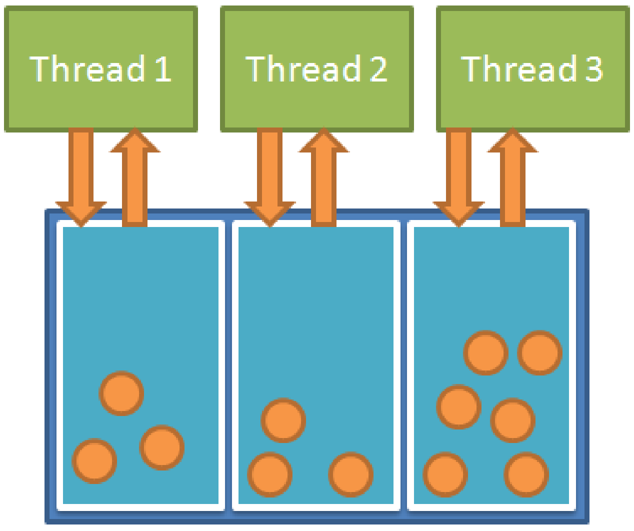


Figure 2-29: Illustrates the concept of Concurrent Bag when threaded (Etheredge, 2010).

This method is usually being used to combine data that is being read from multiple threads before its being processed into another file or injected into a database. This is a good way to hold temporary data for it support multithreading and also allows data specific objects to be stored. The best thing about this method is that its thread safe. This has a similar concept to how array list works but this support multithreading. Concurrent bag allows implementation of IEumerable<T>, this allows developers to iterate over it in the same way as other class that supports IEumerable. LINQ queries could be executed against it. Developers are able to enumerate it while some other threads are adding item into it at the same time.

2.7 COMPARISON BETWEEN AVAILABLE WAYS TO DO BULK INSERTION

	Bulk Copy	**Table Locking**	**Transact-SQL**	**Parallelizing**	**Parallel in .Net 4 (PLinq)**
Chances of bottleneck	Yes		No	Yes	No
Supports C#	Yes	Yes	Yes	Yes	Yes
Application Dependent			Yes	Depends	Depends
Supports Multithreading	Yes (not stable)	No	No	Yes	Yes
Support MySQL 5.0		Yes	No	Yes	Yes
Support SQL Server 2008	Yes		Yes	Yes	Yes
Microsoft Access 2010				Yes	
Complexity		High	Moderate	High	Moderate
Scalability	High	High	High	High	High
Speed of transaction	High	High	High	High	High
Multi-Connection	Application Define		No	Yes	Yes
Threading Method	Explicit		Explicit	Implicit & Explicit	Implicit
Portability	Yes	No	No	Yes	Yes
Flexibility	No	No	Moderate	Yes	Moderate

Table 2-2: Comparisons between DAL methods.

From the comparison in Table 2-2, it is known that parallelizing would improve performance. It is being supported on most of the platform and is highly flexible as well as scalable. In parallelizing, the developer is almost in control of how he/she wants the program to flow and would have more room to work with compared to the other methods. The next best method would be parallel in .Net 4. This method maybe highly scalable but there is flexibility to a certain extend where most of the things, it's not in control of the developer. There are pro's and con's in all of the method but parallelizing would most likely do the job best of inserting a huge amount of data into the database in the shortest time possible. Parallelizing has more variables to change and tweak to get better performance and being in control of threads and connections which are the most important thing in multithreading with database.

2.8 DISCUSSION

2.8.1 DIFFERENCES BETWEEN SINGLE-THREADED DAL AND MULTI-THREADED DAL

	Single-threaded	**Multi-threaded**
Insertion method	Serial	Parallel
Number of database Connection	One	Multiple
Number of rows per command	One	In Bulk
Processor Utilization (multi-core processor)	Low	High
Insertion Speed (huge amount of data)	Slow	Fast
Number of thread used	One	Multiple
Complexity	Less Complex	Complex
Scalability	Not Scalable	Highly Scalable
Performance (theory)	Slow	Fast

Table 2-3: Comparison between single and multi-threaded.

2.8.2 DATABASE COMPARIOSN

	Access 2010	**SqlServer 2008**	**MySql 5.0**
Speed			
Maximum Connections	255	32,767	4000 (user define)
Max Worker Thread (64-bit processor, 4 Cores)		512	8
Multithreading		Yes	Yes
Bulk Insert	Yes	Yes	Yes
Transact-SQL		Yes	
.Net Framework 4	Yes	Yes	Yes
Stored Procedure		Yes	Yes

Table 2-4: Comparison between database engines.

2.9 CONCLUSION

From all the findings, the conclusion is that most database engines today supports multithreading and some even have built in multithreading methods to reduce the complexity of developers' codes. Research finding has also shown that multithreading on database will produce a performance improvement when we compare it to a single threaded application on a database. However none of them have has shown which threading method is better for which database, number of optimal threads and connections to open, how to perform bulk insertion with multithreading, create a DAL that is able to recognize what database and decide the optimal number of threads and connections to be created. They are also usually application or domain specific, which makes the application or DAL not portable to change environment. They have not done much research on multithreading done on top of bulk insertion or on top of what the database. Bulk insertion itself is already very efficient method but there are still ways to be able to improve the performance of bulk insertion. Threading method is also very important, none of them specify which threading method suits which database and also what other performance tuning is being done on the database or the application itself to improve the performance. Hereby there is still much room to improve on multithreading a database engine. This leads to finding a more efficient and faster way into doing bulk insertion.

CHAPTER 3 METHODOLOGY

3.1 INTRODUCTION

In this chapter, an explanation on how the experiments were being carried out as well as how the DAL is being developed. This project is being classified as a Threading Methodology in Database. As mentioned in Chapter 1, the purpose of this project is to find a more efficient way into doing bulk insertion. In this project, it would only be focusing on the INSERT statement of a DAL. Existing DAL are usually single threaded and few are industry proven to run on multi-threaded platform. They are application, database and machine dependent. From the experiment result, the DAL would be developed according to the number of row to insert and type of database. From there it would determine the insertion method as well as the number of threads to be used. The main purpose of this project is to prove that multi-threading does improve the performance of bulk insertion. Due to time constraint, the experiment was conducted on a particular machine, two different database engines were being tested as well, and three insertion method with various numbers of threads were used.

The test includes getting the time taken to insert a specific number of rows. Number of rows range from 1 to 50,000 rows. Machine utilization is also being monitored to see the relationship it has with the RAM, CPU and magnetic disk I/O.

A timer is being set in place to take the time to complete the whole processes which include reading, processing and insertion. The time is being clocked in milliseconds. It would be used to monitor the performance. The same process would be repeated for five cycles and an average it taken as the result.

3.2 Implementation of Concept

3.2.1 Threading Method

Throughout the experiments, threads are manually spawned in order to maintain a controlled environment. The codes below show how the program is being threaded where two threads are used.

```
//create and start threads
ThreadStart threadDelOne = new ThreadStart (insOne.RunInsertion);
ThreadStart threadDelTwo = new ThreadStart (insTwo.RunInsertion);

Thread threadOne = new Thread(threadDelOne);
Thread threadTwo = new Thread(threadDelTwo);
threadOne.Start();
threadTwo.Start();

//thread join
threadOne.Join();
threadTwo.Join();
```

Figure 3-1: Create and Start threads

3.2.2 Sequential Insertion

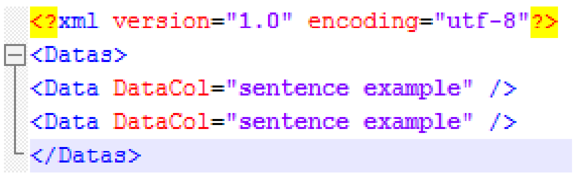
Sequential insertion is done by using the standard SQL insert command and each command would insert one row. Transaction is being used in this process where the whole block will be committed after the last data is inserted. Rollback is being used if there is an error [6]. The same code is being used for 1, 2, 4 and 8 threads. Before inserting the row, it is being formatted into compatible SQL command by replacing certain characters to work with SQL command formatting. For sequential, the test is done with and without transaction.

```
for (int i = 0; i < dataList.Count; i++)

{

        sqlStr = "INSERT INTO TestTbl(DataCol) VALUES(N" + dataList[i] + ")";

        sqlCmd = new SqlCommand(sqlStr, conn, transaction);
        sqlCmd.ExecuteNonQuery();

}

transaction.Commit();
```

Figure 3-2: Sequential Insertion

3.2.3 SQLBulkCopy Insertion

SQLBulkCopy is a .NET4 function to insert data in bulks into SQL Server 2008 [8]. It receives XML's and inserts them. The format for the XML file is as shown in Figure 3-3. This method is tested by using 1, 2, 4, and 8 threads; where the number of individual XML's are read according to the number of threads used respectively. For example, if 8 threads are used, they will read from 8 individual XML's.



```xml
<?xml version="1.0" encoding="utf-8"?>
<Datas>
<Data DataCol="sentence example" />
<Data DataCol="sentence example" />
</Datas>
```

Figure 3-3: An example of XML formatting.

The codes below shows how the SQLBulkCopy insertion is being done.

```
dataSet dataSet = new DataSet(); dataSet.ReadXml(xmlFileName);

sourceData = dataSet.Tables[0];

//perform insertion

using (SqlConnection conn = new SqlConnection(connStr))

{

                conn.Open();

                using (SqlBulkCopy bulkCopy = new   SqlBulkCopy(conn.ConnectionString))

    {

                bulkCopy.ColumnMappings.Add("DataCol", "DataCol");

                bulkCopy.DestinationTableName = "TestTbl";

                bulkCopy.WriteToServer(sourceData);

    }

        conn.Close();

}
```

Figure 3-4: SQLBulkCopy insertion

3.2.4 MySQL Bulk Loader Insertion

MySQL Bulk Loader from the MySQL .NET Connector 6.2.4 [7] is used for this experiment. It receives flat files and inserts them using the MySQL import loader. The number of files created would depend on the number of threads used. This method is being tested with 1, 2, 4, and 8 threads. The following codes illustrate how the import loader is performed.

```
//perform import loader
try
{
        MySqlBulkLoader myBulk = new MySqlBulkLoader(conn);
        myBulk.Timeout = 600;
        myBulk.TableName = "testDatabase.testTbl";
        myBulk.Local = true;
        myBulk.FileName = fileName;
        myBulk.FieldTerminator = "";
        myBulk.Load();
 }
```

Figure 3-5: perform import loader

3.2.5 System Utilization

In the next experiment, the RAM, CPU and hard disk drive utilization are captured throughout the insertion period. This is done during 70,000 to 80,000 rows on all the insertion methods, number of threads and database engines as shown in Table 1. A sample is captured every 30 seconds and the average from the samples would be taken as the result [9]. Codes below illustrates the system utilization is being captured.

```
PerformanceCounter cpuUsage = new PerformanceCounter("Processor", "% Processor Time", "_Total", true);
PerformanceCounter memoryAvailable = new PerformanceCounter("Memory", "Available MBytes");
PerformanceCounter physicalDiskTransfer = new PerformanceCounter("PhysicalDisk", "Disk Bytes/sec", "_Total", true);
startMemory = totalMemoryCapacity - memoryAvailable.NextValue();
```

3.3 TESTING

3.3.1 DATASET

From the research of others, only by inserting at least 3 million rows of data with a combination of ‘double’, ‘varchar’, and ‘integers’ would it make a difference between single and multi-threaded. But from the tests done, this is proven to be not true. A significant difference in performance increase could be seen with much lesser rows. 3 million rows would require a much more complicated data reader with buffer in place; due to time constraint I didn’t create such a data reader. I tested my DAL using a data range of 1 to 50,000 rows. Even from here there is significant difference. The dataset used in the tests consist of 302 characters that include characters, integers, symbols, and space.

Below is the number of rows the dataset was being used for testing:

1	2	5	10	15	30	50	100

500	1,000	5,000	10,000	50,000	60,000	70,000	80,000

3.3.2 DATABASE

In the tests, two database engines were used, which are MySQL 5.2 and also Sql Server 2008 Enterprise. The database is being formatted to have neither rules nor relationship. This is to reduce complications and making it a controlled environment. The database has one table and 2 columns. One column would serve as an index with integer data format and the other is to store the data and ‘MAX VARCHAR’ is being used. Index is automatically generated by the database engine.

3. 3.3 TEST MACHINE AND SOFTWARE SPECIFICATION

The specification of the machine used for testing is in Table 3-1.

Item	**Details**
Processor	Core 2 Quad 2.66Ghz Q9400
Random Access Memory	3.9 GB
Magnetic Hard Disk Drive	Seagate Barracuda 7200rpm 320GB
Development Software	Visual Studio 2010 .net4 framework
Programming Language	C#
Operating System	Windows XP Sp3(32-bit)
Ram Usage at Start	438 MB
CPU Usage at Start	0%
Database Engine	Microsoft SQL Server 2008 Enterprise Oracle MySQL 5.2

Table 3-1: System and Software specification

3. 3.4 Program Flow

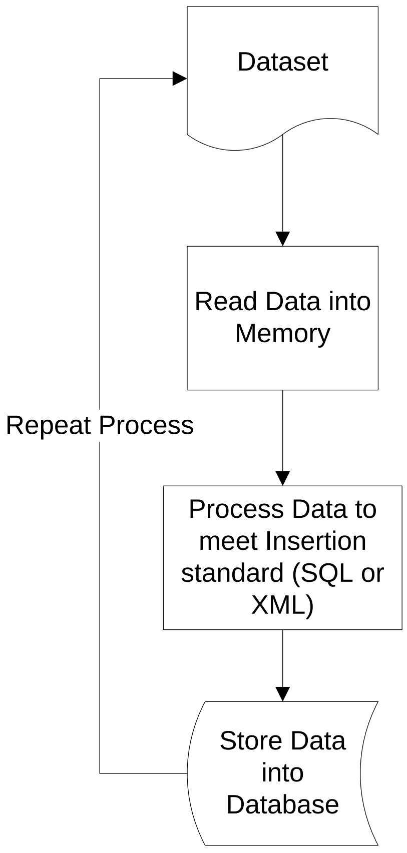


Figure 3-6: Flow of the Insertion process

Figure 3-6 shows the flow of the insertion process. Dataset is being stored in individual text files. Time taken includes reading data from the text file into the memory as well as process the data to meet insertion standard is to convert the data to meet SQL string insertion. By adding symbols, it will meet SQL statement standard. Figure 3-7 is the snippet of formatting the string for SQL parameters.

```
   public string FormatSqlParam(string strParam)
{
   string newParamFormat = null;

   if (strParam == string.Empty)
   {
      newParamFormat = "'" + "NA" + "'";
   }
   else
   {
      newParamFormat = strParam.Trim();
      newParamFormat = "'" + newParamFormat.Replace("'", "''") + "'";
   }

   return newParamFormat;
}
```

Figure 3-7: Code snippet of formatting SQL Parameters.

All the time taken from test is being stored in a text file to be process. A frequency of 5 times is done on each test and an average it taken from it. The same and all dataset are being used on all tasks. This would maintain consistency and also control the environment.

3. 3.5 TASK 0: BUILD DATASET

The dataset is build based on comments from eBay.com. Over one thousand comments were taken and duplicate it for all the dataset used. To make it constant at 302 characters, a program is build to copy and paste the front of the sentence to make up 302 and remove additional characters. Figure 3-8 shows the codes of the process.

```
while (reader.Peek() != -1)
{
    curStr = reader.ReadLine();
    curStr = curStr.Trim();
    curLength = curStr.Length;
    if (curLength < 300)
    {
        copyLength = 300 - curLength;
        copyStr = curStr.ToCharArray();
        x = 0;
        while(curStr.Length <= 300)
        {
            if (x < copyStr.Length)
            {
                curStr = curStr + copyStr[x];
                x++;
            }
            else {
                x = 0;
                curStr = curStr + copyStr[x];
            }

        }

        if (curLength > 300)
        {
            cutLength = 300 - curLength;
            copyStr = curStr.ToCharArray();
            for (int z = 0; z < 300; z++)
            {
                curStr = curStr + copyStr[x];
            }
        }
    }
    else if (curLength > 300)
    {
        cutLength = 300 - curLength;
        copyStr = curStr.ToCharArray();
        for (int y = 0; y < 300; y++)
        {
            curStr = curStr + copyStr[y];
        }
    }
    writer.WriteLine(curStr);
    Console.Write(".");

}
```

Figure 3-8: Snippet of dataset builder process.

3. 3.6 TASK 1: CONNECTIONS AND THREADS

In the initial plan, there were four types of test to be done before processing with different insertion methods. But due to some feasibility issues, the plan has been reduce to only two sections out of four. The two tests done were ‘single connection, single thread’ and ‘multiple connections, multiple threads’. The initial plan is shown in Figure 3-9.

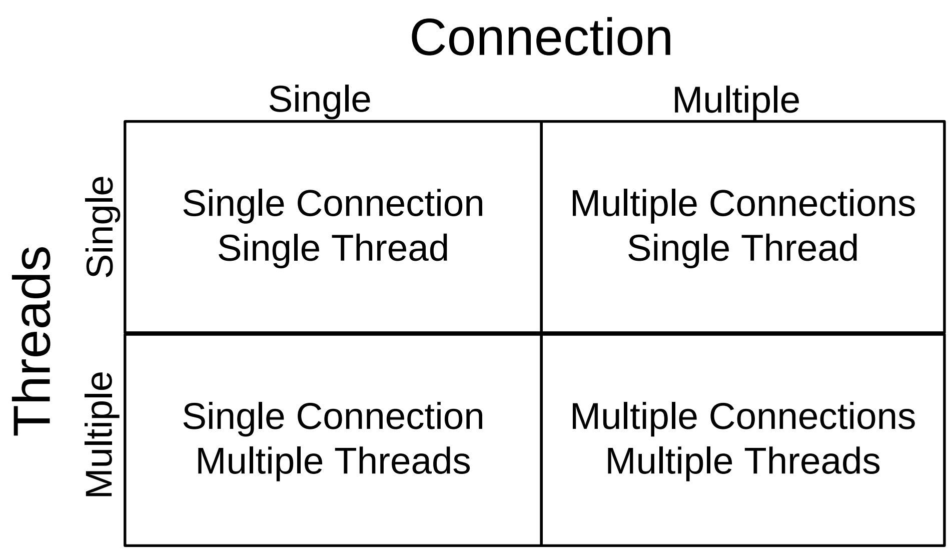

Connection
Single
Multiple
Threads
Single
Multiple
Single Connection Single Thread
Multiple Connections Single Thread
Single Connection Multiple Threads
Multiple Connections Multiple Threads

Figure 3-9: Thread’s and Connection’s testing combination.

From that the tests, multiple connection to a single thread and also single connection to multiple threads is not feasible in programming point of view. Therefore only single connection on single thread and multiple connections to multiple threads were being tested. The threading method used is manual threading. This allows me to control the number of threads to create and also have full control over the threading.

The first experiment done was on single connection and thread. The insertion method used here is sequential insertion method. The data would be inserted row-by-row using the standard SQL Insert statement. This task is to set a benchmark for the subsequent test to follow.

```
for (int i = 0; i < dataList.Count; i++)
{
        sqlStr = "INSERT INTO TestTbl(DataCol) VALUES(N" + dataList[i] + ")";
        sqlCmd = new SqlCommand(sqlStr, conn, transaction);
        sqlCmd.ExecuteNonQuery();
}
transaction.Commit();
```

Figure 3-10: Snippet on sequential SQL insertion.

The figure above shows how the data is being inserted into the database. Test is being done with the transaction feature and without it. This is to test if the transaction feature does make a difference and improve the performance. Codes without transaction feature share similar code as in Figure 3-10.

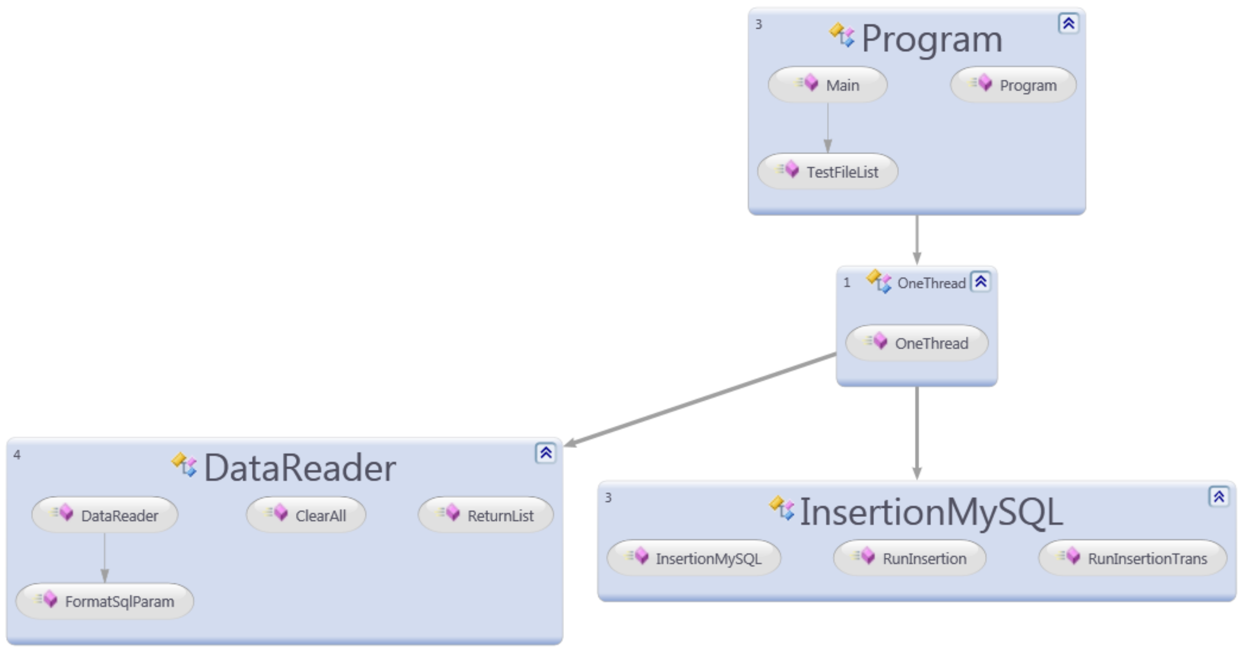


Figure 3-11: Chart shows the architecture of the program.

The method TestFileList is where all the names of the dataset files are being stored. The program in main would repeat for every test file. One thread and InsertionMySQL would recur 5 times to get the average time. This architecture is being used for both database engines.

3. 3.7 TASK 2: MULTI THREADED AND CONNECTION

The next experiment done was on multi-thread the processes. This is done on 2, 4 and 8 threads. Each thread will hold its own connection. The same insertion method is being used in task 1. The dataset is being split into the number of threads to be used. All the data is being stored in the memory to optimize the performance.

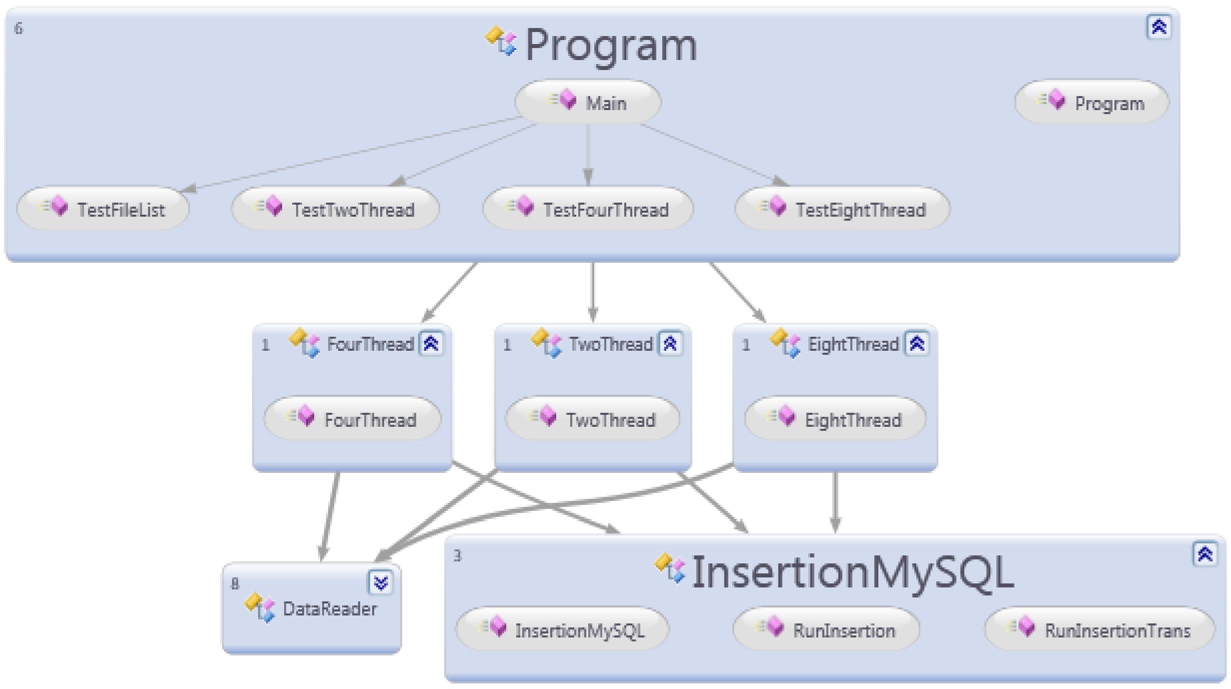


Figure 3-12: Program architecture for multi-threaded SQL sequential insertion

In the dataReader class, it will split the data into individual arrays according to the number of threads being used. In this process, the data reader would split the data into individual arrays according to the number of threads used. This test is being done on both database engines.

3. 3.8 TASK 3: THREADING WITH BULK COPY

In this test, a package from the .Net 4 Framework is being used. It’s a module that supports bulk insertion and it accepts data in XML and does the insertion. Its method name is called SqlBulkCopy (BCP). BCP provides significant performance jump compared to the conventional SQL INSERT method (Bulk Copy Operations in SQL Server (ADO.NET), 2011). The data has to be converted into XML files before insertion. This is done in the data reader process and it will be split into multiple files according to the number of threads used. Each thread will take one for to be inserted. This test would involve testing on 1, 2, 4 and 8 threads. Each thread will hold one connection.

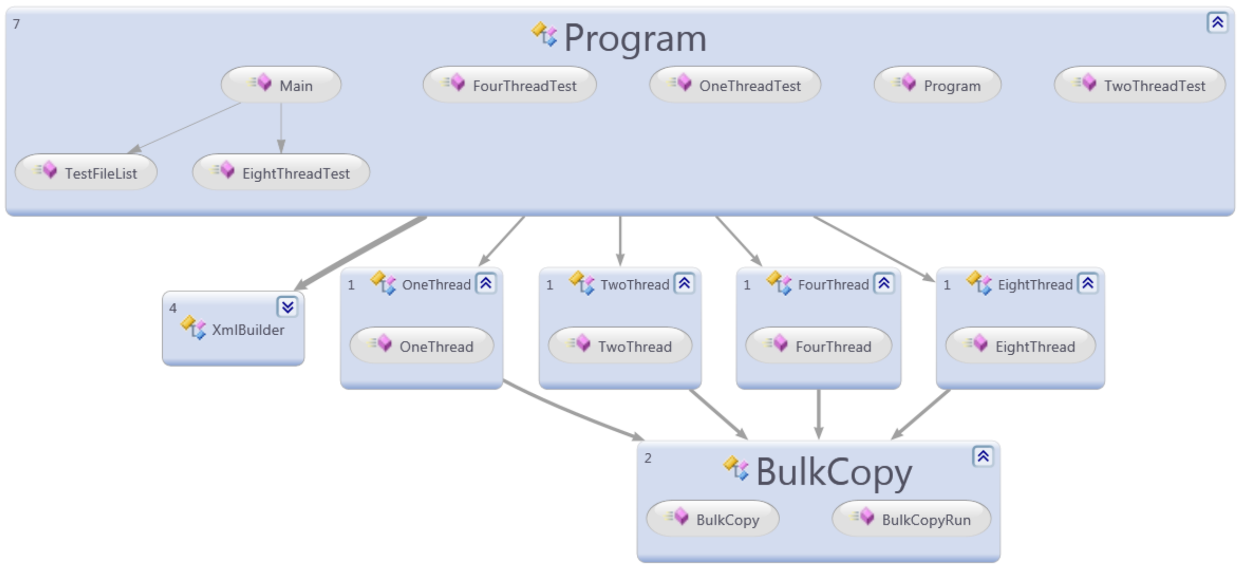


Figure 3-13: Program architecture for multi-threaded bulk copy.

BCP is a specifically build module by Microsoft to do bulk insertion on Microsoft Sql Server. In this task, testing is done to see if there is a performance increase when BCP is being multi-threaded. In theory it should have an improvement in performance. This test will use all the same dataset as all other tests. From here, a comparison between conventional SQL insertion method is being done to see how much performance improvement it provides.

3. 3.9 TASK 4: THREADING WITH IMPORT LOADER

Import loader is similar to BCP, but it receives flat file instead of XML files. It's a module develop by Oracle MySQL itself to support bulk insertion. The dataset file is being separated into multiple flat files according to the number of threads to be used in the test. This process is being done during the data reading process.

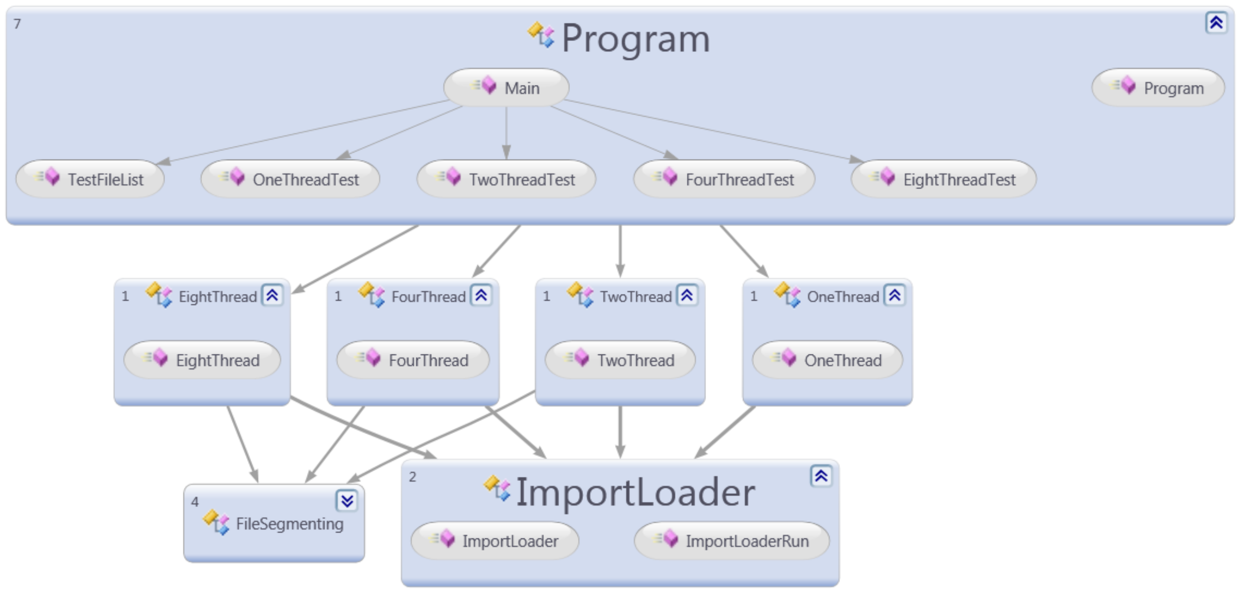


Figure 3-14: Program architecture for multi-threaded import loader.

3. 3.10 TASK 5: MACHINE PERFORMANCE ON LOAD

This test is done to see if database insertion performance would suffer when CPU is at certain load points. Time were taken to measure the performance efficiency. A CPU loading emulator is being used to load the CPU usage. The CPU is being loaded at 25%, 50% and 75%. The same insertion program is being used while the CPU loading emulator is being run on the background.

3. 3.11 TASK 6: MACHINE UTILITY MONITOR

This test would monitor the CPU, RAM and hard-disk utilization throughout the insertion process. In this task a sampling method is being use to monitor the machine utilization. A sample of each item is taken every half a second. To access the utilization, .Net 4 Framework Performance Monitor is being used to monitor the machine utilization. The samples are then being stored in a flat file and an average is taken as the result. The same insertion program used in task 1, 2, 3, and 4 is used in this task.

3.2.12 TASK 7: DEVELOP SMART DAL

This is the final task of the project. From the gathered data, I would then develop a DAL that is able to decide which insertion method to use and how many threads to use for task. The decision would be based on the number of rows to be inserted. The user will have to insert necessary information and also the data to be inserted and the

DAL would decide based on the information gathered according to the database engine used.

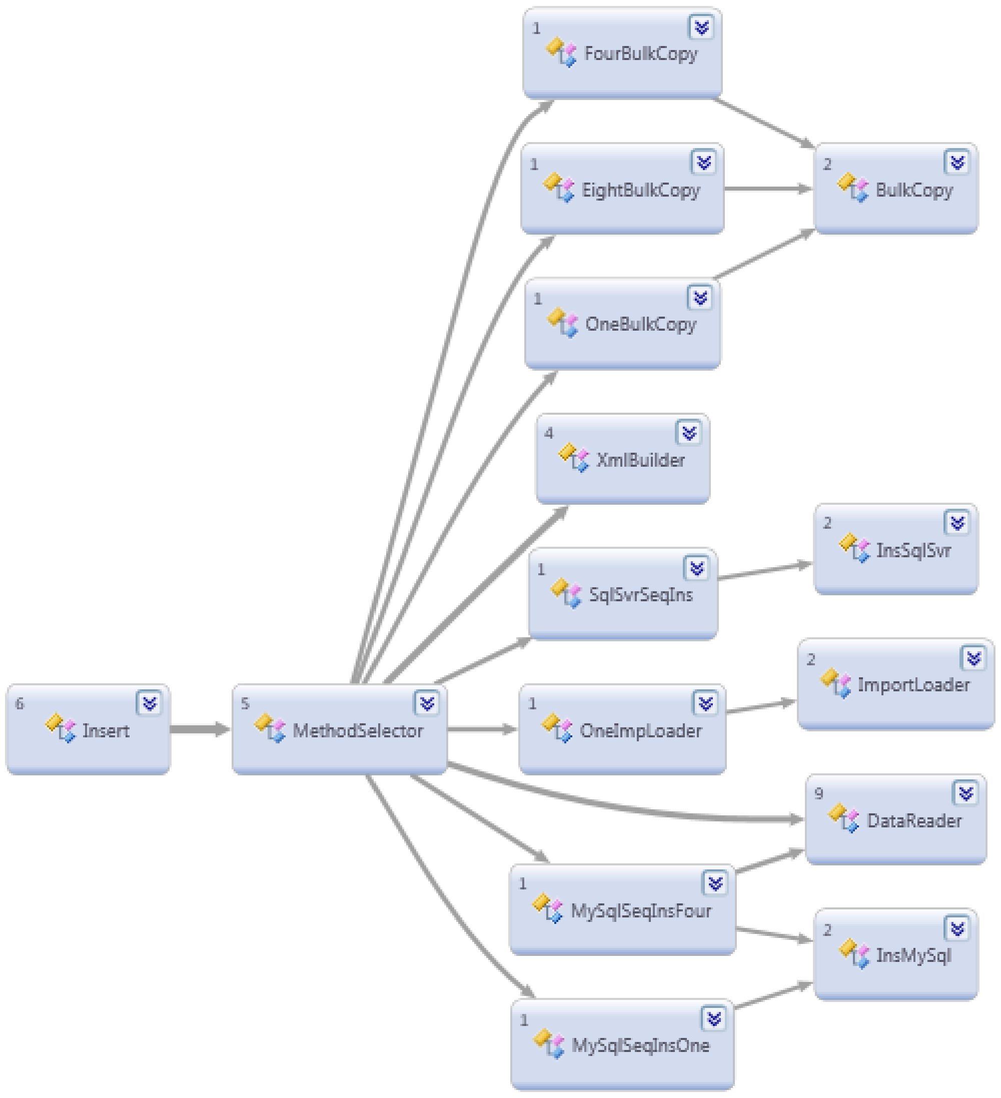


Figure 3-15: Program architecture for Smart DAL.

The user will only have to select the methods in the Insert class and the DAL would perform the insertion task by itself. The decision is being develop based on the test result I gather in task 1, 2, 3 and 4. This is also developed to the specification of one table and one column insertion. Modifications have to be made to accommodate other specifications.

CHAPTER 4 DISCUSSION

4.1 TEST RESULTS

Test results are gathered from the test done referring to Chapter 3.

4.1.1 TEST RESULT FOR SQL SERVER 2008

Test was done on SQL Server 2008 Enterprise edition.

4.1.1.1 PERFORMANCE TEST

no. of row	**single thread***	**2 thread***	**4 thread***	**8 thread***
1	0			
2	0	9		
5	1	12	16	
10	3	18	21	31
15	4	21	16	30
30	9	13	18	33
50	17	23	20	37
100	29	25	26	74
500	147	73	56	68
1,000	316	112	93	125
5,000	1,428	486	93	513
10,000	3,044	954	562	852
50,000	15,068	4,606	2,839	3,633

***** All timing is in milliseconds.

Table 4-1: Table shows the result of SQL sequential insertion

no. of row	**single thread***	**2 thread***	**4 thread***	**8 thread***
1	0			
2	1	8		
5	1	8	15	28
10	3	9	15	28

15	4	9	16	30
30	8	11	16	32
50	14	13	17	32
100	28	24	22	35
500	151	54	42	65
1,000	309	102	68	98
5,000	1,434	463	288	381
10,000	2,834	917	587	736
50,000	23,077	4,518	3714.5	3,781

* All timing is in milliseconds.

Table 4-2: Table shows the result of SQL sequential insertion with transaction code

no. of row	**1 thread bulk copy***	**2 thread bulk copy***	**4 thread bulk copy***	**8 thread bulk copy***
1	4			
2	4	14		
5	4	14	25	
10	5	15	26	52
15	5	16	26	52
30	5	16	26	56
50	6	17	26	58
100	7	18	45	62
500	23	26	51	89
1,000	36	46	134	99
5,000	175	136	154	220
10,000	359	286	282	372
50,000	4,673	6,627	2,689	4,019
60,000	14,557	20,280	9,445	6,723
70,000	20,947	28,538	29,028	11,480
80,000	33,109	41,976	41,207	18,898

* All timing is in milliseconds.

Table 4-3: Table slows the result of SQL Bulk Copy

The additional 60,000, 70,000 and 80,000 rows are only being tested with bulk copy to show the significance in performance improvement when multi-threading is in place.

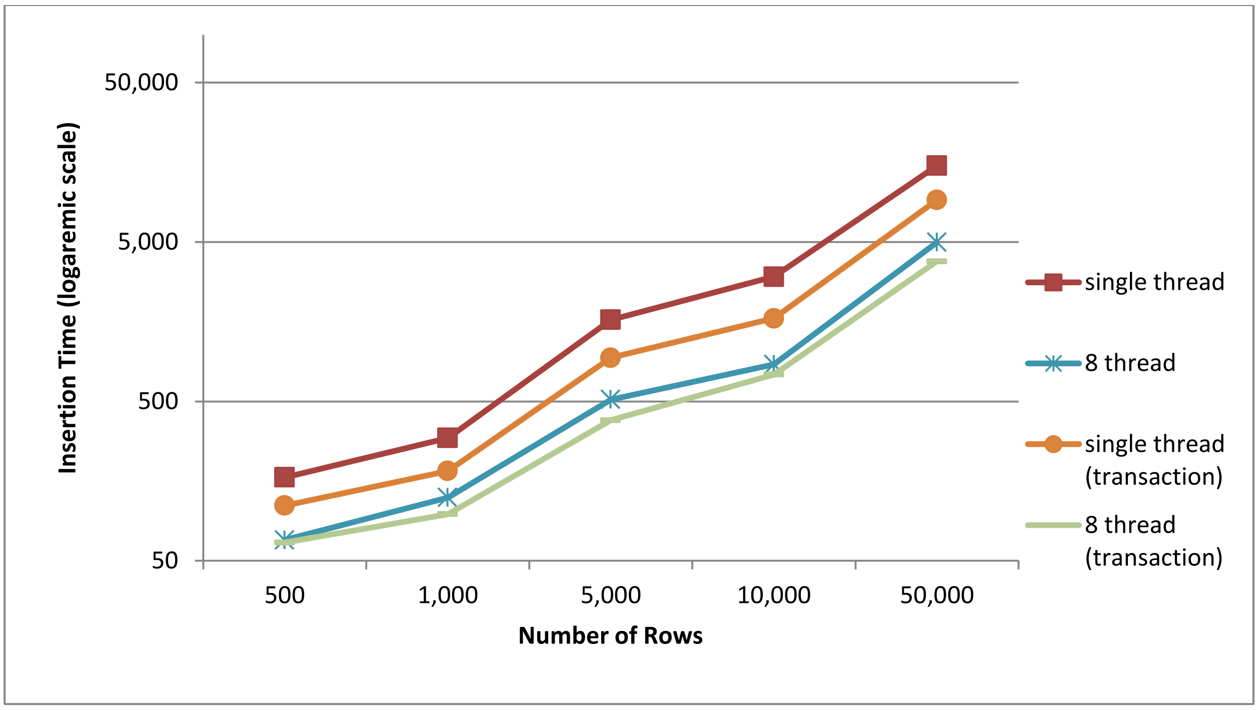


Figure 4-1: Comparison between 1 & 8 threads with and without transaction

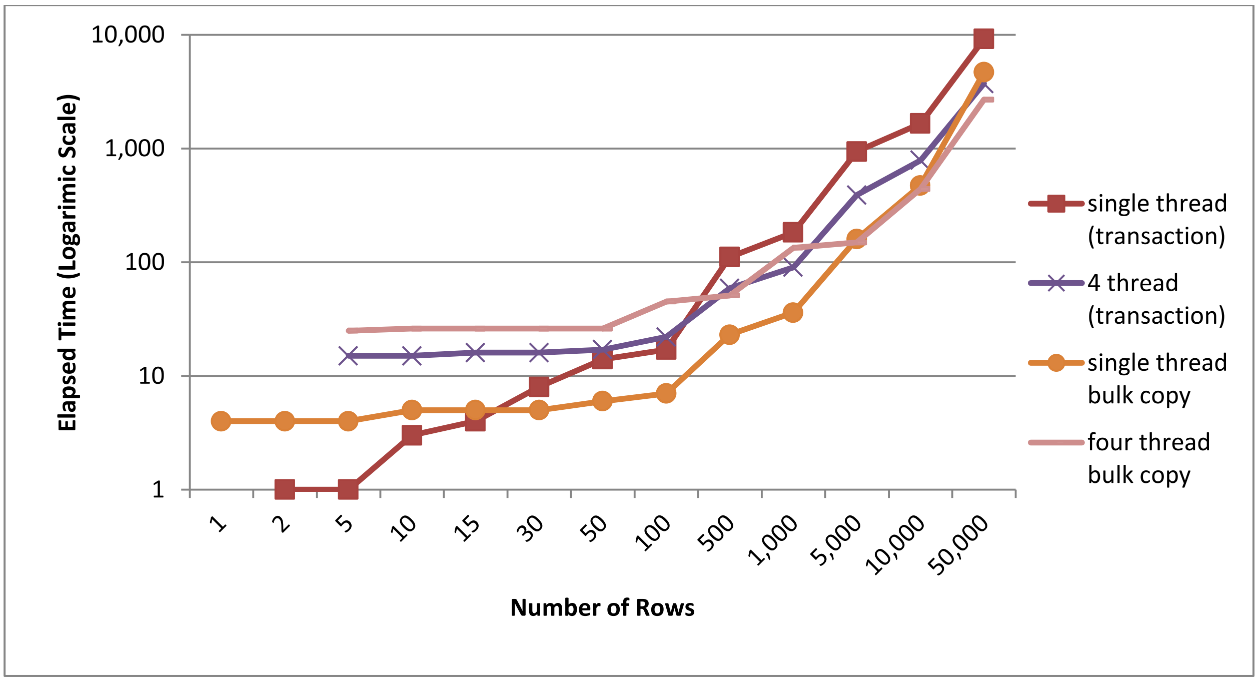


Figure 4-2: Comparison between BulkCopy and sequential insertion with transaction

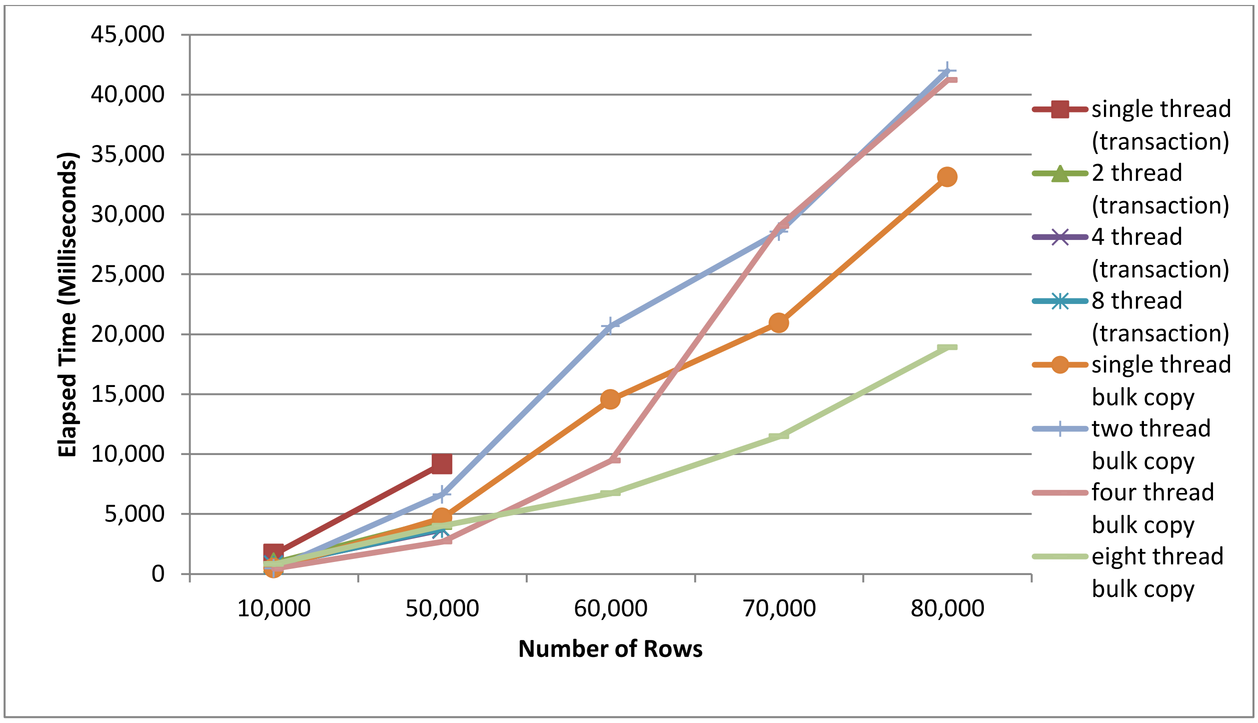


Figure 4-3: Comparison between BulkCopy and sequential insertion with transaction

4.1.1.2 PERFORMANCE DISCUSSION

From the experiments done, we can see from the charts above that multi-threading does help improve bulk insertion performance on SQL Server 2008. There is significant improvement to the performance when the number of rows is above 50,000. At 50,000 rows, it gives at 67% performance boost without the use of transaction. With the use of bulk insertion, at 80,000 rows it gives a 43% performance improvement when comparing single thread to 8 threads. When comparing with and without transaction being used on 8 threads, with transaction gives a 24% improvement. All this shows that multi-threading does help in improving bulk insertion performance.

From the results of the experiments; it is found that different methods suits different amount of data to be inserted. This also depends hugely on the architecture of the database engine. This proves the theory of multi-threading does help in improving database insertion performance. For small amount of data, single threaded with transaction would perform better. This is due to the overhead that multi-threading has. To create and kill a thread and distribute the data would consume too much overhead and it’s too costly to the performance. For the range of 1 to 30 rows, the overhead for threading is too costly and single threaded sequential insertion is more efficient. Refer to Figure 4-5 for more detail. For small amount of data, it is more efficient to use sequential insertion method with transaction.

From research, Bulk Copy only works well when there is a large number of data to be inserted. Bulk Copy is a .NET 4 Framework modules that helps improve bulk insertion performance. It receives input data in XML format. The overhead of converting all the raw data into XML is too costly when the amount of data is too small. At 80,000 rows of data, when compared between single and 8 threads, it gives a 42% performance increase. From figure 0-6 we are able to see a significant improve in insertion performance when there are more than 60,000 rows. Therefore it is best not to multi-thread when the amount of data is below 5001 rows.

From all the research data gathered, we can see that Microsoft SQL Server 2008 supports multi-threading and it does improve its database bulk insert performance. Below is a table with a breakdown on which method to use according to the number of rows.

Number of Rows (Rows)	**Threading Method**
1 to 29	Single shread sequential Insertion with transaction
30 to 5000	Single thread BulkCopy
5001 to 50,000	Four threads BulkCopy
50,000 and above	Eight threads BulkCopy

Table 4-4: Number of threads and insertion method according to the number of rows

4.1.2 My SQL 5.2

4.1.2.1 PERFORMANCE TEST

no. of rows	**one thread**	**two thread**	**four thread**
1	105		
2	108		
5	113		91
10	207		173

15	327		199
30	788		490
50	1,339		650
100	2,441		1,264
500	12,612		6,188
1,000	24,644		12,829
5,000	131,022		61,763
10,000	260,477	247,901	121,724
50,000	1,491,899	1,274,236	

* All times are in milliseconds

* table is not complete as test result is not significant

Table 4-5 : Table of MySQL insertion without transaction code

no. of rows	**one thread**	**two thread**	**four thread**	**eight thread**
1	24			
2	24	49		
5	24	49	66	
10	24	63	76	72
15	24	54	68	78
30	30	54	69	96
50	34	63	59	84
100	43	70	81	88
500	117	109	97	111
1,000	202	160	132	136
5,000	1,018	589	411	458
10,000	2,051	1,050	870	1,115
50,000	11,126	6,265	4,731	5,469
60,000				
70,000				
80,000				

* All times are in milliseconds

* table is not complete as test result is not significant

Table 4-6: Table of MySQL insertion with transaction code

no. of rows	1 thread insert loader	2 thread insert loader	4 thread insert loader	8 thread insert loader
1	93			
2	114	78		
5	140	146	96	
10	160	211	177	101
15	180	281	254	203
30	200	355	332	298
50	220	429	406	395
100	246	502	489	490
500	289	652	575	600
1,000	338	734	673	740
5,000	461	1,130	1,089	1,314
10,000	670	1,671	1,886	1,987
50,000	2,564	4,016	3,894	3,867
60,000	6,276	8,391	8,985	8,970
70,000	10,314	13,842	15,130	15,265
80,000	15,072	20,003	21,783	23,063

* All times are in milliseconds

* table is not complete as test result is not significant

Table 4-7: Table of MySQL insertion using insert loader

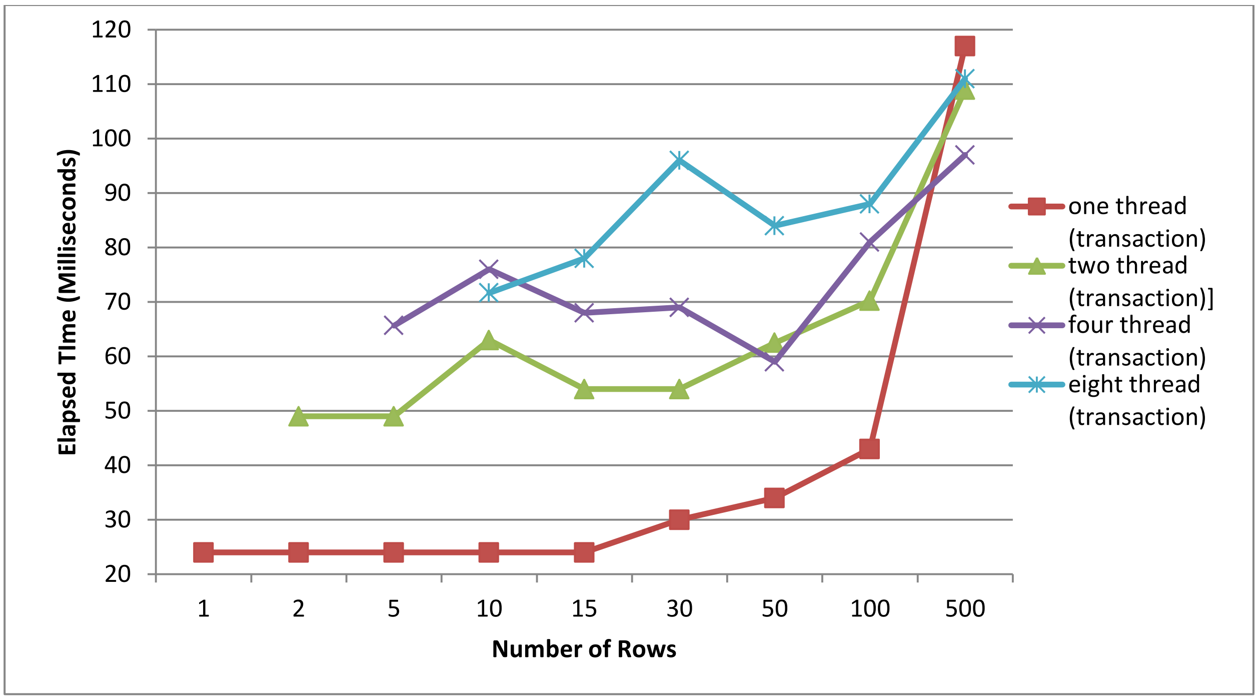
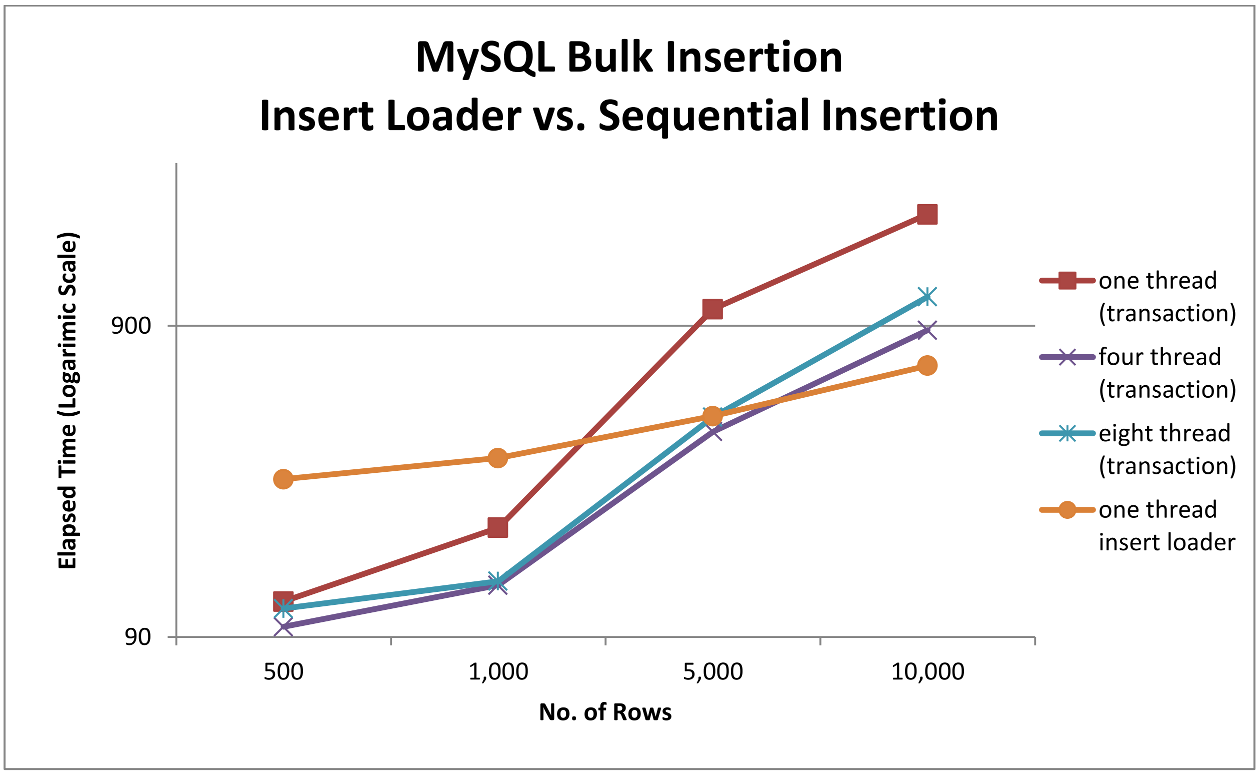


Figure 4-4: Comparison between different numbers of threads using sequential insertion



4-5: Comparison between insert loader and sequential insertion at 500 to 10,000 rows

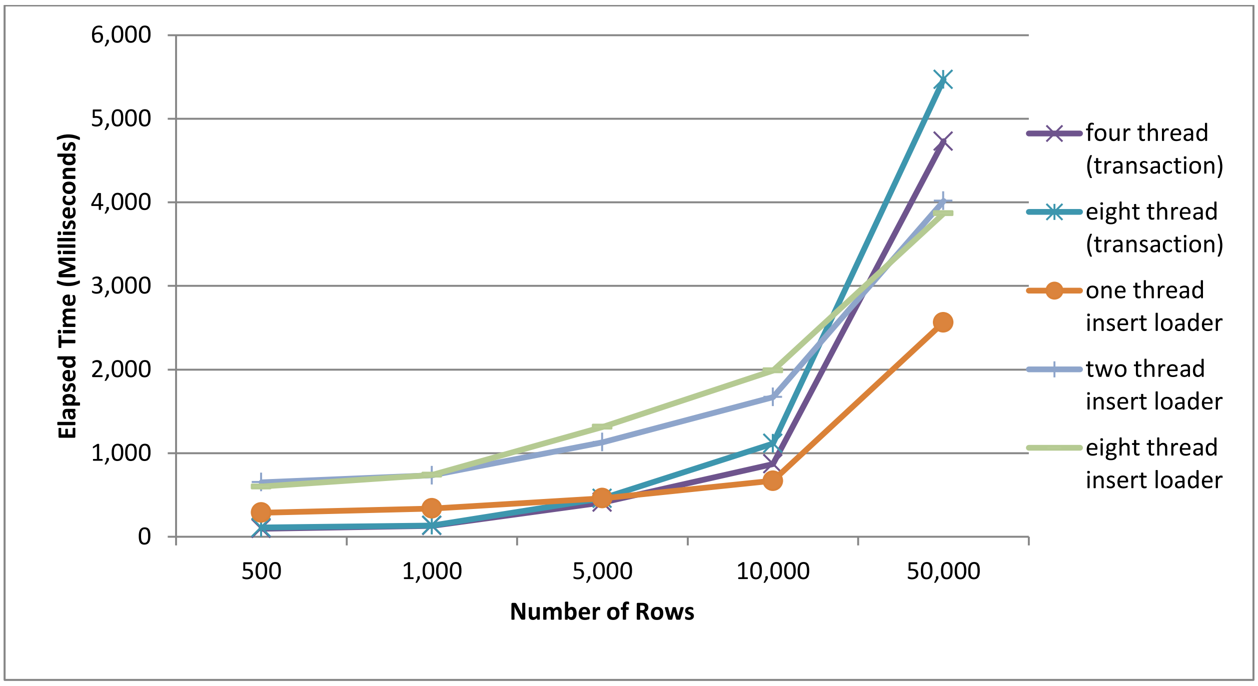


Figure 4-6: Comparison between sequential insertion with transaction and insert loader

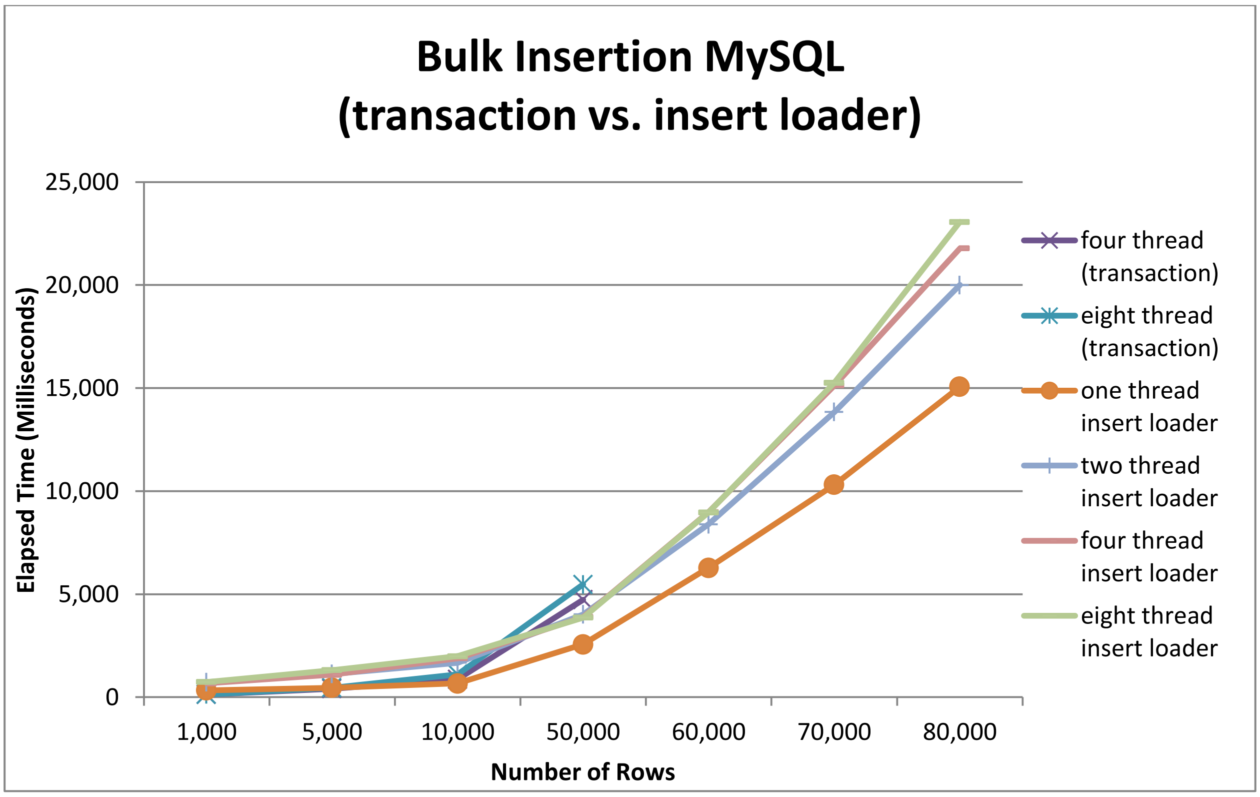


Figure 4-7: Comparison between sequential and insert loader from 1,000 to 80,000 rows

4.1.2.2 PERFORMANCE DISCUSSION

From the experiment's that done with MySQL, it does not work really well with multi-threading. Threading just slows the process down. This proves that multi-threading does not apply to all database system and its database engine dependent.

MySQL works very well with transaction codes; it's not efficient without transaction code. When comparing with and without transaction codes at 50,000 rows on single thread there is a 99.25% performance improvement with transaction being used. To insert 50,000 rows using sequential insertion without transaction codes, it requires approximately 22 minutes compared to 6.3 seconds with transaction codes. In the data range of 1 to 500 rows, single threaded with transaction works most efficiently. This could be referred in Figure 0-10. From the graph patterns, we could see that the more threads being used, the less efficient it gets. But when the number of rows grows from 500 and above, four threads works more efficiently compared to single threaded. This could be due to the architecture and threading overheads. At 500 rows, four threads give a 17.09% performance improvement which gradually increases to 59.63% at 5,000 rows.

At more than 5,000 rows, single threaded insert loader works best. Comparing between single and eight threaded insert loader; single threaded has a 64.92% performance advantage at 5,000 rows. At 80,000 rows single threaded has a 34.65% advantage over 8 threads insert loader. This could be seen in Figure 0-13. From all the findings, that the results shows that MySQL does not work well with multithreading expect for certain environment or situation. This proves that multi-threading on database insertion depends on the architecture it was developed on. This could also because MySQL database engine is already multi-threaded and if threading is done on the application level, it would cause a drop in performance.

Because of this scenario, it is not advisable to thread the whole system without test before implementation. Therefore it is best not to multi-thread MySQL, except for the data range of 501 to 5000.

From all the data gathered from the tests, the optimal threading methods according to the number of rows. It is shown in Table 4-8.

Number of Rows (Rows)	Number of threads and insertion method
1 to 500	Single thread sequential insertion with transaction
501 to 5000	Four threads sequential insertion with transaction
5001 and above	Single thread insert loader

Table 4-8: Insertion and threading method according to amount of data

4.2 Performance Monitoring

4.2.1 SQL Server 2008 Enterprise

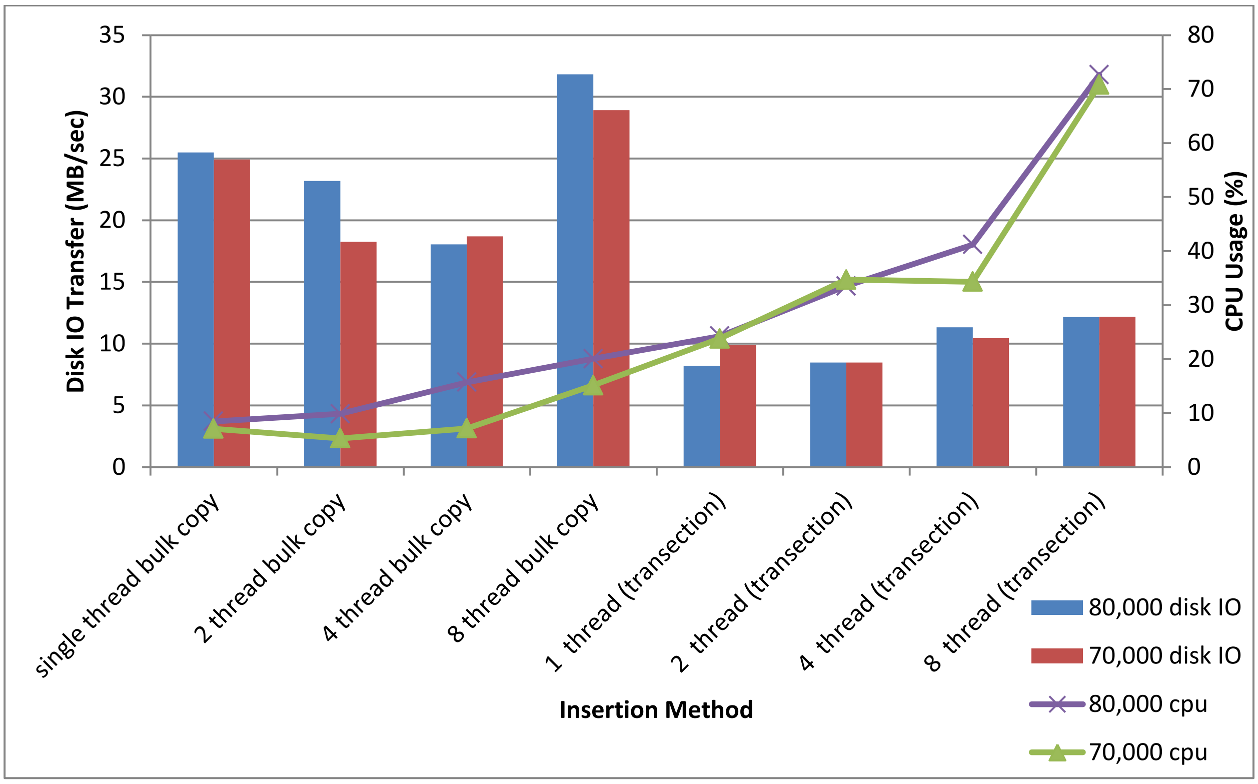


Figure 4-8: System utilization between Disk IO transfer and CPU Usage for SQL Server 2008

Referring to Figure 4-8, we can see that the CPU usage increase as the number of threads increases. This shows that the system is utilizing the available processing power of the CPU and multi-threading is at work. The CPU usage increases significantly at 8 threads. From the figure above we could also see that bulk copy consumes less processing power and this would be very useful in environments where the server is busy. This information is particularly useful when the server is being shared by other application on virtual machine and the CPU is constantly at high usage. Bulk Copy could solve such problem where if sequential insertion is being used, it will be inefficient. We could see that Bulk Copy does not consume much processing power even at 8 threads. This proves that this module supports multi-threading with reference to the discussion in 4.1.1.2.

The magnetic hard disk usage did not reach maximum as claim capable from the manufacturer. It only achieves 50% of its capability as maximum capability claim by Seagate (Seagate Barracude 7200.10 SATA 3.0Gb/s 320-GB Hard Drive, 2010). This could be due to the architecture that the database engine was developed and restrict write performance or it has its own I/O limitations. Further research has to be done on this factor to maximize the magnetic disk I/O which is believe would further enhance the INSERT performance.

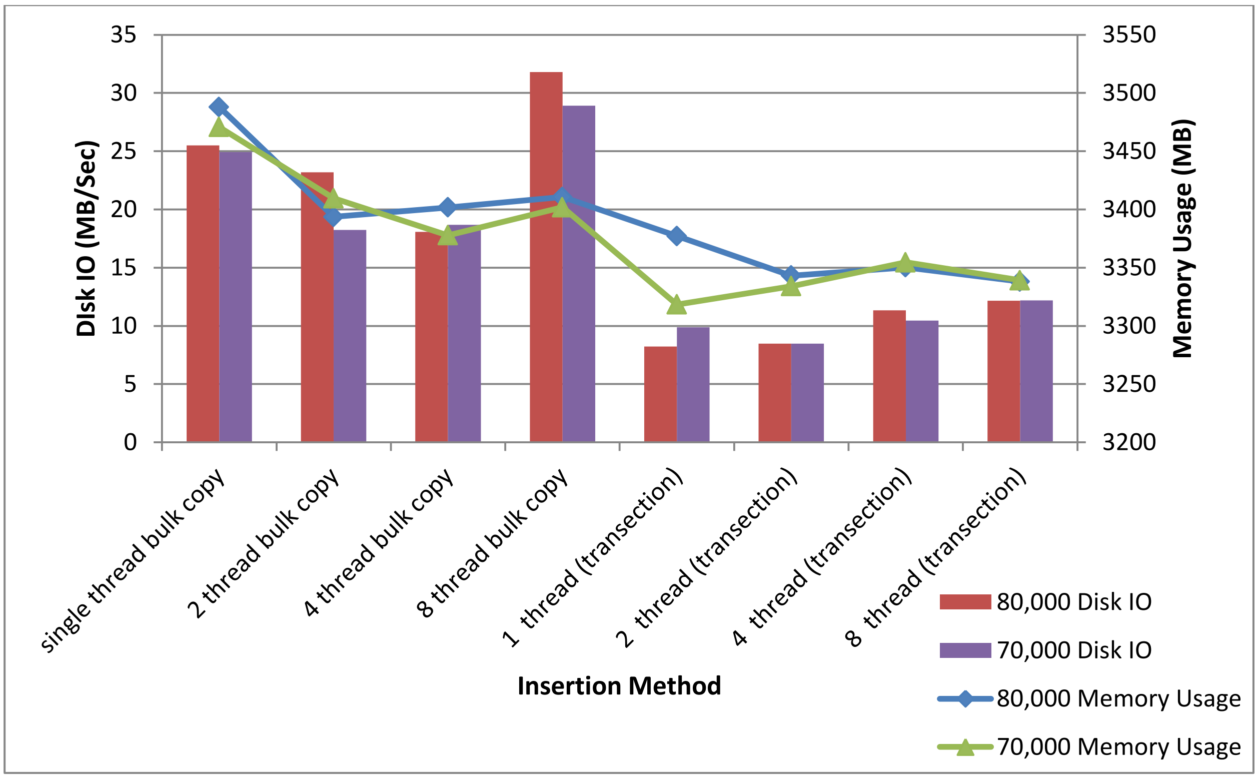


Figure 4-9: System utilization between Disk IO transfer and RAM usage for SQL Server 2008

From Figure 4-9 we can see that the RAM usage decreases the number of threads used increases. This is because the amount of backlogs is being reduced. This is a positive sign for developers as the INSERT statement won't be RAM hungry. On single thread, the RAM usage is high due to the huge amount of data being stored in the RAM waiting to be inserted. From the analysis, bulk copy consumes more RAM compared to sequential insertion and this could be due to how the module was being developed. The difference between the highest and lowest of the RAM usage is very slim at only 4.3% of difference. The average RAM usage is at 3.3 GB. We the result we could see that Bulk Copy consumes more RAM, less CPU usage and it gives the highest disk I/O rate.

From all the analysis above, it can be concluded that there is a close relationship between HDD I/O, RAM, and CPU usage and bulk insertion process. When there is insufficient RAM, there will definitely be a drop in insertion performance. This goes the same when the CPU is busy. HDD I/O is the core and if there is insufficient I/O, there will be a bottleneck on the HDD. This is because all the data is being channeled into the disk at the same time. For more information, section 4.3 would show a simulated effect when CPU is being loaded at certain percentage of usage.

4.2.2 MySQL 5.2

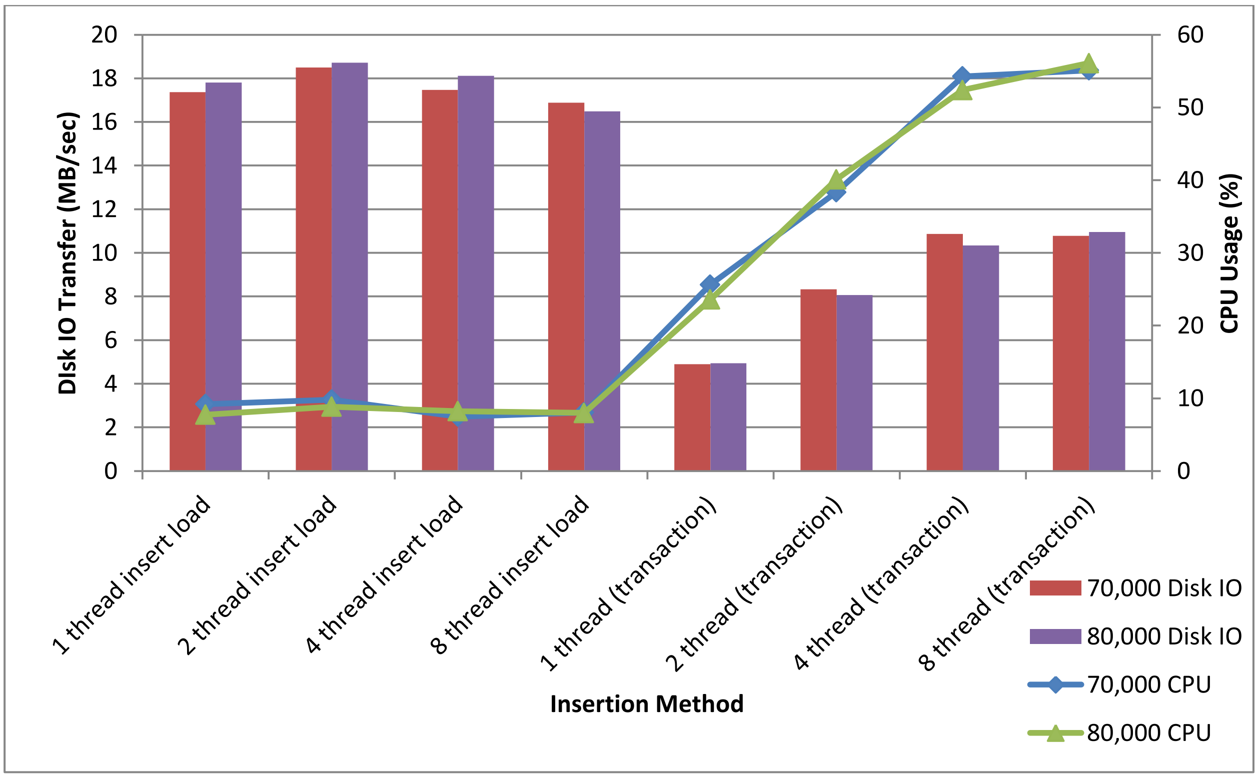


Figure 4-10: System Utilization between Disk IO Transfer and CPU usage for MySQL 5.2

From Figure 4-10 we can see a rather similar pattern found in SQL Server 2008. The CPU usage is relatively low compared and its usage does not vary a lot when compared between single and 8 threads insert loader. This proves that MySQL is light on CPU usage. It is a surprise to know that there is no significant increase in CPU usage when it's being threaded. From the graph, the disk IO is highest at 2 thread insert loader but performance timing proves otherwise. This could be the insertion rate is not constant and cause of it is not known at this moment of research. The disk I/O usage is below 50% of what is claim capable by Seagate (Seagate Barracude 7200.10 SATA 3.0Gb/s 320-GB Hard Drive, 2010). This could be due to the architecture of the database engine. To further enhance and know why it's limited to this rate, further analysis on the database engine's architecture has to be done. From the graph we can see that the performance variation between different numbers of threads for insert loader is not very big. This show's that multi-threading does not help a lot or contributes very little to MySQL bulk insertion process. For sequential insertion method, the disk IO does improve but not the performance time, as before it could be some I/O architecture that is limiting the performance.

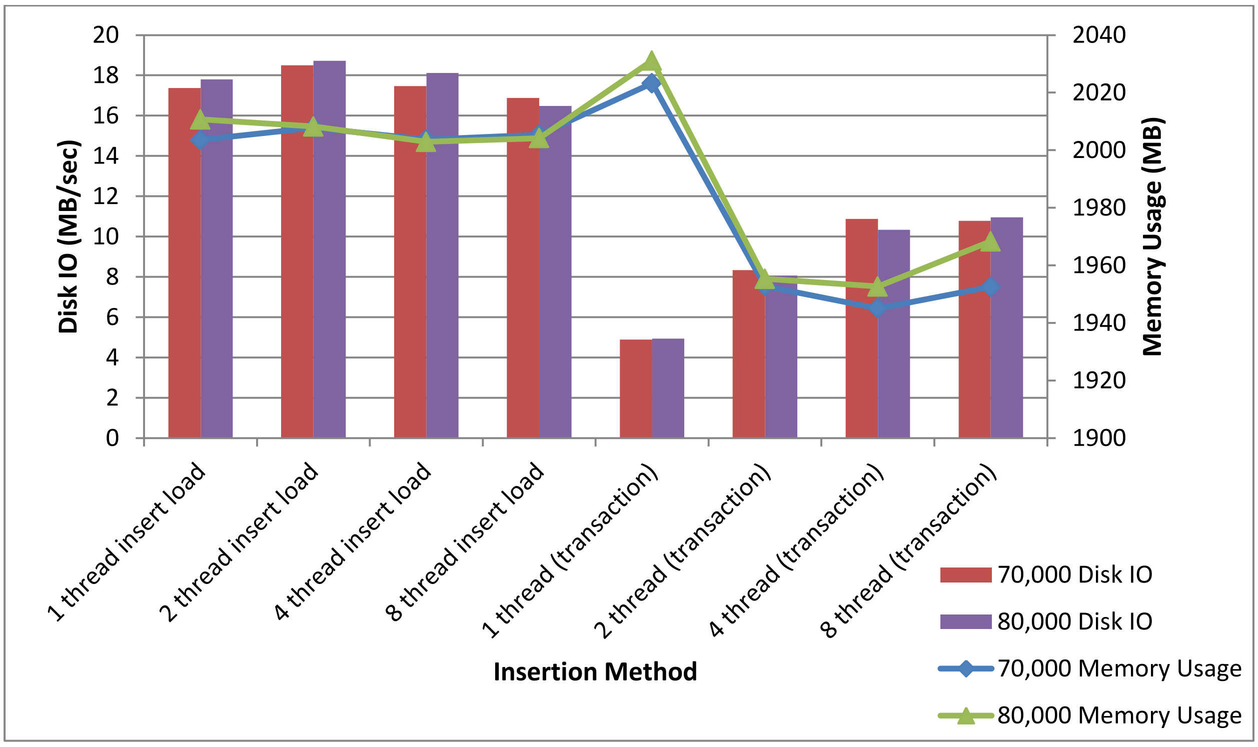


Figure 4-11: System utilization between Disk IO transfer and Memory usage for MySQL 5.2

The pattern of the RAM usage is almost similar to that of SQL Server 2008. When an insert loader is being used, the RAM usage is higher compared to sequential insertion methods. The usage average out to be almost the same for singe, 2, 4 and 8 threads. This show that multi-threading does not really take effect on the database engine. For sequential insertion, an almost similar scenario as of SQL Server 2008 applies here.

From the analysis above, we could see the relationship between hard disk I/O, RAM and CPU towards the performance of bulk insertion. When the CPU is busy, it greatly affects the performance of bulk insertion. There has to be sufficient RAM for the whole process, this is because such task require huge amount of RAM. HDD performance clearly plays a significant role and is the core to bottleneck and it's closely related to the bulk insertion performance. For more details, section 4.3 would show a simulated effect when the CPU is being loaded.

4.3 Affects on Performance

This experiment was done but the result was too difficult to analyze and another approach was taken to gather some more accurate results which is being presented in chapter 4.2.

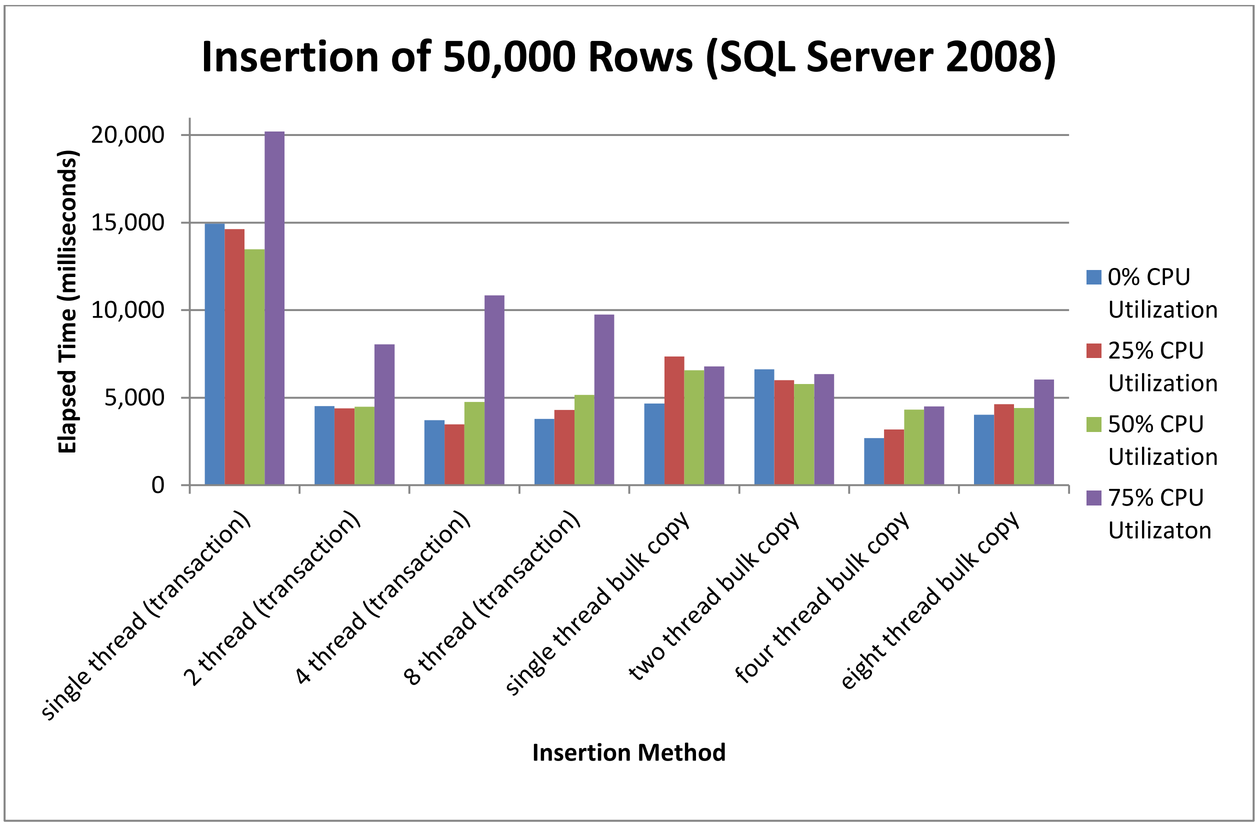


Figure 4-12: Affect on performance at different CPU usage for SQL Server 2008

Figure 4-12 shows that when the CPU is at load of 75% utilization, the insertion performance suffers. This proves that multi-threading is at work and SQL Server does require a significant amount of CPU power to execute the task efficiently. The pattern of the graph is very difficult to analyze because the processor shift its jobs from one thread to another to distribute the loads. Where else when it's on single thread, only one core of the processor is working. When multi-threading is applied, we can see that there is a reduction in time and the CPU does load balancing. Referring to single thread with transaction and comparing it with the other sequential insertion which is being threaded, we can see the improvement in performance multi-threading gives.

From Figure 4-12 it is known that bulk copy requires more processing power. With the increase in CPU usage, there is an effect on the performance. SQL Server 2008 insertion performance is related to the CPU performance. When CPU is busy, it will affect the bulk insertion performance.

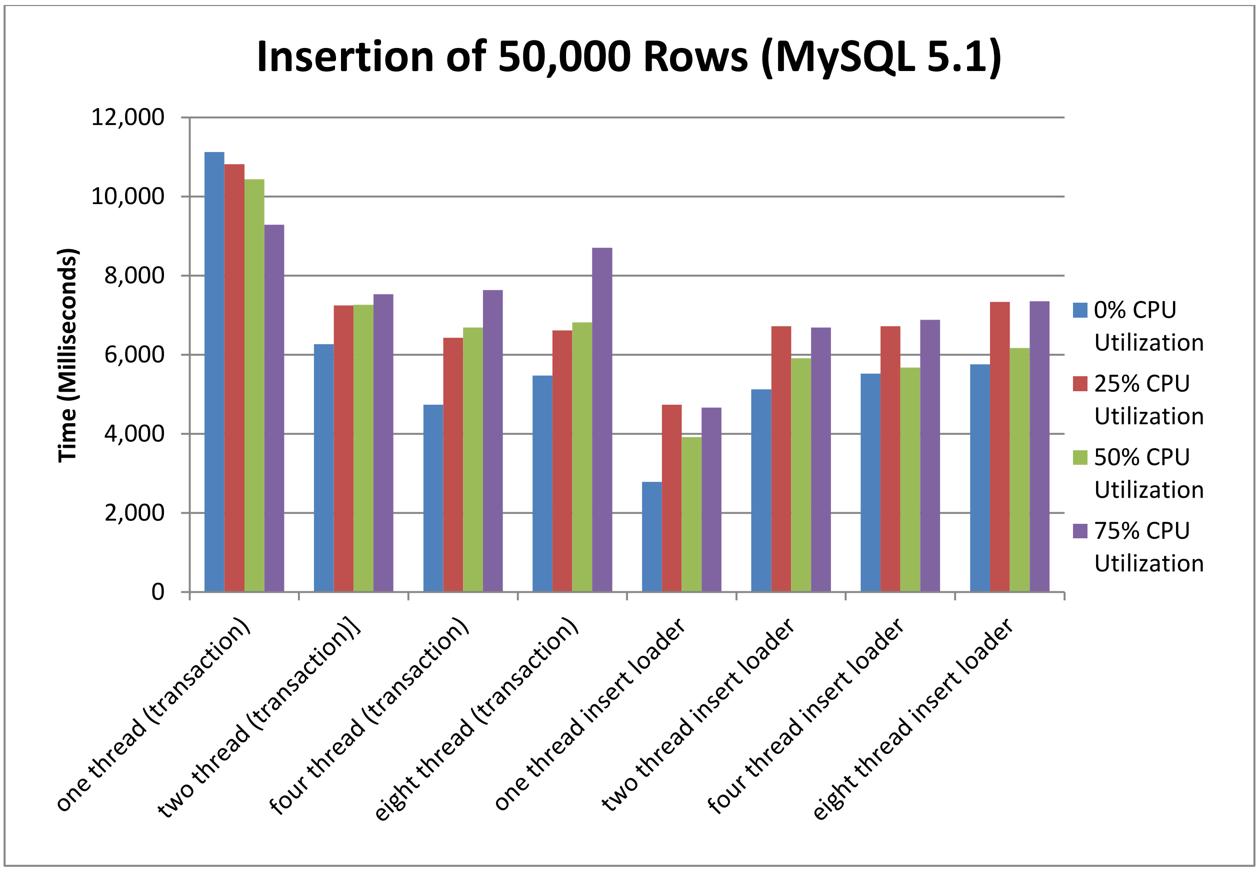


Figure 4-13: Affect on performance at different CPU usage for MySQL

From Figure 4-13 it shows similar effect as with SQL Server 2008 here on MySQL. MySQL uses lesser processing power and we could see the effect on the performance is not as significant as in SQL Server 2008. Even on MySQL which does not really support multi-threading, even when it’s being threaded and CPU load increase it has a significant effect on the bulk insertion performance. There is a relationship between the CPU performances towards Insertion performance.

From the results gathered from the two experiments above is too difficult to draw a conclusion. These are caused by many factors that create lots of anomalies in the result. It could be cause by the OS scheduling, background processes and many others. Due to the lack of fact and proving on why such events are happening, therefore there is no conclusion for this experiment and it has been aborted. To replace this experiment, Section 4.2 is being done to conduct the test on performance relationship and its effect on the bulk insertion performance.

CHAPTER 5 CONCLUSION

Although, the advancement of multicore processors is encouraging multithreaded application to be developed, we found that the performance of the insertion function of a database does not necessary improve proportionately with the number of threads used. Multithreading did improve the performance of both of the databases' insertion function but the speed up is very dependent on the underlying architecture of the database system. Therefore, this work suggests that software developers should investigate the performance of multithreaded operations on databases before designing any system.

Design of a multi-threaded database bulk insertion solution should depend on the database engine as well as the machine it is being implemented on. Database architecture, RAM, CPU, and type of HDD do have an effect on each other. Different database engine react differently toward different approach into doing bulk insertion. From all the test results, it is shown that if it is not being threaded in an efficient manner, it will cause a major drop in performance.

The DAL is being developed with a couple of limitation, reasons for this limitation not being solves is due to time constrain, resource and also beyond the scope of the project. Following are the limitation:

- Limit data size to 1.2 GB, out of memory error will occur if file size is larger than the limit.
- Threading method and solutions are being based on Intel Core 2 Quad machine.
- Supports only 2 database machine.
- DAL only accepts array or text file as input data.

We hope that this research finding has contributed to the database developers and also system developers into ways of improving database bulk insertion.

5.1 Future Works

There is much work to be done with much more questions to be answered. Following are the future works to be done:

- Test model on different CPU platform.

- Test model on different type of HDD, flash drive or solid state HDD.
- Test model on different database engines.
- Test on different threading methods.
- Does virtual machine has an impact on database bulk insertion? This question is being directed to those environments where virtual machine is being used to utilize server capabilities.
- What effect does it has when RAM is being loaded or HDD is being kept busy while bulk insertion process is running.

REFERENCES

Broberg, M. (2000). Performance Tuning of Multithreaded Applications.

Bulk Copy Operations in SQL Server (ADO.NET). (2011). Retrieved January 5, 2011, from Microsoft MSDN: http://msdn.microsoft.com/en-us/library/7ek5da1a.aspx

Bunn, J. J. (2000). Object Database Scalability for Scientific Workloads. In J. J. Bunn, *Object Database Scalability for Scientific Workloads.* Geneva.

Donald, R. (2010). *Rad Software* . Retrieved August 1, 2010, from Data Access Layer Design: http://www.radsoftware.com.au/articles/dataaccesslayerdesign1.aspx

Etheredge, J. (2010, January 27). *CodeThinked*. Retrieved August 11, 2010, from .NET 4.0 and System.Collections.Concurrent.ConcurrentBag: http://www.codethinked.com/post/2010/01/27/NET-40-and-System_Collections_Concurrent_ConcurrentBag.aspx

Granatir, O. (2009). *OMG, Multi-Threading is Easier Than Networking.* Intel Corp.

Gray, D. J. (1992). Parallel Database Systems: The Future of High Performance Database Processing.

Harinath, M. M. (2006, February 07). *Developer Fusion*. Retrieved August 6, 2010, from Bulk Insert from Flat File Using Transact SQL: http://www.developerfusion.com/code/5357/bulk-insert-from-flat-file-using-transactsql/

Harper, M. (2002, Jun 12). *Developers Articles*. Retrieved August 6, 2010, from Sql Server Bulk Copy: http://www.devarticles.com/c/a/SQL-Server/An-Introduction-To-The-Bulk-Copy-Utility/1/

Intel Processor Roadmap. (2010). Retrieved August 9, 2010, from Processors Platform Roadmap: http://edc.intel.com/Platforms/Roadmap/

J.D. Meier, A. H. (2009, jan). *Chapter 12: Data Access Layer Guidelines*. Retrieved July 16, 2010, from Microsoft Pattern & Practice: http://apparchguide.codeplex.com/wikipage?title=Chapter%2012%20-%20Data%20Access%20Layer%20Guidelines&referringTitle=Home%20Page%203

J.D. Meier, A. H. (2009). *MSDN Library (DAL)*. Retrieved from Chapter 12 Data Access Layer Practice: http://apparchguide.codeplex.com/wikipage?title=Chapter%2012%20-%20Data%20Access%20Layer%20Guidelines&referringTitle=Home%20Page%203

Java on Solaris 7 Developer's Guide. (1998). Califonia: Sun Microsystem, Inc.

Jingren Zhou, e. a. (2005). Improving Database Performance on Simultaneous Multithreading Processors.

Julian J. Bunn, K. H. (2007). *Object Database Scalability for Scientific Workloads.* Geneva.

Kanjilal, J. (2007, October 27). *DonNetJohn*. Retrieved August 1, 2010, from Implemting a Data Access Layer in C#: http://www.dotnetjohn.com/articles.aspx?articleid=244

Ltubia. (2008, November 20). *CodePlex*. Retrieved August 6, 2010, from Multithreaded Bulk Copy: http://mbulkcopy.codeplex.com/

Lundberg, D. H. (1999). Multiprocessor Performance Evaluation of a Telecommunication Fraud Detection Application.

Lundberg, D. H. (1999). Multiprocessor Performance Evaluation of a Telecommunication Fraud Detection Application.

Marchand, B. (2008, January). Multi-Core Processing Advancements via Optimized System Resource Allocation and Capacity Management. p. 2.

Maximum Capacity Specifications for SQL Server. (2010). Retrieved July 2010, 10, from MSDN Microsoft Inc.: Maximum Capacity Specifications for SQL Server

Moore, B. (2008). *What Does It Mean to be I/O.* Intel Corporation.

MSDN Library (Bulk Insert). (2010). Retrieved 2010, from Bulk Insert Task: http://msdn.microsoft.com/en-us/library/ms141239.aspx

MSDN Library (Threading). (2010). Retrieved August 6, 2010, from Managed Threading Best Practices: http://msdn.microsoft.com/en-us/library/1c9txz50.aspx

MSND Library (Bulk Insert). (2010). Retrieved from Bulk Insert (Transact-SQL): http://msdn.microsoft.com/en-us/library/ms188365.aspx

MySQL Connection Pooling. (2010). Retrieved July 6, 2010, from Using Connector/J with J2EE and Other Java Frameworks: http://dev.mysql.com/doc/refman/5.1/en/connector-j-usagenotes-j2ee.html#connector-j-usagenotes-j2ee-concepts-connection-pooling

MySQL Overview. (n.d.). Retrieved July 1, 2010, from Oracle: http://www.sun.com/software/products/mysql/

MySql Speed of INSERT Statement. (2003, May). Retrieved August 1, 2010, from Speed of INSERT Statement: http://dev.mysql.com/doc/refman/5.0/en/insert-speed.html

Performance Counter Constructor. (2010). Retrieved January 10, 2010, from MSDN Library: http://msdn.microsoft.com/en-us/library /xx7e9t8e.aspx

PowerEdge R905. (2010). Retrieved November 10, 2010, from Dell: http://configure.us.dell.com/dellstore/config.aspx?oc=becw7vs&c=us&l=en&s=bsd&cs=04&model_id=poweredge-r905

Rubbelke, L. (2008). *SQL Server 2008 white paper.* Califonia: Microsoft Corp.

Ryan Johnson, I. P. (2001). Shore-MT: A Scalable Storage Manager for the Multicore Era. 24-36.

Ryan Johnson, I. P. (2009). Shore-MT: A Scalable Storage Manager for the Multicore Era. 24-35.

Scott R. Taylor,, D. J. (2008). A Comparison of Multithreading Implementations. 1-8.

Seagate Barracude 7200.10 SATA 3.0Gb/s 320-GB Hard Drive. (2010). Retrieved November 18, 2010, from Seagate: http://www.seagate.com/ww/v/index.jsp?vgnextoid=2d1099f4fa74c010VgnVCM100000dd04090aRCRD

Speed of INSERT Statement. (2010). Retrieved August 30, 2010, from MySQL: http://dev.mysql.com/doc/refman/5.0/en/insert-speed.html

SQL Server 2008 Product Information. (n.d.). Retrieved Jun 1, 2010, from Microsoft: http://www.microsoft.com/sqlserver/2008/en/us/benchmarks.aspx

Thread Class. (n.d.). Retrieved December 22, 2010, from MSDN Library: http://msdn.microsoft.com/en-us/library/system.threading.thread.aspx

TransactionScope Class. (n.d.). Retrieved December 20, 2010, from MSDN Library: http://msdn.microsoft.com/en-us/ library/system.transactions. transactionscope.aspx

Using the Bulk Loader. (2010). Retrieved January 2, 2010, from MySQL: http://dev.mysql.com/doc/refman/5.1/en/connector-net-programming-bulk-loader.html

Vadaparty, K. (2008). Multithreaded Parallelism in Windows Workflow Foundation. *.NET Development (Windows Workflow) Technical Articles* , 1-28.

Valduriez, 1. M. (1991). Distributed and Parallel Database System.

Valduriez, P. (1993). "Parallel Database Systems: Open Problems and New Issues". *Distributed and Parallel Databases* , 137-165.

Verenkar, A. (2010, April 9). Using .Net4 Parallel Programming Model to Achieve Data Parallelism in Multi-tier Application. *MSIT, Microsoft Corporation* , pp. 1-13.

Zukowski, M. (2005). Hardware-Conscious DBMS Architecture for Data-Intensive Application. 55-59.

APPENDIX

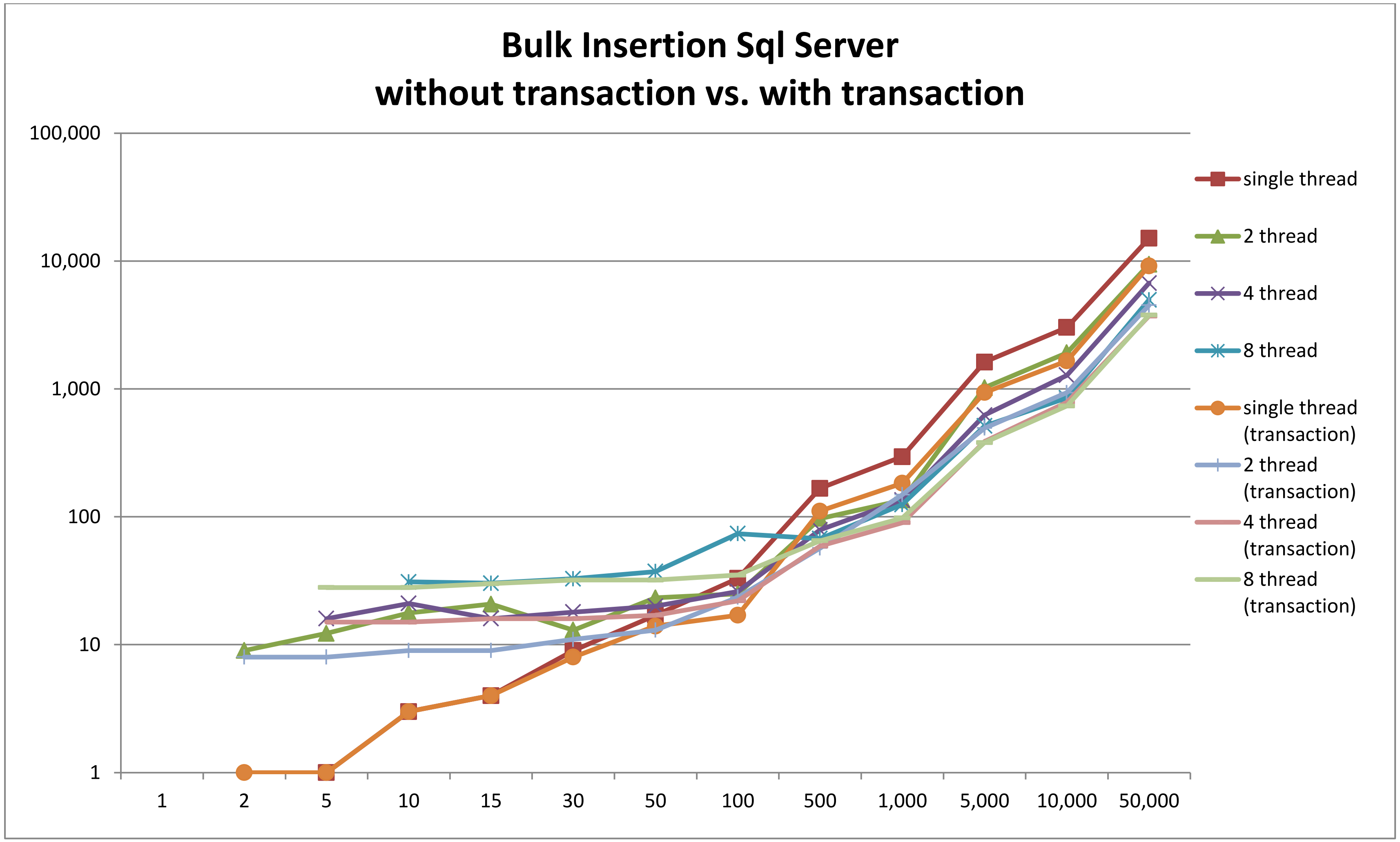


Figure 0-1: Transaction vs. without Transaction chart

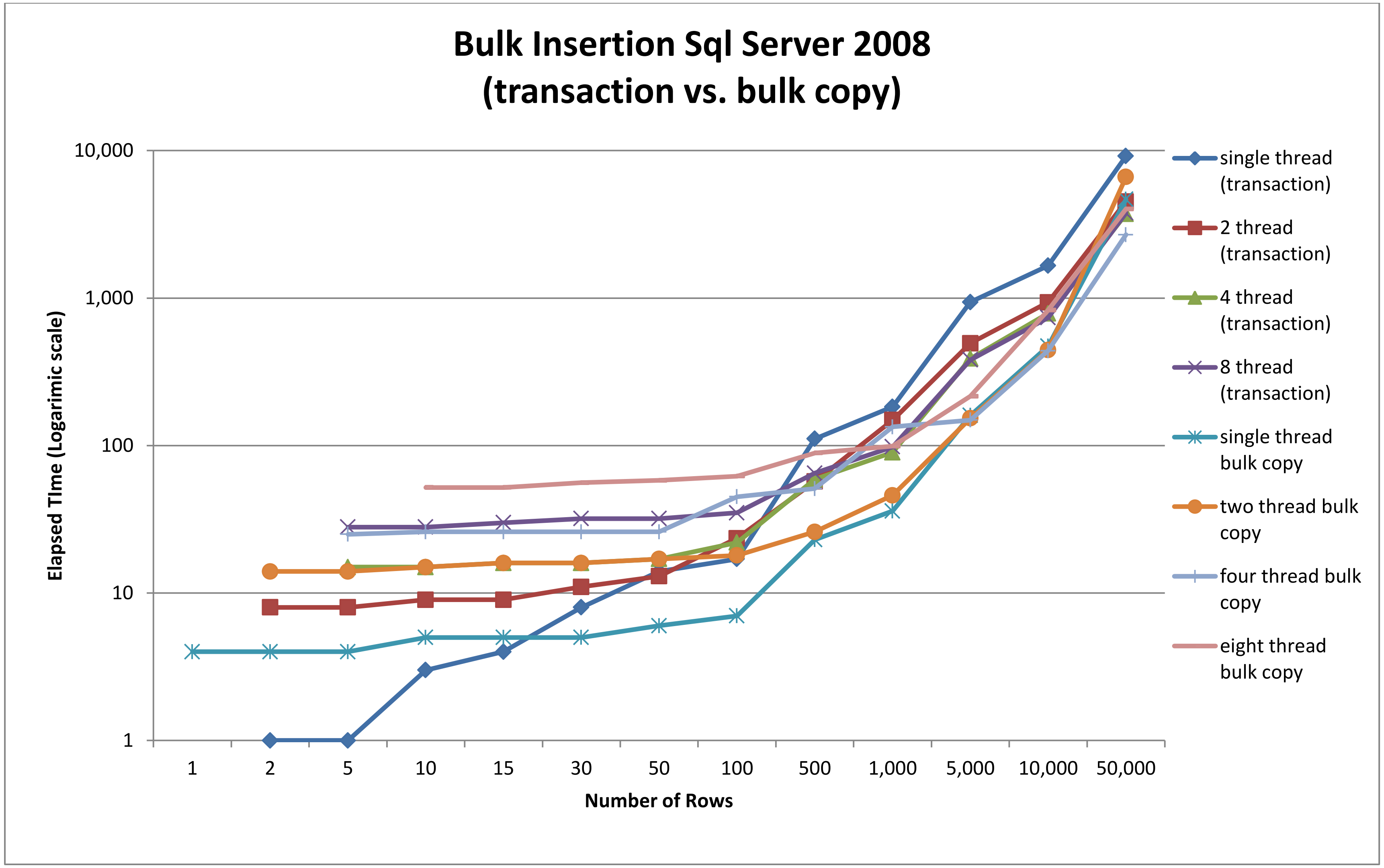


Figure 0-2: Transaction vs. BulkCopy chart

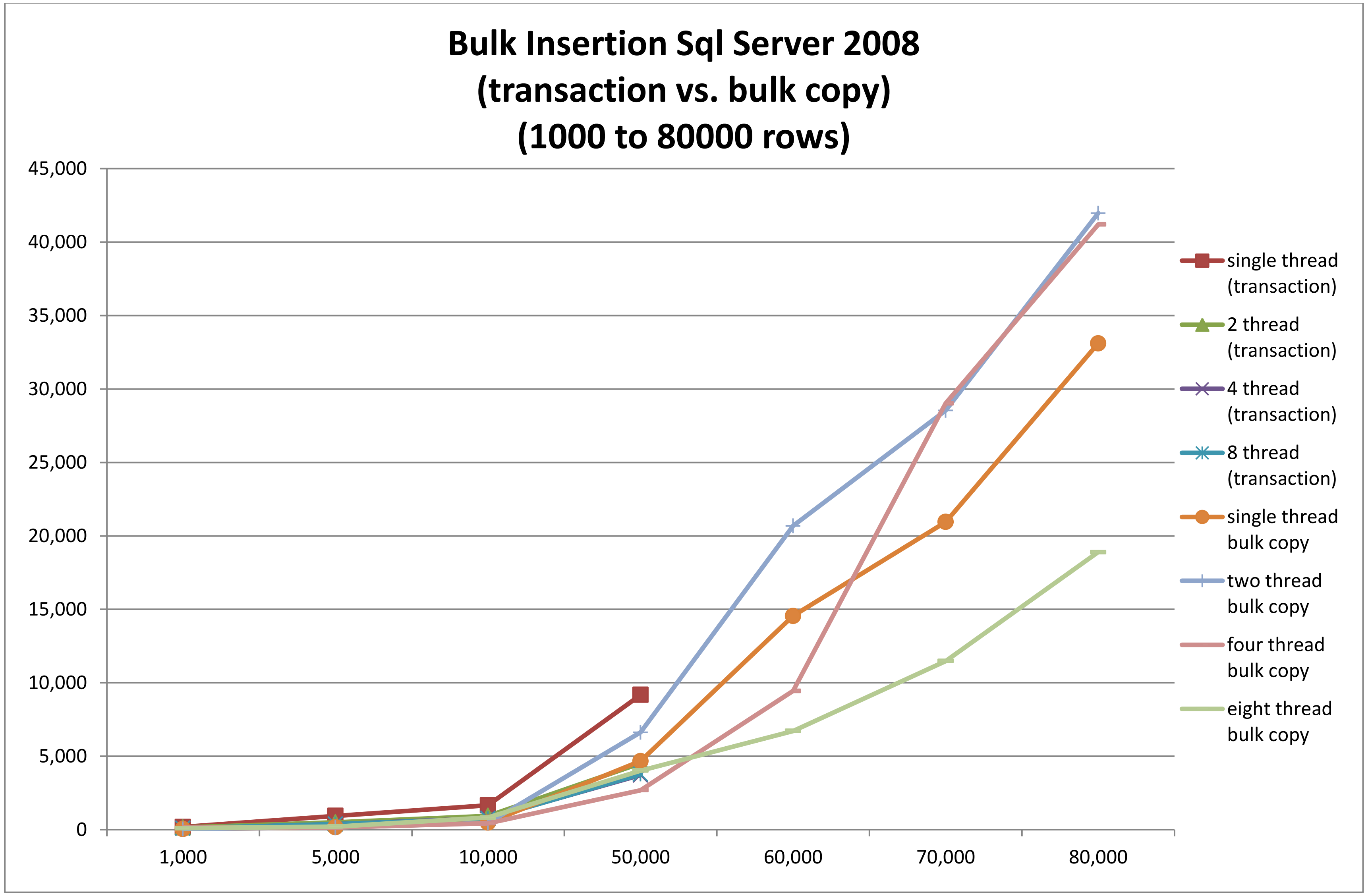


Figure 0-3: transaction vs. bulk copy for 1,000 to 80,000 rows

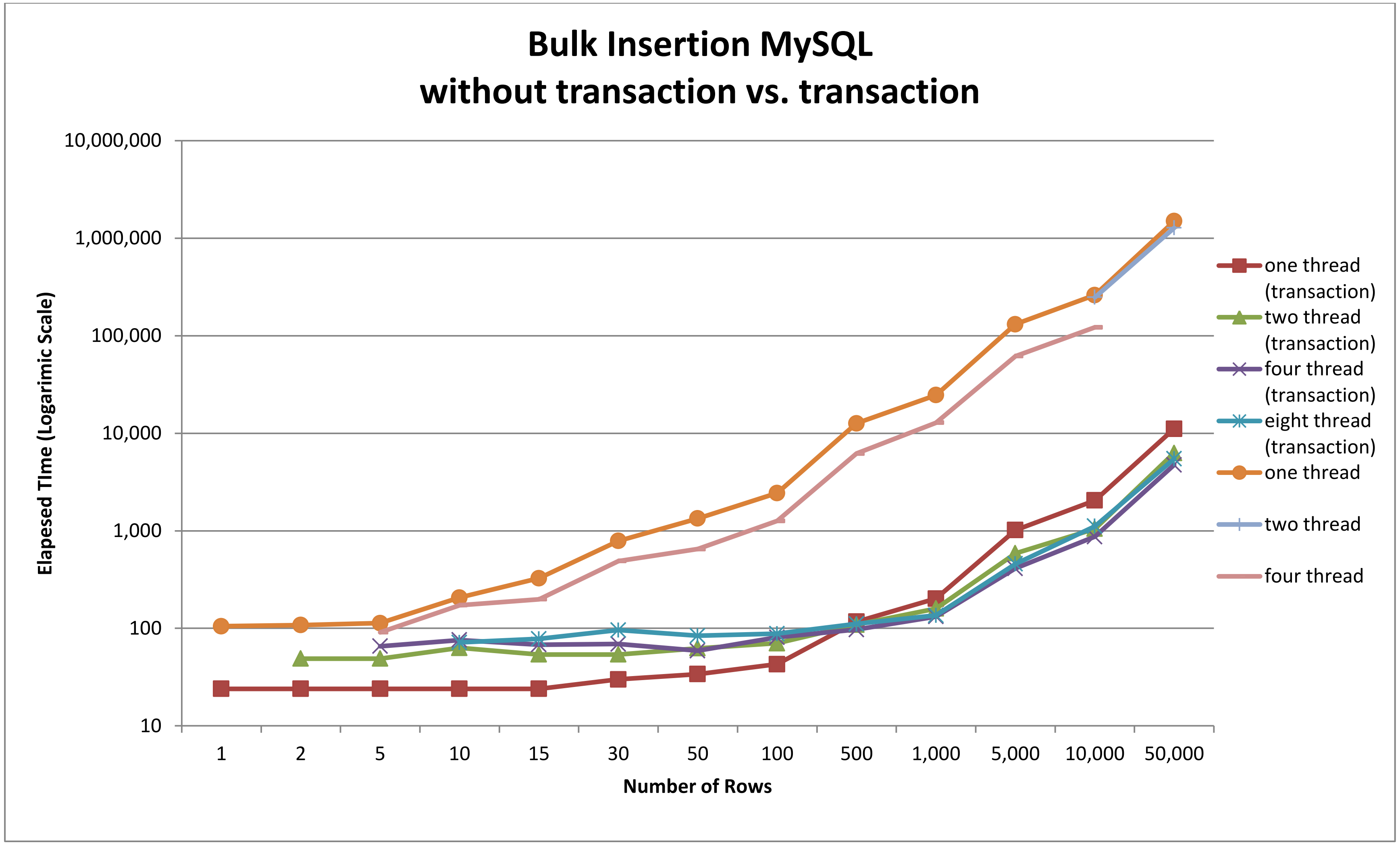


Figure 0-4: comparison between with and without transaction codes

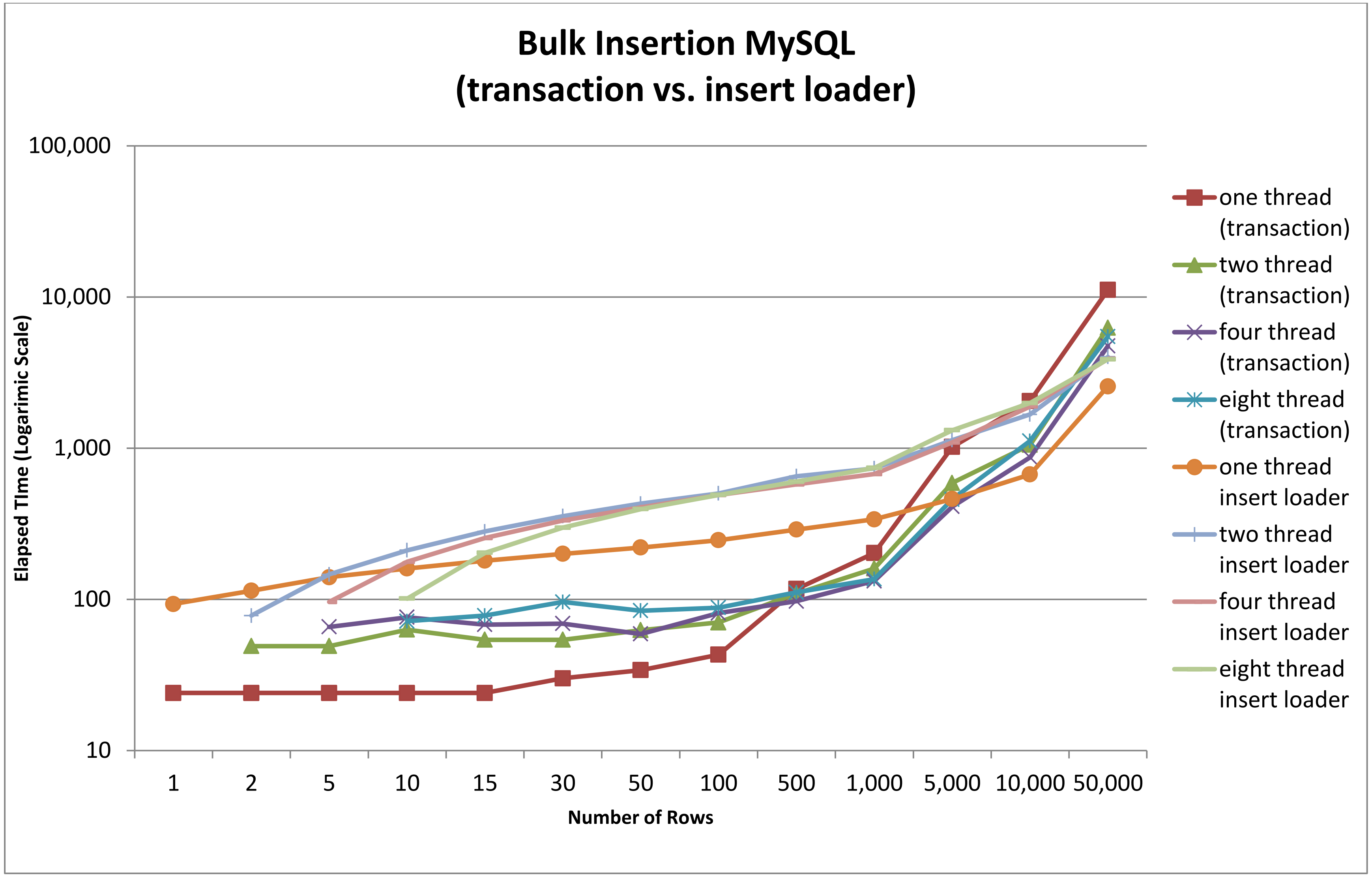


Figure 0-5: Comparison between transaction and insert loader

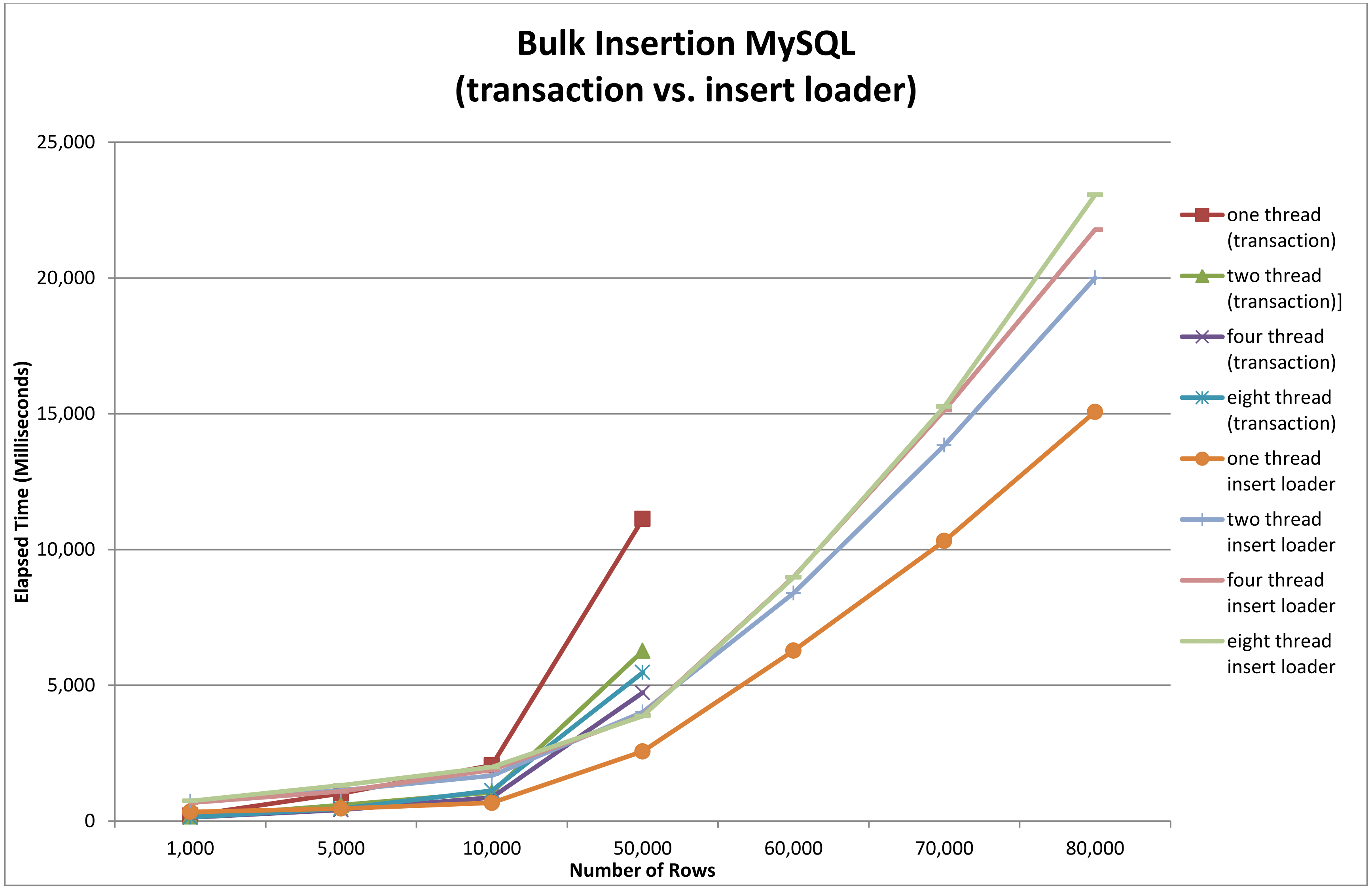


Figure 0-6: Insert Loader vs. Transaction at 1,000 to 80,000 rows