MASTER'S THESIS | LUND UNIVERSITY 2018

A Method for Performance Change Assessment Before a DBMS Upgrade

David Phung, Tobias Ronge

Department of Computer Science
Faculty of Engineering LTH

ISSN 1650-2884
LU-CS-EX 2018-43

A Method for Performance Change Assessment Before a DBMS Upgrade

David Phung
dat13tph@student.lu.se

Tobias Ronge
to5420ro-s@student.lu.se

November 20, 2018

Master's thesis work carried out at Consafe Logistics.

Supervisors: Per Andersson, per.andersson@cs.lth.se
Sverrir Valgeirsson, sverrir.valgeirsson@consafelogistics.com

Examiner: Flavius Gruian, flavius.gruian@cs.lth.se

Abstract

Different SQL implementations may differ in a large number of areas, including SQL syntax, output, memory usage and timing. Therefore, a strategy to evaluate an SQL implementation is needed, in order to allow a database manager to safely upgrade a DBMS or switch to a new system by migrating the database. In this master thesis, a method for measuring performance on relational databases is developed and presented.

As part of developing the method, a Windows application has been made in order to help database managers test and compare different versions. With the help of the application, performance differences (namely memory usage and response time) can be evaluated.

Keywords: MSc, SQL, SQL Server, MySQL, DBMS, database, performance, upgrade, migration, relational database, RDBMS

Acknowledgements

We would like to thank the following Consafe Logistics employees: our supervisor Sverrir Valgeirsson, Jens Persson, Tommi Arminen and Mikael Öwall.

We would also like to thank our supervisor at LTH, Per Andersson as well our examiner Flavius Gruian.

Contents

Acronyms and abbreviations

- API - Application Programming Interface - A way of allowing programmers to communicate with certain software.
- C# - An object-oriented programming language commonly used for developing Windows applications in Microsoft Visual Studio.
- DBMS - Database Management System - A system for creating and manipulating databases.
- GUI - Graphical User Interface - What the user sees on the screen and uses to interact with the application when running an application.
- JSON - JavaScript Object Notation - A quite common way of storing and describing data.
- MySQL - Common database management system.
- RDBMS - Relational Database Management System - A DBMS using the relational data model (see section 2.1)
- SQL - Structured Query Language - language used for communicating with databases.
- DBA - Database Administrator
- OLTP - Online Transaction Processing

Chapter 1
Introduction

In this chapter, an introduction to the thesis is presented. First, a problem background is described in section ”Background”. Then, the objective of the thesis is presented in section ”Objectives” and then, the scope of the thesis is presented in section ”Scope of thesis”. An introduction to Consafe Logistics - the company this thesis was made in cooperation with - is presented and finally, the full outline of this report is presented.

1.1 Background

Today, database management systems (shortened DBMS:s) play an important role in many organizations and businesses. They provide support and increase productivity in various processes, such as payment, production, decision making, online transaction etc. Some businesses even have a database at its heart and cannot function without a DBMS.

A DBMS is essentially a piece of software, and just like other software, it changes constantly. A typical release cycle is 18-36 months for major releases with minor updates in between. [16] New versions often introduce lots of new features, technologies, better security and performance. However, according to [15], there are a number of risks associated with an upgrade. Aside from purchasing the new version, whose price often increases 10-25%, the cost for planning, installing, testing and deploying must be taken into account as well. During an upgrade, the database will likely be unavailable, leading to disruption of business operations. Previously supported features might be removed leading to application errors. Supporting software products might not have immediate support for the new version. It is important that the risks and benefits of an upgrade are weighted before hand.

Aside from the upgrading process itself, one of the biggest concerns is the performance of the system. If the system does not behave as expected after the upgrade, it will have many impacts on other operations. It is natural to expect that a newer version will perform better. However, according to [15], when the SQL optimization techniques are changed, it is possible that the new version will generate SQL access paths that perform worse than before. Furthermore, during the risk assessment process, it is useful if the organization can gain estimate data about the system performance after the upgrade, as this can help them weighing between the risks and benefits to make better decisions. The ability to obtain data that reflect just the organization's system is important since that would give them the closest estimation. It is therefore useful for the organization if they can run tests of their own that would provide them with more accurate data to aid in the decision making process.

1.2 Objectives

In this thesis, we introduce a method to quantify the performance change of a system due to a DBMS upgrade. The idea was that before an organization performs a DBMS upgrade, they could apply this method in order to gain some estimated data about how the upgraded system would behave in comparison to the existing system. The results of applying the method would hopefully help them to decide whether the upgrade would be beneficial, how much performance would be gained or lost, or even to detect any potential problems within certain parts of the upgraded system.

The following questions will be answered: How can we be sure that the upgraded system would not perform worse than before? Can we estimate the performance change?

1.3 Scope of thesis

Performance measuring will for the most part follow the black-box approach. It means that we will look at the DBMS as a whole and from the outside. Certain measures from the inside might be taken in order to gain accurate results but we will not look into how different components of the DBMS work and interact with each other internally. This is due to the fact that a DBMS is very complex and we expect that our objectives could be achieved without delving too much into the inner working of the system.

Also, due to budget and time limit and the fact that we need to adequately validate our method, we have chosen to focus on only the following:

- Measurements will be done on response time and memory usage only. Due to lack of time, correctness of returned results from queries is not controlled. We will also not check consistency of data in the database after a workload is run against it.
- Microsoft's SQL Server (see [14] and section 1.5) will be used to validate our method. This is because SQL Server is widely used, has good support and documentation. It also has a free version for non-commercial uses. Moreover, SQL Server is also currently used at Consafe Logistics making it easier for them to help us and to validate our method.

- During the thesis, we will focus on relational DBMS. This comes as a direct consequence of our decision to use SQL Server since it is a RDBMS. Another reason is that the differences among different types of DBMS might make it infeasible to develop a method applicable for all. A selection is therefore necessary.

- We will focus only on the case of DBMS upgrade and not on the case of DBMS switching, that is: to move the database to a completely different DBMS. This is because switching requires us to migrate our test database to another DBMS which is a very difficult and time-consuming process (our test database is a large database provided by Consafe Logistics). We will still discuss the possibility of applying our method to the case of DBMS switching.

1.4 About Consafe Logistics

Consafe Logistics is a company selling and managing warehouse management systems. Its headquarters is located in Lund, Sweden. According to its website, the company operates in 30 countries and has approximately 350 employees [4].

As Consafe Logistics deals with warehouse management, databases play a very important role for them. They are used to keep track of every product and possible component a warehouse holds. If a database crashes, the consequences could be devastating. Because of this, the company is looking for a safe way to upgrade their current database systems, either to a new version of the currently used system or a completely different system.

Consafe Logistics has big expertise in back-end development and database management, yet lacks a method for evaluating database management systems after an upgrade has been performed. If an upgraded version of the DBMS requires more memory, for example, hardware might need to be upgraded. If the upgraded version runs too slow, it might be avoided.

1.5 Methodology

Initially, the company had an idea of what the method should be like but there were a lot of uncertainties around it, such as whether it was feasible or what the detailed steps would be. Therefore, we started the project by performing a preliminary study followed by the formulation of the method and its validation.

Preliminary study was done in beginning of the project and took a large portion of time. The goals were to gather theoretical and empirical data to prepare for the later phases. Theoretical data was gathered mainly through literature study. Most of these were about performance evaluation techniques, both theory and the actual evaluations that had been carried out. This part of the project gave us more understanding about the area, the studies that had been done and what we could learn from them. Aside from literature study, some information about the inner working of a modern DBMS, especially regarding the query optimizer, was obtained through a meeting with an expert from the company. Empirical data was gathered through testing, vendors' documentation and other sources such as web articles, forums Q&A websites. The most important goal was to determine the feasibility of the idea the company had proposed at the beginning. In order to obtain more reliable data, each finding in this phase was verified by empirical testing to see whether it actually worked. Different tools were also examined to see if they could be used to support the process. It is worth to note that we were only able to try out free tools.

After the preliminary study, we performed a brainstorming session in order to analyze the findings. The results showed that the initial idea was feasible and we started to formulate it formally based on the data we had gathered up until that point.

After having formulated the method, we applied it to the situation at the company as a case study. The company provided us with a test database and workload captured from their test system. During this, we had to tweak our tool a number of times as the actual workload from the company introduced a number of issues we had not thought of before.

1.6 Outline

After the introduction, the report will go on to the chapter "Theory", which will give a brief introduction to relational databases, the SQL standard and some information about query optimization (techniques used by a DBMS to make transactions go faster).

After this, chapter 3 will introduce related works in the area. Chapter 4 will present results of our preliminary study and the method formulation.

Next, chapter 5 will talk about how the method was validated and after this, chapter 6 will provide discussion on the results and talk about what remains to be done. This chapter also mentions how syntax and output differences between DBMS:s can be accounted for.

Chapter 7 provides a short conclusion.

Full instructions on how testing can be performed using the applications is found in the instructions document.

1.7 Contributions

The contribution of this thesis is a systematic formulation of a database performance evaluation method based on capture and replay. In addition, the report provides some insights into the state of database performance evaluation in general.

1.8 Work distribution

During the course of the thesis, much work came into experimenting with the DBMS:s (Microsoft SQL Server and MySQL) and ODBC. This was done by both of us.

Programming was done by both of us, with David responsible for the response time aspect and Tobias responsible for memory usage.

Both of us met with representatives of Consafe Logistics on regular meetings.

As for the report, the abstract, popular science conclusion, sections ”About Consafe Logistics”, ”Outline” and ”Measuring methods” and chapters ”Theory”, ”Evaluation” and ”Conclusion” was written by Tobias. The sections ”Background”, ”Objectives”, ”Scope of Thesis”, ”Methodology”, ”Method Formulation”, ”Capture” and ”Tool development” and chapter ”Related Work” was written by David. Chapter ”Validation” was written by both Tobias and David.

Chapter 2

Theory

In this chapter, some theory about relational databases and the SQL language is presented, giving the reader an overview of how SQL works. This will likely be needed in order to understand further theory.

2.1 An introduction to relational databases

Relational databases are databases relying on a relational model. That is: a database including different tables where data in a specific table corresponds (relates) to data in other tables. The usual programming language for relational databases is SQL (Structured Query Language). By forming commands using SQL, the data in a database can be affected according to the programmer's wish.

The data in a database is stored in different tables created by the database administrator alternatively database developer. Each table contains different columns, whereas each column has a specified data type, such as numbers, text or timestamps. Finally, the actual data is stored as rows (or entries) in the table. Data in different tables can be linked together with common values, usually called ”id:s”. "Linked together" in this sense means that queries can select information from different tables at the same time, where (for example) a common id is specified in order to link data from multiple rows from the different tables together. It is this kind of linking which is also known as relations.

An application that allows the creation of manipulation of databases is called a database management system, or DBMS. Some examples of common DBMS:s are Microsoft SQL Server [14], MySQL [17], PostgreSQL [21] and SQLite [25].

By using API:s, the use of a DBMS can be extended to websites, Java, mobile apps and standard Windows applications, to pick some examples.

There are a large number of implementations of SQL, which differ. SQL seems to be the by far most used language for relational databases: the 10 most used DBMS:s according to [5] implements SQL or SQL-like language.

In this thesis, only relation databases are regarded.

2.2 SQL standard

SQL has been adopted as a standard language for relational databases by ANSI and ISO/IEC [19]. It has been standardized in the documents both called "Database Language SQL" (see [9] and [10]). Unfortunately, the document is not available for free. It is therefore unclear to us what the standards cover and to what degree it is followed by the different implementations.

When data is selected from a database, a command known as a *query* is sent to the DBMS. The query returns the data, which is presented as a subtable of the original table in the DBMS. In the query, information about what table(s) to select from, what columns to select from and possible statements which need to be valid is included.

Not all SQL commands are queries. Other examples of SQL commands regard data insertion, data deletion and script execution. Another name for such scripts are ”stored procedures”.

2.3 Microsoft’s SQL Server

Microsoft’s SQL Server is one of the most common database management systems today, with the latest major release being SQL Server 2017. According to [5], SQL Server is the third most used DBMS in the world today.

We have almost exclusively focused on SQL Server during this thesis, together with some focus on MySQL. This is mainly because of the fact that Consafe Logistics currently uses SQL Server.

SQL Server turned out to differ quite a lot from other SQL DBMS:s, such as MySQL and PostgreSQL. These differences were found in the syntax and data types used and the system also has its own scripting language used as an extension to SQL (known as Transact-SQL [14], or T-SQL). T-SQL allows for variables and procedural programming, which is otherwise not part of the SQL standard. There is no reason to believe the differences in SQL syntax would make it harder to apply our method to other DBMS:s.

SQL Server is often distributed with an application known as SQL Server Management Studio (SSMS). It allows for creation and manipulation of databases. The other way of doing this is through SQL commands, which can be done with the help of formerly mentioned Transact-SQL and Microsoft's Visual Studio. The differences between SQL Server and other DBMS:s do not in any way mean that the method we develop can not be applied to other systems as well.

In this thesis, Microsoft's SQL Server 2012 and 2017 was used on Windows 10 environments.

2.4 MySQL

MySQL [17] is an open-source DBMS today developed by Oracle. Development of MySQL began in 1994 and the system was first released a year after. According to [5], MySQL is the second most used DBMS today, after Oracle.

In this thesis, MySQL was used in order to serve as an alternate system to SQL Server. This is partly due to its popularity but also the fact that MySQL is one of the candidates employees at Consafe Logistics can see themselves use in the future.

2.5 Query optimization

Query optimization are built-in methods used in many different DBMS:s. By letting the DBMS record information about how the database is used and then use these statistics to run queries in a more efficient way, performance can be enhanced.

In SQL Server, statistics objects are created in order to enable the query optimizer to "make the right plan choices based on estimated costs" [13]. Example of such statistics is how many times a certain value or row has been fetched.

Query optimization caused a potential problem to this project, as such optimizations was believed to affect the outcome of the applications. This is as the statistics might have changed over the course of the runs. This means that the preconditions for the DBMS changes, causing unwanted randomness in the measurements. To solve this problem, the statistics were cleared before each run using the command "DBCC FREEPROCCACHE" [14]. Unfortunately, not all statistics were possible to clear. In order to remove statistics for the indexes, the indexes had to be removed completely [14].

2.6 Performance measuring

Control and optimization of database systems require good performance models capturing dynamics during high loads [11]. Performance measuring can be done by sending data to a database according to a mathematical model. In this thesis, however, a more in-depth strategy has been taken with the aim of actually mimicking real communication, thus creating a realistic load. This is done by using SQL queries recorded from real database usage and replay them a large number of times in a test environment. See sections 4.1 and 4.4

for information about the method and an application allowing such queries to be re-played from a trace file.

Chapter 3
Related work

According to [29], there are four main techniques used to evaluate database systems:

- Queuing model: These models view the dynamic behavior of a DBMS as a queue model with jobs moving around demanding services from the resource stations. This technique is used to study system throughput, resource utilization and job response time. Due to the complexity of database systems, a queuing model can usually only describe parts of the behavior such as concurrency control, data allocation, locking mechanism etc.

- Cost model: Cost analysis can be used to obtain estimates about storage costs and query response time. However, according to the paper, cost models fails to account for dynamic behavior of database systems.

- Simulation modeling: The two above techniques rely on analytical models to evaluate performance which usually falls short due to the complexity of real life systems. Simulation modeling takes another approach in which it approximates the behavior of the real system and therefore produce better estimates.

- Benchmarking: This technique compares performance of two or more different DBMS:s by running the same workload against them. The results can then be used to compare the systems together.

Among the four techniques above, we were only able to find literatures related to the first and fourth techniques, namely queuing model and benchmarking. For cost modeling, no related studies were found, for simulation modeling, two were found. Therefore, in this section, we will present our findings of the queuing model, benchmark techniques. The simulation technique will be presented briefly. In addition, we will also present some studies done to evaluate the performance of different DBMS:s.

3.1 Queue modeling

Software in general can be viewed as a network of queues. The idea was originally proposed by Lazowska [12] and later developed into various software performance engineering methodologies. Since a DBMS is essentially a software system, this approach can be applied as well. The authors of[20] have done a survey on different queuing models. The paper divided these models into four categories based on the level of details at which transactions are modeled:

- Black-box: The database is represented as a single queuing service center. Each transaction class has an arrival rate and a service demand on the center. The internal design of the transaction is not represented in the models.
- Transaction processing model: The database is represented by the underlying hardware architecture. Each transaction class is defined by its service demand on components of the hardware architecture.
- Transaction size model: Each transaction class is described by number of data objects it accesses. These data objects can be rows, data pages or locks.
- Transaction phase model: Each transaction is represented by the number of phases it goes through, e.g. execution phase, data accessing phase.

Models in each of the above categories are suitable for different purposes, e.g. black-box is better for overall system performance and tractability while transaction size, transaction phase are better for statement and index performance evaluation. The paper also evaluated a number of assumptions that most models made and how they affected the results. An example is that all studies except one assumed data is accessed uniformly. By examining the workloads from the production databases of 10 of the world's largest corporations, the authors pointed out that this was not the case for realistic workload and caused the performance results to appear better than they actually were.

A concrete example of a queuing model is in the paper [11] The authors performed some preliminary experiments on MySQL to discover whether the system could be modeled as a M/M/1 system. A M/M/1 system is a system in which there is one single server, incoming jobs follows the Poisson process and service times have an exponential distribution. By taking into account load dependency in service time, they were able to modify the system so that it would correspond with the experiment results in some cases. The same authors also performed another study on modeling a database server with write-heavy workload [6]. Such systems are more complicated due to the buffering of disk operations when data is too large to be stored in main memory. A queue model must capture not only the CPU dynamics but even the hard drive dynamics. The paper proposed a model consisted of a network delay, job queue and a dirty page buffer. After passing the network delay, requests enter the job queue and dirty pages is placed in the dirty page buffer. In order to validate the model, the authors developed a simulation program of the model and compared the response time distribution with the results from a testbed experiment. The results from the simulation fitted accurately with the testbed experiment. Queuing model studies often focus on a system with a certain characteristics and then model one or two aspects, such as load balancing, page caching, concurrency control protocol, lock acquisition etc. Assumptions are usually made on other aspects which lead to uncertainty in the results, as mentioned above. The queue modeling technique is not widely used in industry. The paper [20] suggests it could be due to the lack of tools and empirical evaluation on their applicability on real situations.

3.2 Simulation modeling

The simulation approach approximates a real system with a model and then use that model to gather estimates about the true performance characteristics of the system. In the study [2], the authors used a simulation model to show that the interpretation overhead done by the data manipulation routines can be a significant factor affecting performance. Another study [8] used a simulation model to show that security constraints can be incorporated into a real-time database system in such a way that the performance is not significantly affected.

3.3 Benchmark

Benchmarking is often used to assess the relative performance of different DBMS:s. It is the act of running similar workloads against multiple systems while measuring the performance. Since the workloads are the same, the results give an estimate about how different database systems fare against each other. Before performing a benchmark, a system configuration must be defined. The system configuration consists of the DBMS and the environment it runs on. Notable factors of the environment are hardware and operative system [29].

In order for benchmark results to be comparable, certain measures need to be taken. Running different workloads against different systems will not yield anything meaningful while it is difficult to design an workload representative for all systems. This has led to the forming of the Transaction Processing Council (TPC), a non-profit corporation founded to define industry standard benchmarks as well as to review and monitor those benchmarks [28]. The members of the council are among the largest on the market, such as Oracle, Microsoft, IBM etc. TPC has developed a number of benchmarks suitable for different types of computing environments. The TPC-C benchmark, for example, is suitable for Online Transaction Processing (OLTP) systems. It consists of a complex database with nine types of tables and a workload that simulates multiple real users making transactions towards the database. These transactions include order acceptance, delivery, payment recording etc. It simulates a company owning warehouses on different districts [27]. This benchmark is therefore useful for systems with similar characteristics. Other examples include TPC-H for decision support systems, TPCx-V for databases running in virtual machines, etc. On TPC's website, users can see benchmark results published by different DBMS vendors. It is worth to note that a higher benchmark value does not necessarily mean an overall better DBMS. It only means that the published DBMS, running on the published system configuration yielded that particular performance value. We cannot draw any certain conclusion when that DBMS is run on another system configuration [26].

The findings regarding benchmarking raise an interesting questions. Can we look at the benchmark values of two versions of SQL Server on the TPC's website and determine the performance change after the upgrade? There is certainly a correlation between those values and the unknown performance change we are looking for. However, without knowing this exact correlation, it is still difficult to gain a good estimate. Our thought process regarding this matter has been that performance depends on four factors: the DBMS itself, system configurations (operative system and hardware), the database and the workload. When one of these factors varies, the performance changes. When interpreting the benchmark results from the TPC's website, users need to compare their specific system with the benchmark in terms of these factors. How similar are the database, the workload, the system configurations of the organization in comparison to the ones in the benchmark? This thought process is of course unproven, but it can be reinforced by a statement in the TPC-C's specification[27]: "Despite the fact that this benchmark offers a rich environment that emulates many OLTP applications, this benchmark does not reflect the entire range of OLTP requirements. In addition, the extent to which a customer can achieve the results reported by a vendor is highly dependent on how closely TPC-C approximates the customer application. The relative performance of systems derived from this benchmark does not necessarily hold for other workloads or environments. Extrapolations to any other environment are not recommended." A similar statement can be found in the specifications of the other TPC-benchmarks.

A study [24] and a PhD thesis [30] introduced an approach called application-specific benchmark. This approach incorporates characteristics of the application of interest in the benchmarking process, thus producing results that closely reflect the behavior of the application. It is worth to note that these studies were about software benchmarking in general. They did not specifically target DBMS:s.

3.4 Evaluation studies

In 2011, a master thesis was done to compare the performance of SQL Server and Oracle in terms of response time and memory usage [18]. A database with four tables was used. Four types of SQL statements were executed against the database multiple times, after each time the number of rows in the database was increased in order to monitor the performance change on growing database. It was not clear how measurements were obtained. The result was that Oracle required times more memory than SQL Server and both had about the same response time.

In 2012, a study was done to compare the performance of five DBMS:s, namely SQL Server 2008, Oracle 11g, IBM DB2, MySQL 5.5 and MS Access 2010[3]. A tool was developed to fill the database tables with 1 000 000 rows of data and then execute the queries. The database was modeled to simulate a business retail system with a front end to create invoices, receipts, orders and a back end to manage item stocks. A set of 10 queries with different levels of complexity was executed against the database. It was unclear how many times each query was run. The performance aspects that were measured were response time, CPU utilization, memory utilization, virtual memory utilization and number of threads used. These values were obtained by using Windows' task manager. The result was that there was no clear winner, different DBMS:s were strong at different aspects.

In 2015, a study was done in order to compare the performance of MySQL 5.6.17 and SQL Server 2008 in terms of response time[23]. A database was deployed on MySQL and then migrated to SQL Server. Then a number of SELECT, UPDATE and DELETE statements were executed against each of them. Each statement was executed five times. The paper gives some information about the structure but not the exact statements. The results were that SQL Server performs better on the statements that were used .

All of the above evaluation work followed the benchmark approach in which they developed a database to model a certain type of system and then executed a number of statements against the database while taking measurements. Another common characteristic is that they all performed the testing on a personal computer in lab-controlled environment. There are therefore uncertainties in interpreting their results. Can the results obtained by these studies be translated into other situations, with different system configurations, a different database and realistic workloads? Here we applied the same reasoning we presented in the end of section 3.3 about benchmarking.

Chapter 4
Results

In this section, we first introduce the method, then we present our findings regarding capturing of an workload as well as how to measure the response time and memory usage.

4.1 Method formulation

The principle of the method is similar to that of the benchmark technique in which we run the same workload against two different DBMS:s. The difference is that instead of using synthetic workload and database, we will use the actual ones. In another word, we will use the database that the organization is using and the workload that their current system has to handle. This way the characteristics of the specific system are automatically included.

First, a test environment must be set up. The test environment should consist of the old and new DBMS version, system configurations (hardware and operative system), and a test database. Then a workload is captured and replayed against the old and new DBMS versions while measurements are obtained. It is not necessary to do these replays in parallel. Finally, the measurements are reported.

We think that the capturing should be done in the production environment as this would capture the real workload that the system currently has to handle. This, however, can affect the server and therewith the organization's current operations. Careful considerations must be made before performing live capturing. Therefore we recommend that the process should first start with a planning step where the risk of production environment capturing is determined. The affect of capturing could be assessed first in a test environment.

During the replaying step, the captured workload is run by a tool against the two DBMS:s. The tool should run the workload as accurately as possible. By accurately we refer to the order of the requests, the number of connection sessions and the delay between requests.

The delay can be obtained from the timestamps of the requests when we captured the workload. The connection session information will be mentioned later.

Measuring is done at the same time as replaying. Different options will be presented in 4.3. It is possible to incorporate the measuring task into the replay tool.

Figure 4.1 illustrates the process of capturing and replaying. The checkpoint is a backup of the production server right before the workload is captured, so that the test server can be restored to this point before the replaying is done.

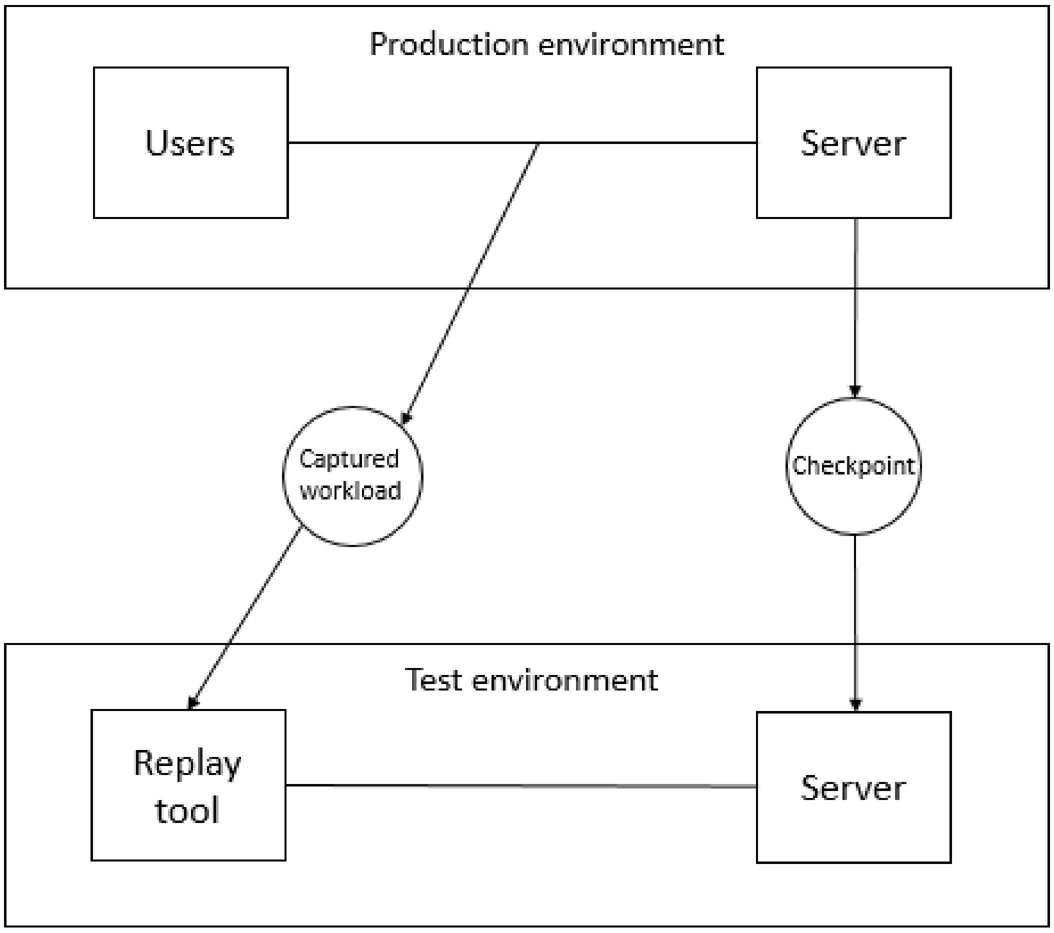

Figure 4.1: The process when capturing is done on production database.

4.2 Capture

In this section, we present our findings regarding the uncertainties we had in the beginning of the project: whether it is possible to capture a workload. Two approaches were studied, packet sniffing and DBMS features.

4.2.1 Packet sniffing

Packet sniffing is the act of listening to traffic flowing on a network. On a broadcasting network, a machine can be set to promiscuous mode which causes it to silently accept all the packets on the network, even if they are not intended for that machine [1]. This technique can be used to for the capturing task. The advantage of packet sniffing is that it can be run on a separate machine and therefore does not use the server's system resources. This technique was studied briefly in the project but we did not go into depth with it. Instead, we focused on another technique which we will present in the next section.

4.2.2 DBMS features

Instead of capturing SQL data independently from the DBMS as above, the method we are about to present here rely on features supported by the vendors in order to record activities happening from the inside the database engine. DBMS:s often provide advanced users, such as the database administrator and developers, with functionalities to look inside the system for tuning and debugging. During our study, we found out that these could be used for the purpose of our capturing workloads as well.

In SQL Server, the dedicated feature used for viewing actions happening inside the DBMS at real time is called SQL Trace [14]. A trace is a collection of events and data returned by the database engine. When started, a trace runs in real time among other processes in the engine and stores recorded information in a table. All events have a number of attributes represented by data columns in this table. There are about 200 types of events and 60 types of attributes. At the time of creation, users can specify what events and attributes they would like to be included in the trace. Among the attributes, server process ID (SPID) can be used to distinguish between different connection sessions. For the purpose of replaying, we do not need all of these event and attributes. In addition, capturing fewer data also reduces the performance impact on the server.

SQL Server provides two methods to work with SQL Trace: through system stored procedures or the tool SQL Server Profiler. There are four system stored procedures to work with traces: *sp_trace_create*, *sp_trace_setevent*, *sp_trace_setfilter*, *sp_trace_setstatus* [14]. These can be called from Transact SQL script and sent from an application, suitable for writing custom tools specific to certain needs. For most other general purposes, the tool provided by Microsoft, SQL Server Profiler, is sufficient [14]. It provides same results as the four mentioned procedures with the addition of a GUI. Once done, a trace can be saved as a file format native to SQL Server or as an XML file. It is also worth to mention that the feature SQL Trace is in maintenance mode and may be removed in the future versions of SQL Server. Instead it is going to be replaced by another feature called Extended Events, a lightweight performance monitoring system, although it still includes all the functionalities of SQL Trace [14]. In this thesis, we have chosen to focus mainly on SQL Trace instead of Extended Events because we are examining the case of upgrading from an old version to a new version. Capturing must therefore be done on an old version of the DBMS which does not have the new Extended Events feature.

4.3 Measuring methods

4.3.1 Client-side

The client side refers to what is run at the computer running the application.

If the DBMS runs on the same computer as the application, timing and memory can be measured client-side. This might be the case in a test environment. Memory is measured by looking at the memory usage of the corresponding Windows process of the DBMS, while timing can be measured either by profiling (letting the DBMS provide timing statistics) or by letting the application measure it. If the latter is chosen, the ”stopwatch”-class of the .NET framework is used by the application.

4.3.2 Server-side

Server-side refers to the DBMS side of the system, while the DBMS is located on a remote computer.

Since the application runs on client-side only, there is currently no way of measuring memory server-side built-in to the application. This is as there is no functionality in SQL Server allowing memory allocation to be extracted while there is also no functionality in the application allowing the user to find memory of processes on the remote computer.

The latter functionality could be added in the future, for example by the creation of a server-side application which would read the memory of the DBMS process on the remote computer and then send it on the network to the client.

Timing analysis can hopefully be done by the tool using SQL Server Profiler as ODBC allows for remote connections. It has, however, not been tested. The network communication is in this case handled by ODBC and timing data is transferred from the SQL Profiler to the application using an SQL request.

4.3.3 Verification of performance measuring methods

During the course of the thesis, an application was programmed which measured memory and timing performance of databases. Timing values was extracted from the DBMS while memory was measured using methods built into the .NET framework. Currently, This application was proved compatible with SQL Server as well as MySQL and possibly compatible with other common DBMS as well.

The application measured timing according to a method selected by the user: either using application-side measuring or by using a built-in profiler. The second option is likely to be more accurate, as it would extract actual CPU time rather than the total timespan.

In order to extract memory usage, process monitoring was used. By calling methods in the .NET framework, processes could be listed and selected and measurements could be extracted. By letting a separate thread take measurements on memory while the application executed SQL commands, average and maximum memory values could be found.

As the methods were proven successful, both of these techniques for measuring performance were implemented in the final application as well.

4.4 Tool development

During the project, we developed a tool to automate the process of sending requests to the DBMS while taking measurements of the response time and memory usage. Initially we wanted to write the tool in Java, but since the company had better expertise in C#, we decided to use C# instead. This way they could better help us during the process if we encountered programming problems. There are four main tasks performed by the tool: loading of captured workload, replaying, measuring and reporting. The total lines of code for the tool is 1041, obtained by the code metric function in Visual Studio 2017.

Loading of captured workload

Before, the captured workload can be replayed, it must first be loaded together from the file. How this is done depends on the format the trace file was stored in. In our implementation, we used the format native to SQL Server instead of XML. The reason was because SQL Server supplies a function to allow easier reading of this file, the function returns a table that can be used directly by the application. If we had used XML file, we would have had to concern ourselves with the matter of parsing the file. The supplied load function is sys.fn_trace_gettable and can be used as part of a select statement. The following is the SQL statement that we used to query the content of the trace file. It returns the events and attributes necessary for our replay strategy.

```
SELECT EventClass, TextData, EventSequence,
StartTime, EndTime,  SPID, DatabaseName,
ApplicationName, HostName
FROM sys.fn_trace_gettable(pathToTraceFile)
WHERE EventClass = 11 OR EventClass = 13
   OR EventClass = 14 OR EventClass = 15
   OR EventClass = 17
```

As can be seen on the select part of the statement, the event type, text data, start time, end time, server process id (SPID), database name, application name and host name of each event were read from the trace file. The event classes 11, 13, 14, 15 and 17 correspond to *SQL:BatchCompleted*, *RPC:Completed*, *Audit Login*, *Audit Logout* and *Existing Connection* events. This selection is not necessary if the trace file only contains these data. However, we did not want to impose such requirements on the capturing process.

Replaying

Before each time the workload is run against the database, we performed a backup restore to restore the database to the same state. This requires that a backup must first be created and stored somewhere. Our tool will not do this, instead, it must be done manually. The procedure to create and revert a backup is fairly simple is is described here [14]. The tool requires location of the backup in order to work. Through practical testing, we found that restoring does not work if there are existing connections to the database. All connections to the database must be closed first. The task of killing existing connections is done automatically by our tool.

Each event type is represented by a separate class. Each class contains its own logic to handle the execution of the corresponding event type. The Login class, for example, corresponds to the *Audit Login* event type and will create and open a new connection object when executed. The connection objects are of type *OdbcConnection* and contains necessary functions to execute SQL batches or stored procedures against the DBMS.

After the workload is loaded into a table, it is turned into a list of event objects and stored in a *ExecutionContext* object. The *ExecutionContext* class contains common data needed by each event object in order to execute itself. Aside from the the event list, another important data stored in the context object is the map of connection objects and their corresponding server process ID (SPID). Each connection to the server is given a separate SPID even if they come from the same application. This map is therefore used to maintain the same number of connections to the server as when the workload was captured. Each time an event is about to execute itself, it checks to see if its SPID is in the map. If yes, it uses the corresponding connection, if not, it opens a new connection. The event list has the same order as the loaded table and is executed in order by only one single thread. This preserves the chronological order of the workload. Another option could be to use one separate thread for each connection and run them concurrently. If the work done by each connection is independent from each other, this should not cause any errors.

During testing, we discovered a number of difficulties regarding replaying of the workload. Some of these problems were solved by introducing a preprocessing phase. The preprocessing is run after the trace file is loaded and before it is replayed. The following presents the problems we encountered and how we chose to solve them.

- Auto-generated stored procedures: We found that in the trace file, the *Audit Logout* was always preceded by a *exec_reset_connection* stored procedure. Through testing, we were able to determine that the procedure was auto-generated, it was unclear whether by the DBMS or by ODBC. Our solution was to ignore all of these event in the preprocessing phase.

- Related events: Some of the stored procedure events have output variables that store values returned by the DBMS. These values can later be used in other stored procedures. Similar to above, implementing this would require a parser to identify the output variables and store their returned values. Since we did not have time to implement a parser, we had decided to ignore these events from the trace file in the preprocessing phase. The amount of such events was very small compared to the rest of the workload so the impact should be minimal.

Measuring

Response time measuring is done by using a class in C# called *StopWatch*. The *StopWatch* object is stored in the *ExecutionContext*. Each event class starts the stopwatch right before sending a request to the DBMS and pauses it right after a response has been received. This is to reduce the inaccuracy of the measurements due to execution overhead. Memory measuring is done by monitor the process SQL Server is run on. A separate thread is dedicated for this. It checks the memory usage of the process every 200 ms and store the results in an array. After the replay is finished, the average of array elements together with the min and max values are reported.

Reporting

In order to show the progress in real time, we have decided to implement a small GUI with two charts for response time and memory usage. Each chart shows measurement values obtained so far and the current average. The GUI also provides additional data such as standard error and 95% confidence interval. The goal of this GUI is to help detecting problems with distribution of measurements in real-time as well as to decide when to stop the replay. At the end of the process, all values are written to a file in the .cvs format for storage, final analyses and other reporting activities. This part of the program used two external libraries. LiveChart was used for drawing charts and MathNet was used to obtain the t-value used for confidence interval calculation.

Chapter 5
Validation

In order to validate the method, we needed a database and a workload. These were provided by Consafe Logistics. First, a test database was sent to us from Consafe Logistics and imported into our SQL Server instances. Then, a trace file recorded by Consafe Logistics was sent to us. With the help of the trace files, our application (see 4.4) was run with the test database.

The testing environment was a personal computer Acer Aspire V5-552, running Windows 8.1 Basic Edition. The CPU type was AMD A10-5757M APU 2.5GHz with a total number of 4 cores. The workload was run 30 times against the SQL Server 2012 Express Edition and SQL Server 2017 Developer Edition. After each run, the total response time, average memory, min memory and max memory were stored in a file.

5.1 Response time

Figures 5.1 and 5.2 show the response time of the 30 runs for the 2012 respective 2017 versions. The gray lines represent the average values. For the 2012 version, this value is 20487 ms while for the 2017 version, it is 19217 ms. Table 5.1 presents the benchmark values obtained from the TPC-website. Initially, we searched for results of the TPC-C and TPC-H benchmarks. However, there was no TPC-C result for either the 2012 or 2017 version of SQL Server. Therefore only the TPC-H benchmark results are presented. QphH (the last column) is the metric of the benchmark, called the composite query-per-hour performance metric.

Although the differences in system configuration make it difficult to obtain a fair comparison, we can still observe from the table that there is a very large difference between the 2012 and 2017 versions. Our results, however, showed a very small difference. As we looked closer into the workloads from Consafe Logistics and the TPC-H benchmark, we discovered that the TPC-H queries were more complex. The database used by the benchmark is also much larger than the database Consafe Logistics gave us. This led us to think that the TPC-H benchmark comes closer to the top processing capability of the 2012 version, causing it to perform worse. The Consafe Logistics workload is lighter and can be handled with approximately the same ease by the two versions. This, of course, is a very simple explanation. An in-depth study of the differences between the two workloads and the databases is outside the scope of this thesis.

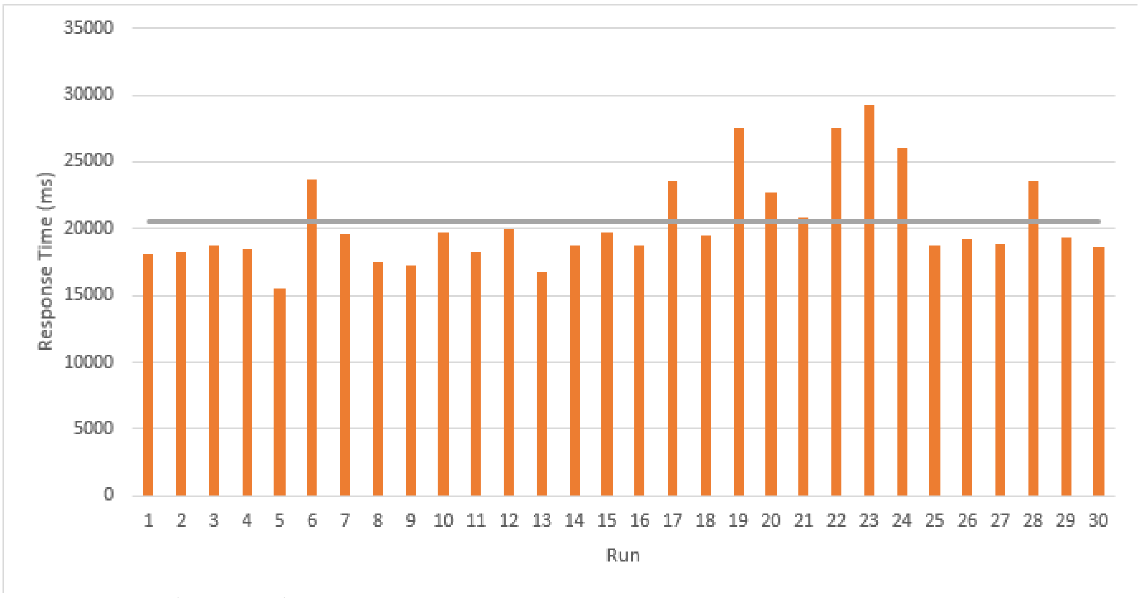

Figure 5.1: SQL Server 2012: The total response time values over 30 runs. The gray line is the average.

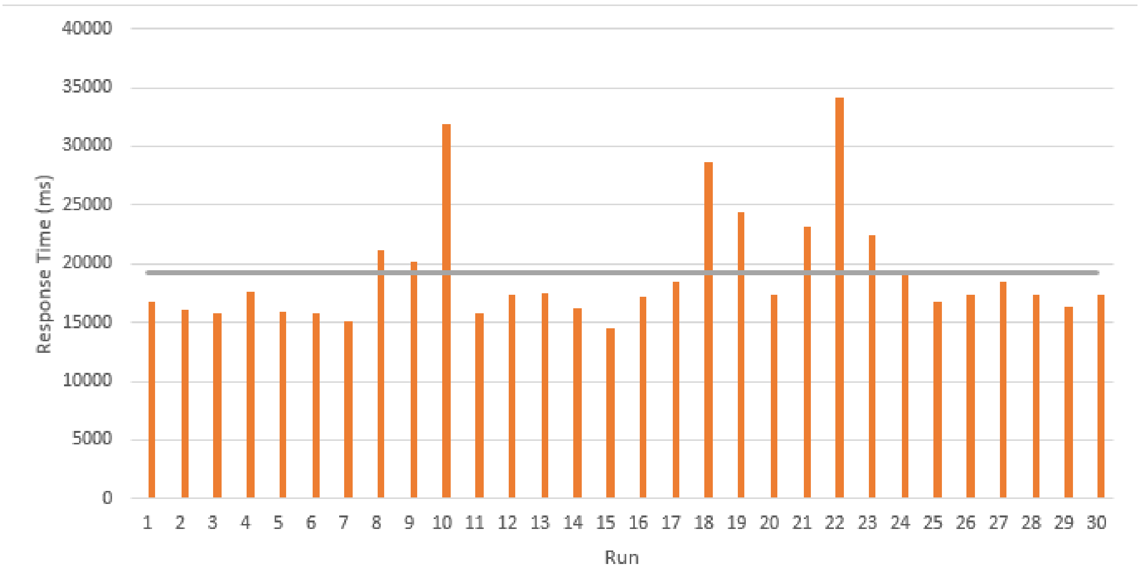

Figure 5.2: SQL Server 2017: The total response time values over 30 runs. The gray line is the average.

Database Software	**System**	**Operating System**	**CPU Type**	**#CPUs**	**#Cores**	**#Threads**	**QphH**
Microsoft SQL Server 2017 Enterprise Edition	HP Proliant DL580 Gen10	Microsoft Windows Server 2016 Standard Edition	Intel Xeon Platinum 8180M 2.50GHz	4	112	224	1479749
Microsoft SQL Server 2017 Enterprise Edition	Lenovo ThinkSystem SR950	Microsoft Windows Server 2016 Standard Edition	Intel Xeon Platinum 8180M 2.50GHz	4	112	224	1336110
Microsoft SQL Server 2012 Enterprise Edition	HP ProLiant DL980 G7	Microsoft Windows Server 2012 Standard Edition	Intel Xeon E7-4870 - 2.40 GHz	8	80	160	158108.3

Table 5.1: TPC-H benchmark results with scale factor 10000.

5.2 Average memory

Figures 5.3 and 5.4 show the average memory of the 30 runs for the two versions. Again, the gray lines represent the final averages. We have not found any benchmarks regarding memory usage.

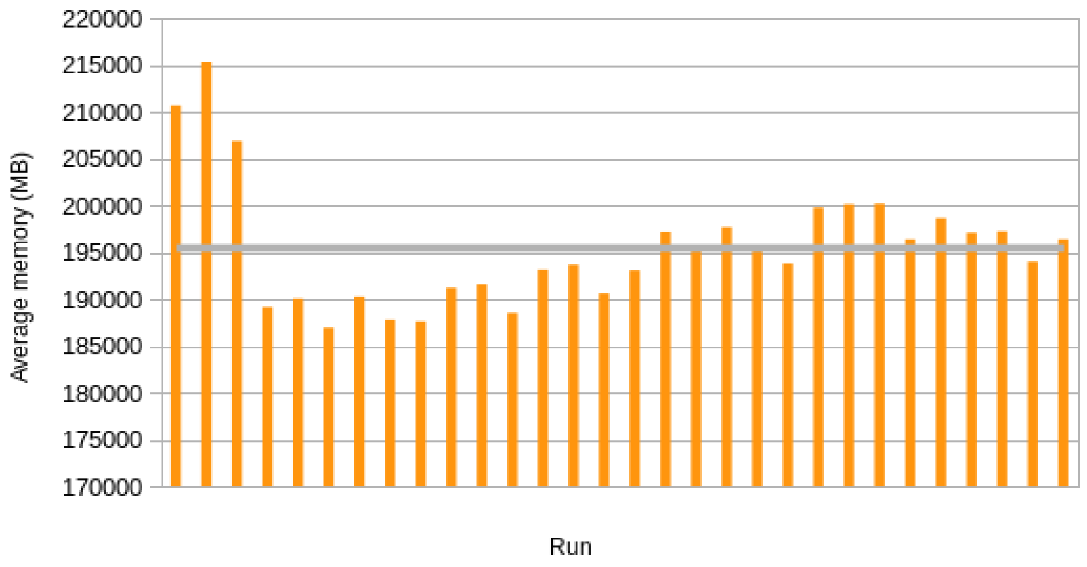

Figure 5.3: SQL Server 2012: The average memory values over 30 runs. The gray line is the final average.

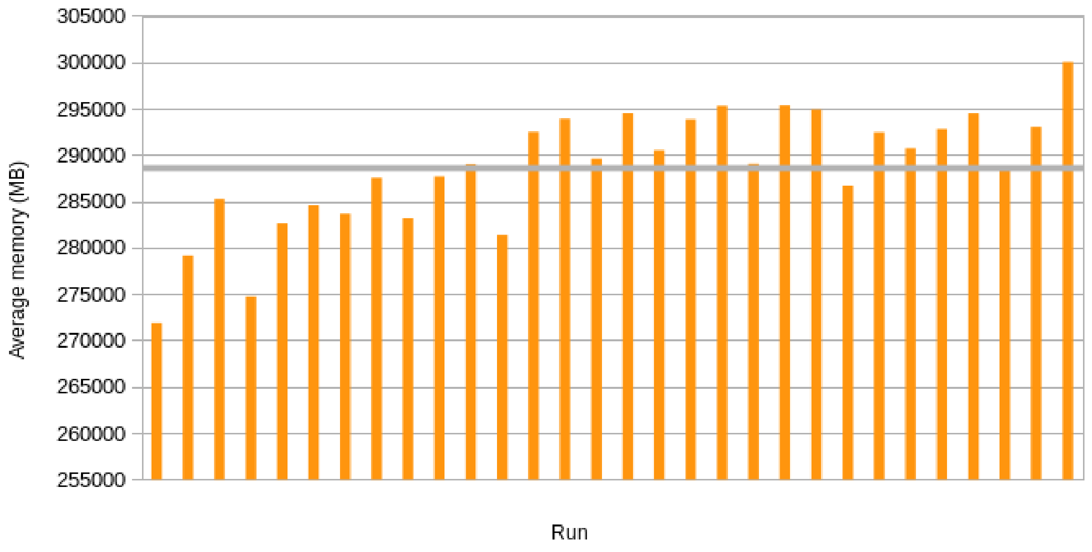

Figure 5.4: SQL Server 2017: The average memory values over 30 runs. The gray line is the final average.

Chapter 6
Evaluation

In this chapter, an evaluation of the resulting method is made. It will be explained what the uses are of the method, and its advantages and drawbacks.

It is worth mentioning that performance is affected not only by the DBMS but also by hardware. For applications including databases too large to store in main memory, hard drive dynamics will influence performance in high loads. [7] Further, operations can be buffered at the server in order to improve efficiency [7] (see [22] for more information on network file system buffering). Because of this, it should be concluded that the environment which the testing is performed on should be the same environment (or a very similar one) as the one actually used in reality. The main reason for not using the actual system already in place would be that it is currently in use.

Also noteworthy is that large fluctuations in timing may occur, as the DBMS needs to deal with buffering and caching (see [7]). It is therefore particularly important that the test runs a large number of times, in order to even out the results and obtain the true maximum values. An approximation of the number of times needed is hard to know and has not been obtained by us.

The project only focused on SQL Server. The applicability on other DBMS:s is not yet clear. This means that the method currently cannot be used for database switching, such as when migrating a database from SQL Server to MySQL, or upgrading a completely other DBMS such as MySQL. Future work can be done here, which will be discussed more in chapter 7.

Problems might occur when letting the application parse the trace files. If complications happens, human intervention might be required. Should exactly everything in the trace file be replayed?

Another problem which might show up when switcing database vendor is that queries will need to be changed in order to match the target DBMS:s SQL syntax. After experience from working with this thesis, it has turned out that this is impossible to automate completely unless every single possible DBMS is studied first. This conclusion was drawn after finding out about the differences between common DBMS:s. If such a study indeed was made, it might be possible to achieve such an application, but it would require a lot of work and would be far outside the scope of this thesis. It should be noted that changing queries in trace files is currently not supported by our tool.

Chapter 7

Conclusion

The performance measuring application was successful in measuring timing and memory usage for SQL Server, thus allowing for the completion of our method of measuring and comparing performance of different DBMS versions. As such, it is believed that this application could be used on the market or in research. An advantage to this method in comparison to, for example, the method of using a proxy which mirrors SQL commands to a test server is that the commands, when they have been recorded, can be replayed a large amount of times and without unnecessary delay (that is: as fast as the DBMS can handle them).

It is recommended that these applications are to be further developed, for example in order to be compatible with other DBMS:s. This should be possible to achieve, although it is hard to safely say this as more information about how the other DBMS:s work is needed. See the next section.

7.1 Future work

Future work remains mostly in further development of the test application. This will be explained in detail in section 7.1. The issue of finding logical differences between DBMS vendors will also be discussed in section 7.2. Finally, different approaches for evaluating database management systems might be attempted as well, as will be described in the last section of this chapter.

7.1.1 Application

Feasibility study on other DBMS

One interesting question worth studying is whether the application used can be extended to be compatible with other database systems, such as MySQL, PostgreSQL and DB2. While this would not solve the issue occurring when switching *between* different systems, it would still allow for safer upgrades of the compatible systems.

In order to find out more about this topic, documentation of different DBMS:s needs to be studied. During the course of this thesis, MySQL's tracer was briefly studied in order to let a test application extract timing values. As such, it is clear that such a tracer exists for MySQL and that it includes the different queries that have been executed on the system. In conclusion, it is already clear that our tool could be extended to be compatible with MySQL.

Comparing outputs

Yet an other way of further developing the application is to allow for checking of differences in output. That is, when the database has been upgraded, the application would check if the output of selected queries is still the same as before. This is quite an important task and could probably be implemented without too much work being needed.

7.1.2 Syntax differences

Syntax differences refers here to the differences in SQL dialects, such as names of data types and stored procedures. It appeared to us that it is impossible to automate the finding of such differences with an application. Instead, the database manager would be required to manually study the new DBMS and compare it with the old one.

7.1.3 Documentation

Finally, a safe (but time consuming) way of evaluating syntax and output differences between completely different DBMS:s (released from different vendors) is simply to make a grand evaluation of all common DBMS:s while documenting the results. For every method available to one ore more DBMS:s, an evaluation report could tell whether the specific DBMS differs from the SQL standard (or, if not included in the standard, the most common way) and if so, how. This could all be concluded in a number of tables, allowing a developer to quickly see if the new system differs from the former one and on what aspects.

A problem with this solution is that the different vendors frequently release new versions of their systems, which would mean that the report would need to be updated very frequently. Still, it might serve as a quick guide to allow the database manager to see where main differences lies, even if the report is not all up to date.

Bibliography

[1] Sabeel Ansari, SG Rajeev, and HS Chandrashekar. Packet sniffing: a brief introduction. *IEEE potentials*, 21(5):17–19, 2002.

[2] AJ Baroody and David J DeWitt. The impact of run-time schema interpretation in a network data model dbms. *IEEE Transactions on Software Engineering*, (2):123–136, 1982.

[3] Youssef Bassil. A comparative study on the performance of the top dbms systems. *arXiv preprint arXiv:1205.2889*, 2012.

[4] Consafe Logistics. About us. `https://www.consafelogistics.com/about-us/`.

[5] DB-Engines. Db-engines ranking - popularity ranking of database management systems. `https://db-engines.com/en/ranking`.

[6] Manfred Dellkrantz, Maria Kihl, and Anders Robertsson. Performance modeling and analysis of a database server with write-heavy workload. In *European Conference on Service-Oriented and Cloud Computing*, pages 184–191. Springer, 2012.

[7] Manfred Dellkrantz, Maria Kihl, and Anders Robertsson. Performance modeling and analysis of a database server with write-heavy workload. 2012.

[8] Maysam Hedayati, Seyed Hossein Kamali, Reza Shakerian, and Mohsen Rahmani. Evaluation of performance concurrency control algorithm for secure firm real-time database systems via simulation model. In *Networking and Information Technology (ICNIT), 2010 International Conference on*, pages 260–264. IEEE, 2010.

[9] ISO/IEC. Database language sql, 2003. ANSI/ISO/IEC 9075:2003.

[10] ISO/IEC. Database language sql, 2003. ISO/IEC 9075:2003.

[11] Maria Kihl, Gustav Cedersjö, Anders Robertsson, and Bertil Aspernäs. Performance measurements and modeling of database servers. In *Sixth International Workshop on*

Feedback Control Implementation and Design in Computing Systems and Networks, 2011.

[12] Edward D Lazowska, John Zahorjan, G Scott Graham, and Kenneth C Sevcik. *Quantitative system performance: computer system analysis using queuing network models*, volume 22. Prentice Hall Upper Saddle River, 1984.

[13] Ami Levin. Sql server: Auto statistics cleanup, 2013. `https://blogs.msdn.microsoft.com/mvpawardprogram/2013/09/09/sql-server-auto-statistics-cleanup/`.

[14] Microsoft. Microsoft sql documentation. `https://docs.microsoft.com/en-us/sql/`.

[15] Craig Mullins. *Database administration: the complete guide to practices and procedures*. Addison-Wesley Professional, 2002.

[16] Mullins, Craig S. The importance of keeping your dbms up-to-date.

[17] MySQL. Mysql, 2012. `https://www.mysql.com/`.

[18] Margesh Naik. Database management system performance analysis and comparison. 2011.

[19] Oracle. Sql standards. `https://docs.oracle.com/cd/B28359_01/server.111/b28286/intro002.htm`.

[20] Rasha Osman and William J Knottenbelt. Database system performance evaluation models: A survey. *Performance Evaluation*, 69(10):471–493, 2012.

[21] PostgreSQL. Postgresql: The world's most advanced open source relational database. `https://www.postgresql.org/`.

[22] Stephen Rago, Aniruddha Bohra, and Cristian Ungureanu. Using eager strategies to improve nfs i/o performance. 2011.

[23] A Saika et al. Comparative performance analysis of mysql and sql server relational database management systems in windows environment. *International Journal of Advanced Research in Computer and Communication Engineering*, 4(3):160–164, 2015.

[24] Margo Seltzer, David Krinsky, Keith Smith, and Xiaolan Zhang. The case for application-specific benchmarking. In *Hot Topics in Operating Systems, 1999. Proceedings of the Seventh Workshop on*, pages 102–107. IEEE, 1999.

[25] SQLite. Sqlite home page. `https://www.sqlite.org/index.html`.

[26] Steve Callan. Database benchmarking. `https://www.databasejournal.com/features/oracle/article.php/3462091/Database-Benchmarking.htm`.

[27] Transaction Processing Council. Tpc-c specification documentation. `http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-c_v5.11.0.pdf`.

[28] Transaction Processing Council. Tpc history. `http://www.tpc.org/information/about/history.asp`.

[29] S Bing Yao and Alan R Hevner. A guide to performance evaluation of database systems. 1984.

[30] Xiaolan Zhang and Margo Seltzer. *Application-specific benchmarking*. Harvard University, 2001.

DEPT. OF COMP. SCI. | LUND UNIVERSITY - FACULTY OF ENGINEERING | PRESENTATION 2018-11-07

THESIS Evaluating and Comparing Performance of Database Systems
STUDENT David Phung and Tobias Ronge
SUPERVISOR Per Andersson (LTH), Sverrir Valgeirsson (Consafe Logistics)
EXAMINER Flavius Gruian (LTH)

A Method for Performance Change Assessment Before a DBMS Upgrade

POPULAR SCIENCE CONCLUSION **David Phung and Tobias Ronge**

In this thesis, instructions for how to perform such impact analysis have been presented, including an application for performance measuring. Such an analysis would make the database developers and administrators feel safer that the database works as before, after an upgrade has been done. And while the area of such impact analysis is huge - it includes timing, memory allocation, output and syntax differences - the work of this thesis can be seen as a step forward in establishing methods for studying these impacts.

Databases are used in a lot of different industries today. If a database fails because of, say, logical errors or lack of memory, there can be devastating consequences. There are several known database management systems available today and many of them are frequently updated. Sometimes, even a switch to a completely different system might be wished for. However, such upgrades and switches are likely to have impacts on the functionality and performance which is why impact analysis should be considered before an upgrade is fully accepted.

In the thesis, a method based on capture and replay - that is, capturing commands sent to the database and replay them - is presented for performance testing, more precisely for measuring memory and timing. Using a test application programmed specifically for this thesis, performance testing can be done on different versions of a well known database management system. By first recording database communication in a so-called trace file, the communication can be replayed on the database a large number of times, resulting in a realistic simulation of real communication while making performance measurements during the execution. The application allows for multiple sequential runs and also presents statistics on the results.

Advantages of the method is that the tester can choose the test environment on which to run the commands, along with the fact that the set of commands can be run over and over again. Using the application, the developer does not need to temporarily take the original database system out of action in order to test it.

The application has been programmed with the intent of allowing future programmers to contribute to the code, allowing for more complex testing and more database management systems to be compatible.