View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Central University Of Technology Free State - LibraryCUT, South Africa

Performance and Reliability Optimisation of a Data Acquisition and Logging System in an Integrated Component-Handling Environment

Bernardus Christian Bothma

Thesis submission in fulfilment of the requirements for the

MAGISTER TECHNOLOGIAE:

ELECTRICAL ENGINEERING

In the

School of Electrical and Computer Systems Engineering

of the

Faculty of Engineering and Information Technology

at the

Central University of Technology, Free State

Supervisor: Prof. H. J. Vermaak

Bloemfontein

February 2011

Declaration of independent work

I, BERNARDUS CHRISTIAN BOTHMA, identity number [redacted], and student number 20307284, hereby declare that this research project submitted for the degree MAGISTER TECHNOLOGIAE: ENGINEERING: ELECTRICAL, is my own independent work that has not been submitted before to any institution by me or anyone else as part of any qualification; and complies with the Code of Academic Integrity, as well as other relevant policies, procedures, rules and regulations of the Central University of Technology.

....

SIGNATURE OF STUDENT **DATE**

Verklaring ten opsigte van selfstandige werk

Ek, BERNARDUS CHRISTIAN BOTHMA, identiteitsnommer 8405245118081 en studentenommer 20307284, verklaar hiermee dat die navorsingsprojek wat vir die verwerwing van die graad MAGISTER TECHNOLOGIAE: INGENIEURSWESE: ELEKTRIES aan die Sentrale Universiteit van Tegnologie deur my voorgelê word, my selfstandige werk is en nie voorheen deur my of enige ander persoon ter verwerwing van enige kwalifikasie voorgelê is nie; dit voldoen aan die kode van akademiese integriteit, so wel as ander relevante beleide, reëls en regulasies van die Sentrale Universiteit van Tegnologie.

....

HANDTEKENING VAN STUDENT **DATUM**

Acknowledgements

I would like to thank the following individuals who contributed towards the completion of this project:

- First and foremost, I would like to thank the Lord God. Without His love, grace and blessings I would not have had the perseverance to complete this project.
- Prof. Herman Vermaak, for his encouragement and being an excellent study leader.
- De Ville Weppenaar, for all the support, encouragement and proofreading.
- Jean Janse van Rensburg, for his encouragement and for assuming the role of network administrator.
- To the Central University of Technology, Free State, for providing the research facilities and financial support.
- The National Research Foundation (NRF) and specifically the Advanced Manufacturing Technology Strategy (AMTS), for providing the necessary funding (under project number AMTS-07-20-P) to complete this study.
- My family and friends who supported and encouraged me throughout this study.
- To all my colleagues and individuals not mentioned here who helped or supported me in any way, thank you.

Summary

Databases are commonly used today, whether for storage of historical data or as a backbone to a transactional system, such as a Point of Sale or e-commerce system. The ideal would be that the data is stored reliably and is always available and quickly accessible. In some cases, with write intensive loads, it is also important to quickly transfer the data to disk. In most cases this ideal is never achieved. This is due to many factors including: hardware and/or network failure or latency, execution time of queries on large databases, software corruption, power failures, etc.

This work is aimed at improving the performance and reliability of an OLE for Process Control (OPC) data acquisition and logging system and at the heart of the matter: the database. The system consists of several key components, including: the database, the data-logging application and a hardware platform. In order to optimise the system, each of these components was evaluated to identify possible areas that could be improved.

The database component focuses on using the capabilities and new features of the MySQL database management system (DBMS) to improve the performance and reliability of the database. The data-logging application component focuses on implementing OPC server and item browsing in the application and improving its performance.

The hardware platform component focuses on improving several smaller components that form part of it. Attention was given to the capabilities of the servers that hosts the applications and databases, and implementation of a backup system. Several Redundant Array of Independent/Inexpensive Disks (RAID) technologies were investigated as a means to increase the performance and reliability of the database server. A means to protect the system from sudden power fluctuations or failures is also considered.

Opsomming

Databasisse word deesdae algemeen gebruik vir die berging van historiese data, of as ruggraat van ’n transaksiestelsel, soos ’n punt-van-verkoop of e-handelstelsel. Die ideaal sou wees dat die data op ’n betroubare manier gestoor word en altyd beskikbaar en vinnig toeganklik sal wees. In sommige gevalle, waar ’n skryf-intensiewe lading betrokke is, is dit belangrik dat die data vinnig na die hardeskyf oorgedra word. In die meeste gevalle word die ideaal nooit bereik nie. Dit is te wyte aan verskeie faktore, insluitend: die versuim of vertraging van die hardeware en/of netwerk, uitvoeringstyd van navrae op groot databasisse, korrupsie van sagteware, kragonderbrekings, ens.

Hierdie werk is daarop gemik om die werkverrigting en betroubaarheid van ’n “OLE for Process Control”- (OPC) data verkryging- en stawingstelsel, en die kern van die saak: die databasis te verbeter. Die stelsel bestaan uit verskeie sleutelkomponente, insluitend: die databasis, die datastawingsprogram, en ’n hardeware platform. Ten einde die stelsel te optimaliseer, is elkeen van hierdie komponente geëvalueer om moontlike areas waarop daar verbeter kan word, te identifiseer.

Die databasis komponent fokus op die gebruik van die vermoëns en nuwe funksies van die MySQL databasis-bestuurstelsel (DBMS), om die werkverrigting en betroubaarheid van die databasis te verbeter. Die datastawings program-komponent is daarop toegespits om die OPC-bediener en itemblaaifunksie in die program te implementeer en die werkverrigting daarvan te verbeter.

Die hardeware platform-komponent fokus op die verbetering van verskeie kleiner komponente wat deel daarvan vorm. Aandag is gegee aan die vermoëns van die bedieners wat die programme en databasisse steun, asook die implementering van ’n rugsteunstelsel. Verskeie “Redundant Array of Independent/Inexpensive Disks”- (RAID) tegnologie is ondersoek as ’n manier om die werkverrigting en betroubaarheid van die databasis-bediener te verbeter. ’n Manier om die stelsel te beskerm teen skielike kragwisselings of -onderbrekings is ook ondersoek.

Table of Contents

DECLARATION OF INDEPENDENT WORK ..... **I**

VERKLARING TEN OPSIGTE VAN SELFSTANDIGE WERK ..... **II**

ACKNOWLEDGEMENTS ..... **III**

SUMMARY ..... **IV**

OPSOMMING ..... **V**

TABLE OF CONTENTS ..... **VI**

LIST OF ACRONYMS ..... **IX**

LIST OF FIGURES ..... **XI**

LIST OF APPENDIX FIGURES ..... **XIII**

1 INTRODUCTION **1-1**

1.1 OVERVIEW 1-1

1.2 STATEMENT OF THE PROBLEM 1-3

1.3 HYPOTHESES 1-4

1.4 AIM OF THE INVESTIGATION 1-5

1.5 METHODOLOGY 1-6

1.6 LIMITATIONS OF THE STUDY 1-7

1.7 ORGANISATION OF THE REPORT 1-8

2 LITERATURE REVIEW **2-9**

2.1 SQL OPTIMISATIONS 2-9

2.1.1 INSERT statement optimisations *2-9*

2.1.2 SELECT statement optimisations *2-10*

2.1.3 DELETE statement optimisations *2-14*

2.1.4 Stored procedures *2-14*

2.2 DATABASE SCHEMA 2-15

2.2.1 Data types *2-15*

2.2.2 Storage Engines *2-17*

2.2.3 Table and Index partitioning *2-21*

2.2.4 Important Server System Variables *2-22*

2.3 OPTIMISATION OF THE DATABASE SERVER 2-24

2.3.1 RAID *2-25*

2.3.2 Data Backup *2-33*

2.4 ENHANCEMENT OF THE SOFTWARE APPLICATIONS 2-38

2.4.1 The Data Acquisition and Logging Client (DALC) *2-38*

2.4.2 Automation of administration tasks *2-38*

2.5 SUMMARY 2-40

3 METHODS AND TECHNIQUES 3-41

3.1 HARDWARE OPTIMISATIONS 3-41

3.1.1 Overview of the Proposed Database, Application and Web Servers *3-41*

3.1.2 RAID Configurations and Benchmark Methodology *3-44*

3.1.3 Design and Implementation of a Backup Scheme *3-45*

3.1.4 Additional Reliability Improvements *3-53*

3.2 OPTIMISATIONS OF THE DATABASE SCHEMA AND DBMS 3-57

3.2.1 Redesign of the Database Schema *3-57*

3.2.2 SQL optimisations *3-59*

3.2.3 Configuration of the Server System Variables *3-62*

3.3 IMPROVING THE DALC 3-63

3.4 SUMMARY 3-64

4 RESULTS 4-65

4.1 HARDWARE INFRASTRUCTURE 4-65

4.1.1 RAID Benchmarks and final RAID configuration 4-68

4.1.2 Backup strategy in Action 4-70

4.1.3 UPS and Power management 4-70

4.1.4 Security and Database user accounts 4-71

4.2 THE MYSQL DBMS 4-72

4.2.1 The Database schema 4-72

4.2.2 Bulk INSERT and Single INSERT Comparison 4-76

4.2.3 SELECT Optimisations 4-77

4.3 FINAL CONFIGURATION OF THE MYSQL SERVER 4-81

4.4 THE NEW DALC 4-88

4.5 SUMMARY 4-90

5 CONCLUSIONS 5-92

5.1 THE CONTRIBUTION OF THE PROJECT 5-92

5.2 CONTRIBUTIONS BY OTHER STUDENTS 5-93

5.3 RECOMMENDATIONS 5-93

5.4 FUTURE WORK 5-95

5.5 CONCLUSION 5-96

APPENDIX A INSERT QUERY BENCHMARK SCRIPTS 98

APPENDIX B SELECT QUERY BENCHMARK 100

APPENDIX C SERVER SYSTEM VARIABLE BENCHMARK SCRIPTS 100

APPENDIX D SQL SCRIPTS TO CREATE THE DATABASE SCHEMA 102

REFERENCES 104

List of Acronyms

ACID	–	Atomicity, Consistency, Isolation and Durability
ATA	–	Advanced Technology Attachment
CPU	–	Central Processing Unit
DALC	–	Data Acquisition and Logging Client
DB	–	Database
DBA	–	Database Administrator
DBMS	–	Database Management System
DDR	–	Dual Data Rate
ECC	–	Error Correcting Code
eSATA	–	External SATA (see SATA)
GA	–	Generally Available
I/O	–	Input/output
IT	–	Information Technology
JBOD	–	Just a Bunch of Disks
KVM	–	Keyboard, Video and Mouse
MOBO	–	Motherboard
MS	–	Microsoft
MTTF	–	Mean Time To Failure

MVCC	–	Multi Version Concurrency Control
NAS	–	Network Attached Storage
OLE	–	Object Linking and Embedding
OPC	–	OLE for Process Control
OS	–	Operating System
PCI	–	Peripheral Component Interconnect
PCIe	–	PCI Express
PSEA	–	Pluggable Storage Engine Architecture
PSU	–	Power Supply Unit
RAID	–	Redundant Array of Independent/Inexpensive Disks
RAM	–	Random Access Memory
RC	–	Release Candidate
SAS	–	Serial Attached SCS
SATA	–	Serial ATA
SQL	–	Structured Query Language
UPS	–	Uninterruptable Power Supply
USB	–	Universal Serial Bus
VT	–	Virtualisation Technology

List of Figures

FIGURE 2.1 OUTPUT OF THE EXPLAIN STATEMENT ON A SELECT QUERY 2-11

FIGURE 2.2 DESCRIPTION OF A TABLE WITH A THREE COLUMN COMPOSITE INDEX 2-12

FIGURE 2.3 THE MYSQL PLUGGABLE STORAGE ENGINE ARCHITECTURE 2-18

FIGURE 2.4 ILLUSTRATION OF THE OLD SYSTEM 2-25

FIGURE 2.5 A FOUR DISK RAID 0 CONFIGURATION 2-27

FIGURE 2.6 A TWO DISK RAID 1 CONFIGURATION 2-28

FIGURE 2.7 A FOUR DISK RAID 5 CONFIGURATION 2-30

FIGURE 2.8 A FOUR DISK RAID 1+0 CONFIGURATION 2-31

FIGURE 2.9 A FOUR DISK RAID 0+1 CONFIGURATION 2-32

FIGURE 2.10 CREATING A SIMPLE EVENT AND STARTING THE EVENT SCHEDULER 2-39

FIGURE 3.1 NETWORK DIAGRAM OF THE PROPOSED SYSTEM 3-42

FIGURE 3.2 BACKUP SCHEDULE – FOR JANUARY 2009 3-47

FIGURE 3.3 BACKUP SCRIPT TEMPLATE 3-49

FIGURE 3.4 THE NEW TRIGGER DIALOGUE 3-50

FIGURE 3.5 THE NEW ACTION DIALOGUE CONFIGURED TO EXECUTE THE BACKUP SCRIPT 3-51

FIGURE 3.6 NEW BACKUP DIALOG IN NAVICAT 3-52

FIGURE 3.7 CREATING A NEW BATCH JOB IN NAVICAT 3-53

FIGURE 3.8 UPSMON CONFIGURATION 3-55

FIGURE 3.9 TIMELINE OF THE ACTIONS TAKEN BY UPSMON WHEN AC POWER FAILS 3-56

FIGURE 3.10 THE SCRIPT FOR CREATING THE TABLE USED IN THE INSERT BENCHMARKS 3-60

FIGURE 4.1 DETAILED NETWORK LAYOUT 4-67

FIGURE 4.2 RAID WRITE PERFORMANCE 4-68

FIGURE 4.3 RAID READ PERFORMANCE 4-69

FIGURE 4.4 THE OPC DATA-LOGGING DATABASE SCHEMA 4-73

FIGURE 4.5 SQL CODE FOR THE TWO EVENTS 4-75

FIGURE 4.6 PERFORMANCE COMPARISON OF BULK RECORD AND SINGLE RECORD INSERT STATEMENTS 4-76
FIGURE 4.7 THE SELECT QUERY TO BE OPTIMISED 4-77
FIGURE 4.8 THE FIRST REVISED VERSION OF THE QUERY 4-79
FIGURE 4.9 THE SECOND REVISED VERSION OF THE QUERY 4-79
FIGURE 4.10 SELECT OPTIMISATION BENCHMARK RESULTS 4-81
FIGURE 4.11 BULK INSERT TRANSACTION BENCHMARK OF THE BUFFER POOL 4-83
FIGURE 4.12 SINGLE INSERT TRANSACTION BENCHMARK OF THE BUFFER POOL 4-84
FIGURE 4.13 BULK INSERT TRANSACTION BENCHMARK OF THE LOG FILE SIZE 4-85
FIGURE 4.14 SINGLE INSERT TRANSACTION BENCHMARK OF THE LOG FILE SIZE 4-85
FIGURE 4.15 BULK INSERT TRANSACTION BENCHMARK OF THE TRANSACTION FLUSH BEHAVIOUR 4-86
FIGURE 4.16 SINGLE INSERT TRANSACTION BENCHMARK OF THE TRANSACTION FLUSH BEHAVIOUR 4-87
FIGURE 4.17 DALC CONFIGURATION TOOL – SERVER INFORMATION 4-88
FIGURE 4.18 DALC CONFIGURATION TOOL – BROWSE FOR OPC SERVERS AND ITEMS 4-89
FIGURE 4.19 THE DATA-LOGGING AND ACQUISITION CLIENT 4-90

List of Appendix Figures

FIGURE A-1 BENCHMARK SCRIPT TEMPLATE USED TO COMPARE THE PERFORMANCE OF SINGLE RECORD AND BULK RECORD INSERT STATEMENTS 98

FIGURE A-2 STORED PROCEDURE TO EXECUTE 10 SEPARATE SINGLE RECORD INSERT STATEMENTS 99

FIGURE A-3 STORED PROCEDURE TO EXECUTE A BULK RECORD INSERT STATEMENT WITH 10 RECORDS 99

FIGURE B-1 SELECT QUERY BENCHMARK TEMPLATE 100

FIGURE C-1 SCRIPT USED TO IMPLEMENT PART A OF THE FIRST BENCHMARK PROCESS 100

FIGURE C-2 SCRIPT USED TO IMPLEMENT PART A OF THE SECOND BENCHMARK PROCESS 101

1 Introduction

1.1 Overview

Databases are commonly used today, whether for storage of historical data or as a backbone to a transactional system, such as a Point of Sale or e-commerce system. The ideal would be that the data is stored reliably and is always be available and quickly accessible. In some cases, with write intensive loads, it is also important to quickly transfer the data to disk. In most cases this ideal is never achieved. This is due to many factors including: hardware and/or network failure or latency, execution time of queries on large databases, software corruption, power failures, etc.

A more achievable approach would be to design a reliable and redundant system, so that if any single component fails it can recover quickly or can, to some extent, continue with its operation. Advances in hardware technologies have led to faster data access and transfer times. Faster hardware in combination with the use of certain strategies like Redundant Array of Independent/Inexpensive Disks (RAID) can lead to a faster and more reliable system. Database management systems, such as MySQL, also have many features available to optimise the database's performance and reliability.

It is also vital to implement a power management system to protect the hardware component against power fluctuations or failures. Untimely power failures and fluctuations can result in data loss or corruption, and in some cases can even cause damage to the hardware. The addition of an Uninterruptable Power Supply (UPS) can protect the system against these failures. The implementation of a proper backup scheme will add another level of data protection. It will ensure that critical data can be recovered even in the event of complete system failure or destruction.

With the generally available (GA) release of MySQL 5.0 in 2005 many new features like views, stored procedures, functions, triggers, Precision Math, and an information schema was added. MySQL 5.0 also introduced the Pluggable Storage Engine Architecture (PSEA), which sets it apart from its rivals. These new features coupled with the new storage engines and performance enhancements over older versions, made MySQL 5.0 a true competitor with commercial offerings like MS SQL Server and ORACLE. The soon to be released MySQL 5.1 (in Release Candidate (RC) status at the time of writing) offers features such as table and index partitioning, row-based replication, and an event scheduler, which makes it even more flexible.

Forming a vital part of the data-logging system, the data-logging and acquisition client (DALC) application performs the task of retrieving the data from the OPC server and logging it to the database. This application needs to be easy to use and configure, while still offering good performance. Automatic OPC item and server browsing are vital features that will be added to the new version of the DALC.

1.2 Statement of the Problem

The existing data-logging system [1], although operational, is still very basic and has room for improvement. It forms part of an integrated component-handling system, and logs all sensory data made available by several OPC servers. Forming part of the component-handling system, while not yet complete, is a website that will give users with sufficient rights the ability to access real-time data from the OPC servers and historic data from the database.

This can lead to the possibility of a great number of users accessing the database at once. Unless the database is re-evaluated and the Database Management System (DBMS) is properly configured, the rate at which queries are answered and the rate at which new data is stored in the database may deteriorate to such an extent that the entire system might come to a halt. The two main factors to improve upon are reliability and performance. Both of these factors are influenced by one or a combination of the parts of the system.

The software component of the data-logging system consists of two main parts: the MySQL database, and the OPC DALC application. The database has the most significant shortcomings; some of which are bound to the limitations of the hardware, and some which are due to improper configuration and ill design. The shortcomings include: very slow read and write speeds, almost no data security and a large log table. The DALC on the other hand, while functional, is by no means user friendly. It has no support for OPC server and Item browsing. The result is that every server, with its items, has to be added manually. Another limitation is that it can only connect to OPC servers on the host computer.

The hardware component consists of the OPC compliant devices, the network, and an application/database server. As mentioned, the shortcomings of hardware itself contribute to the shortcomings of the database. The first problem is that one server has to act as the application server, hosting the OPC servers and other vital software, and the database server. This means that disk access, memory, CPU time and other resources have to be shared, decreasing the overall performance.

The second problem is that the host computer is nothing more than a medium range office desktop with a single hard disk, 2 GB of memory and a dual core processor. The single disk equates to no redundancy, meaning that data will be lost in case of a disk failure. It also means that it has to serve the disk access needs of the operating system, software applications and DBMS. The limited memory capacity means that even if the current server was used as a dedicated database server, it would only be able to cache a small amount of data and indexes in memory.

A third problem is the lack of a proper backup scheme and protection against power failures. With no disk redundancy, the lack of a backup scheme will result in total loss of data in case of a disk failure. With no protection from power failures one not only risks data loss, but also damage to the servers and other hardware.

1.3 Hypotheses

There are many ways to address the above-mentioned shortcomings. The first would be to improve the performance and reliability of the hardware platform. One way to achieve this can is by implementing dedicated application and database servers, ensuring that each has their own dedicated resources to suit their requirements.

A second method may be to implement a dedicated Redundant Array of Independent Disks (RAID) configuration that can serve as a dedicated storage medium for the data files of the database, and further improve both the performance and reliability of the database server.

To further improve the reliability of the hardware, an uninterruptable power supply (UPS) can also be installed to guard against power fluctuations or sudden power failure. Additionally a proper backup scheme can also be implemented to ensure data recovery in case of data corruption or loss.

For the database itself there are several things that can be improved. Re-evaluating how the data relates, and what data types to use for the storage of the data can redesign the schema of the logging database. This can also include a separate table to archive dated logs that are still of interest. New features of MySQL 5.1, like partitioning and events, can also be evaluated.

Another possibility can be to investigate which of the MySQL DBMS's system variables have the greatest influence on its performance, and to fine-tune those variables to make optimum use of the available resources.

Possible ways to improve the performance of the Structured Query Language (SQL) queries can also be investigated. Features such as stored procedures and bulk record inserts can be investigated. Another possibility is to attempt to determine some general guidelines for improving to the performance of SELECT queries.

The final two improvements are related to the DALC and the automation of routine administrative tasks. The DALC can be modified or rewritten to add support for OPC server and item browsing. The user interface will also be changed to increase ease of use. Automating the routine administrative tasks can be done in many ways; these include: scripting, a dedicated application, futures like MySQL 5.1 events, or even third party software.

1.4 Aim of the Investigation

The core objectives of this study are to improve the performance and reliability of the data-logging system. The specific objectives are:

1. To redesign the database by making full use of the capabilities and features of the MySQL database management system (DBMS). This includes the implementation of a table for the archived logs.
2. Investigate and set guidelines for improving query performance.
    - Investigate the possible performance increase between stored procedures and normal ad hoc queries.
    - Investigate bulk record inserts as a possible improvement over single record inserts.
    - Investigate ways to improve SELECT queries.
3. Automate some of the routine database administration tasks.
4. Implement dedicated database and application servers with proper power management and security.

5. Investigate Redundant Array of Independent Disks (RAID) strategies as a means to improve data access performance and data reliability.
6. Fine-tune the initial configuration of the MySQL DBMS to properly make use of the available resources and deliver the best possible write performance.
7. Design and implement a proper backup scheme.

1.5 Methodology

The three main components of the system, which will be focused on, are: the hardware infrastructure (specifically the database server), the database, and the software application(s) that access the database.

Relevant literature, both published and unpublished, and several Internet sources will be reviewed to gain a better understanding of the MySQL DBMS and other relevant technologies. MySQL 5.1.28 RC will be used for this study.

Several improvements will be made to the hardware infrastructure. The first of these will be to implement three dedicated servers, and to make use of RAID technologies in an attempt to improve both the performance and reliability of the database server. Secondly a power management system will be implemented to protect the system against power failures and fluctuations.

The third improvement will be the design and implementation of a backup strategy, because it is vital that regular backups are made of critical data. The design process includes the following steps:

- Identify the backup requirements.
- Identify the storage media that will store the backups.
- Plan the schedule.
- Implement the backup system.

The database schema will be redesigned, making use of the latest features available in MySQL. The design will focus on using the smallest possible data types, ensuring that individual records will be as small as possible. A dedicated table to store archived logs will also be created. This ensures that the size of the "active" log table would stay consistent.

The MySQL DBMS has more than 150 system variables that influence its performance and behaviour. The variables that would have the greatest influence on the performance of the DBMS, and more specifically the primary storage engine, will be properly configured to take full advantage of the available resources. These system variables will then be fine-tuned and benchmarked several times.

The DALC will be modified, and possibly rewritten, as to include the new features and improve its ease of use.

1.6 Limitations of the Study

The greatest limitation of this study will be with the hardware. The current database server is based on desktop computer technology, thus any optimisations to it will be limited to compatible technology. This also sets limits to the number of hard disks that can be used in the RAID arrays and the performance gains when compared to similar configuration using server platform technology.

As for the limitations concerning the database; the study will not take into consideration the technologies available for distributed database systems, like replication, since currently a well performing central database system will suffice for the projected work load. An in-depth study of these technologies can be done at a later stage when the need arises and database traffic becomes a prominent factor. A second limitation regarding the database study is related to the storage engines. All engines that are distributed with the MySQL DBMS will be evaluated and their features compared, but the comparative study, in order to determine which will be used for the active and archive databases will be limited to the three or four candidates that appear to be best suited.

1.7 Organisation of the Report

The report has been divided into five chapters.

- **Chapter 1 Introduction:** This chapter gives an outline of the study. It deals with the objectives and limitations thereof.
- **Chapter 2 Literature Review:** This chapter is dedicated to illustrate the relevant literature related to the study. The topics that are discussed in here include:
  - The optimisation of different types of SQL statements.
  - Evaluation of the feature and capabilities of MySQL that can be used to improve the database schema.
  - Reviewing RAID technologies as a possible optimisation to the database server.
  - Review of backup technologies and techniques.
- **Chapter 3 Methodology:** The method, techniques and equipment used in this study are described here. The different benchmarks that will be performed are discussed in detail.
- **Chapter 4 Results:** The results of the study are presented and evaluated in this chapter. The final configuration and implementation of the hardware component, database schema and MySQL DBMS are also described here.
- **Chapter 5 Conclusion:** This chapter offers a brief overview of the study, results and conclusions that were reached as a result of this work. Several recommendations are made, based on the results of the study. Some possibilities for future studies are also mentioned here.

2 Literature Review

Before any optimisations can be done to the OPC data-logging system it is necessary to become familiar with the techniques and technologies that can be used to optimise it. The optimisations to the system are split into several categories: SQL optimisations, database schema optimisations, optimisations to the MySQL DBMS, hardware optimisations of the database server, optimisations to the DALC and other applications. The optimisations focus on the reliability and performance of the system.

2.1 SQL Optimisations

One way to increase the performance of all queries is to design the database schema to be (in terms of record size) as compact as possible. This can be done by using the smallest data types possible to define table columns. Section 2.2.1 discusses this in more detail.

2.1.1 INSERT statement optimisations

The time required to insert a record into a table is determined by the following factors [2]:

- The time needed to establish a connection to the database.
- The time it takes to send the query.
- The time taken to parse the query.
- The time taken to insert the record.
- The time to insert the indexes.
- Time to close the connection.

These do not factor in the initial overhead of each concurrent query opening the table. Also, INSERT statements can gradually become slower as a table grows in size [2]. Another important issue to note is that the time needed to insert the record and index depend on the size of the record and the number of indexes. Large records and redundant indexes can thus greatly influence the performance of an INSERT query. There are, however, several general ways to speed up INSERT statements.

Client applications that insert many records at the same time can make use of bulk record inserts. Bulk record inserts are INSERT statements that make use of the VALUES keyword to insert multiple records at once [2]. For a scenario with a lot of client applications inserting many records, the INSERT DELAYED statement can be used. In cases where thousands of records needs to be loaded from a text file, the LOAD DATA INFILE statement can be used [2]. LOAD DATA INFILE can be up to 20 times faster than the equivalent INSERT statement. This can be particularly useful when restoring the backed up data of very large tables.

Additionally there are several storage engine specific optimisations that can be applied. For tables based on the MyISAM storage engine, the bulk_insert_buffer_size system variable has an influence on the performance of bulk record inserts [2]. The key_buffer_size system variable will influence the performance of INSERT and LOAD DATA INFILE statements. In both cases larger values for these variables can result in increased performance. For InnoDB based tables changing the value of the *innodb_flush_log_at_trx_commit* system variable to 0 or 2 (explained in section 2.2.4) can significantly improve the performance of all INSERT statements, at the cost of reliability [2].

2.1.2 SELECT statement optimisations

Optimising SELECT queries is important, as they are the means to retrieving the information from the database. Optimised SELECT statements mean faster query execution, leading to faster information retrieval and more queries serviced. There are several ways to optimise SELECT queries. In this section the following will be discussed: gaining information on query execution with the EXPLAIN statement, optimising SELECT queries using indexes, and using the query cache to improve SELECT performance.

2.1.2.1 Query Execution Information Using EXPLAIN

The EXPLAIN statement can be used in conjunction with a SELECT statement to gain information on the query execution plan from the query optimiser [2]. Figure 2.1 shows the output of EXPLAIN of a simple SELECT query on the world database [3].

```
EXPLAIN
SELECT city.Name, country.Continent, country.SurfaceArea, country.GovernmentForm
FROM city, country, countrylanguage
WHERE countrylanguage.Language = 'English'
AND countrylanguage.IsOfficial = 'T';
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	country	ALL	(NULL)	(NULL)	(NULL)	(NULL)	239	
1	SIMPLE	countrylanguage	ALL	(NULL)	(NULL)	(NULL)	(NULL)	984	Using where; Using join buffer

Figure 2.1 Output of the EXPLAIN statement on a SELECT query

Each row in the EXPLAIN output contains information on one table used in the query execution. Each row consists of several columns containing the following information:

- The *id* column: Identifies the SELECT in the query. *Id*s are numbered sequentially.
- The *select_type* column: The type of SELECT used to resolve the query.
- The *table* column*:* Shows which table the output is about.
- The *type* column: Shows what type of join is used by the query optimiser.
- The *possible_keys* column: Shows a list of the keys, of the table in question, *that can be* used.
- The *key* column: Shows which key (index) is actually used.
- The *ref* column: Shows which constants or columns are compared to the selected index (key).
- The *rows* column: Shows the estimated number of rows that must be examined to execute the query.
- The *extra* column: Displays additional information on how the query is resolved. The *using filesort* and *using temporary* values should be avoided, as they will slow the query down. A complete list of all the values for this column can found in the MySQL reference manual [**2**].

This information will indicate what is causing the query to be slow, and can be used as a guideline for the query optimisation.

2.1.2.2 Using Indexes to Optimise SELECT Queries

One of the primary reasons for low query performance is badly defined or omitted indexes. Indexes are stored in a separate file and are used to quickly retrieve specific column values [2]. If no index is present for the column in question, the entire table must be scanned to retrieve the data. This can be very slow, especially with large tables containing thousands of records.

If an index is present for the column in question, MySQL can use it to quickly determine where in the table to start scanning for the relevant records. This can be up to a 100 times faster for a table with 1 000 records, than a full table scan [2]. Note that indexes do have some additional overhead and might slow down UPDATE and INSERT statements.

In a case where multiple indexes are available, the index that finds the least records is used. This can provide a significant performance increase for queries that make use of table joins or the WHERE, GROUP BY or ORDER BY clauses.

```
CREATE TABLE `t1` (
  `col1` INT(11) NOT NULL,
  `col2` INT(11) NOT NULL,
  `col3` CHAR(1) NOT NULL,
```

Figure 2.2 Description of a table with a three column composite index

MySQL also allows composite indexes, which is indexes that span multiple columns. Composite indexes allow the query optimiser to use any leftmost prefix of index to find records. For example, consider the CREATE TABLE statement in Figure 2.2.

The table **t1**, created with the code in Figure 2.2, will have a composite index on col1, col2 and col3. This will allow that an index search can be performed on (*col1*), (*col1, col2*) or (*col1, col2, col3*). The following section will discuss several considerations when querying indexed columns.

2.1.2.2.1 Considerations when using indexes

When using indexed columns in table joins, one must ensure that they are of the same type and size [**2**], e.g. CHAR and VARCHAR columns are considered the same if their size is the same, i.e. CHAR (20) and VARCHAR (25) are not considered the same, while CHAR (25) and VARCHAR (25) are. Dissimilar columns, whose values cannot be compared without conversion, can prevent the use of the indexes [**2**].

If possible, calculations on an indexed column should be avoided, as this can lead to a table scan being used instead of the index [**4**]. For example: "SELECT *col_1* FROM *table_name* WHERE *col_2**2 > 30." If there is an index on *col_2* it will not be used, because each record has to be scanned and its *col_2* value multiplied by two before the comparison can be done. To make use of the index the query can be changed as follows: "SELECT *col_1* FROM *table_name* WHERE *col_2* > 30/2." This query can be optimised even more if the calculation (30/2) is replaced with a constant (15).

2.1.2.3 Using the query cache to increase SELECT query performance

The query cache offered by the MySQL DBMS can significantly improve the performance of SELECT queries. The query cache stores the text of a SELECT statement with its result set. If an identical query is received, the result set is retrieved from the query cache instead of executing the statement again [**2**]. The query cache is shared amongst all sessions. This means that a result set of a query sent by one client can be used as the response to an identical query from another client.

The query does not store invalidated data. If a table or its data is altered, the query cache flushes all relevant result sets. It is most useful in read-only or read-mostly scenarios. The MySQL manual contains a list of constraints that must be considered when making use of the query cache [**2**].

2.1.3 **DELETE statement optimisations**

MySQL does not offer many optimisations for DELETE queries. Here are some of the optimisations that were reviewed [**2**]:

- The required time to delete a record from a table is exactly proportional to the amount of indexes on that table.
- For tables based on the MyISAM storage engine the *key_buffer* system variable can be increased to increase the speed of DELETE statements.
- In cases where all the records of a table are to be deleted, a TRUNCATE TABLE statement will offer better performance than a DELETE FROM statement.

The optimisations discussed in section 2.1.2.2.1 for SELECT queries also apply to DELETE Queries. Partitioned tables also offer a means to delete a large number of records, by using an ALTER TABLE DROP PARTITION statement. This can be significantly faster than a DELETE statement using a WHERE clause [**5**]. To prevent large deletes from slowing down other queries it is suggested to split them into a series of smaller batches.

2.1.4 **Stored procedures**

Stored procedures are a much-debated topic amongst Database Administrators (DBAs), with some arguing for [**6**] and others against [**7**] using them. Both groups focus on security, performance and maintainability. A stored procedure is simply a set of SQL statements that are stored in the MySQL server. These stored procedures can be used by client applications instead of issuing individual SQL statements over and over [**2**].

MySQL reasons that their implementation of a stored procedure will be faster, as there is less network traffic between client and server [2]. This results in an increase in Central Processing Unit (CPU) load at the database server. By reducing the traffic between server and client and shifting the query processing to the database server, one lowers the hardware requirements of the computer running the application (normally an application or a web server). But in most cases it is more expensive to scale a database server than an application or web server [8]. It is also important to understand that the stored procedure's performance is reliant on the underlying query. A slow and badly written query will result in a slow stored procedure.

Stored procedures do offer some additional advantages. By moving some of the application logic to the MySQL server, client application developed for different platforms and in different languages can perform the same database operations without the need to implement it in each client. This also comes at a cost: MySQL's stored procedure language is slow [8]. Morgan Tocker, a MySQL AB support engineer, demonstrated that it can be up to 10 times slower than the PHP scripting language with mathematical computations [9].

2.1.4.1 Compound statements

Besides SQL statements, MySQL also allows for compound statements to be used in a stored procedure [10]. Compound statements allow you to use selection statements, like IF statements and loops in stored procedures. Compound statements are enclosed by the BEGIN and END keywords. One can also define variables by using the DECLARE statement.

2.2 Database Schema

2.2.1 Data types

MySQL offers a number of data types. The data types can be divided into one of three categories: numeric types, string types and date and time types [2]. MySQL also supports spatial data types, but these will not be discussed here. Information on spatial data types is available in the MySQL manual [2].

When designing a database schema it is important to select the correct data type for each column. For instance, calculations that are done with string types will have different results from those done with numeric types, i.e. '2' + '2' = '22', whereas 2 + 2 = 4.

Another important factor to consider concerning schema design is using smallest possible data types, i.e. using TINYINT instead of BIGINT for columns with a small numeric range. This will increase the performance of most queries as records, and indexes, will be smaller, resulting in reduced I/O times. Another advantage of smaller records is that more records can be cached in memory, possibly increasing the performance of SELECT queries even further.

2.2.1.1 Numeric Types

The supported numeric types include the standard SQL numeric types [**2**]. These include: INTEGER, SMALLINT, FLOAT, REAL, DOUBLE PRECISION, NUMERIC and DECIMAL. In addition to these standard numeric types, MySQL supports several nonstandard numeric types. These include TINYINT, MEDIUMINT and BIGINT. The BIT and BOOLEAN types are also available, although BIT is not supported by all of the storage engines.

The characteristics of the nonstandard numeric types, BIT and BOOLEAN types need to be considered before they can be properly used. TINYINT can store integer values from (-2^{7}) to $(2^{7}-1)$ and requires only one byte for storage. MEDIUMINT is a 3-byte integer type that can store values from $(-2^{23)}$ to $(2^{23}-1)$. BIGINT enables the usage of large integer values; it can store values ranging from $(-2^{63)}$ to $(2^{63}-1)$, but requires eight bytes for storage.

The BOOL type is implemented as synonym for TINYINT(1) [**2**]. A zero value is considered as FALSE, while non-zero values are considered true. The BIT type is used to store bit field values. A bit field declared as BIT(***M***) can store *M* bits. The length of a bit field can range from 1 to 64 bits. If (***M***) is omitted a default value of 1 is used. MySQL also supports several attributes for its numeric types, including: ZEROFILL, UNSIGNED, and AUTO-INCREMENT [**2**].

2.2.1.2 Date and Time Types

The supported date and time types that are supported by MySQL include: DATE, DATETIME, TIMESTAMP, TIME, and YEAR. Each of these types has a range of legal values, including a "zero" value. The zero value can be used to indicate an illegal value. The zero value can be more convenient to use than a **NULL** value, as it takes less data and index space to store. It is worth noting that the DATETIME, TIMESTAMP and TIME types do not support microsecond precision.

2.2.1.3 String Types

MySQL supplies several data types to store string types. These string types can be divided into categories: character and binary strings. The supported character strings include: CHAR, VARCHAR, TEXT, ENUM, and SET. The supported binary string types include: BINARY, VARBINARY and BLOB.

2.2.2 Storage Engines

The main difference between MySQL and other DBMSs, whether open source or proprietary, is its pluggable storage engine architecture [**11**] (PSEA). The PSEA allows you to select a specialised storage engine for a particular application, without concern for the low-level implementation details at the storage level. This means that while the capabilities of the storage engines can differ, the application is completely shielded from them. Figure 2.3 graphically illustrates the PSEA. The storage engines themselves are the components of the database server that actually perform actions on the underlying data [**11**]. The following sections will discuss several storage engines that are distributed with MySQL.

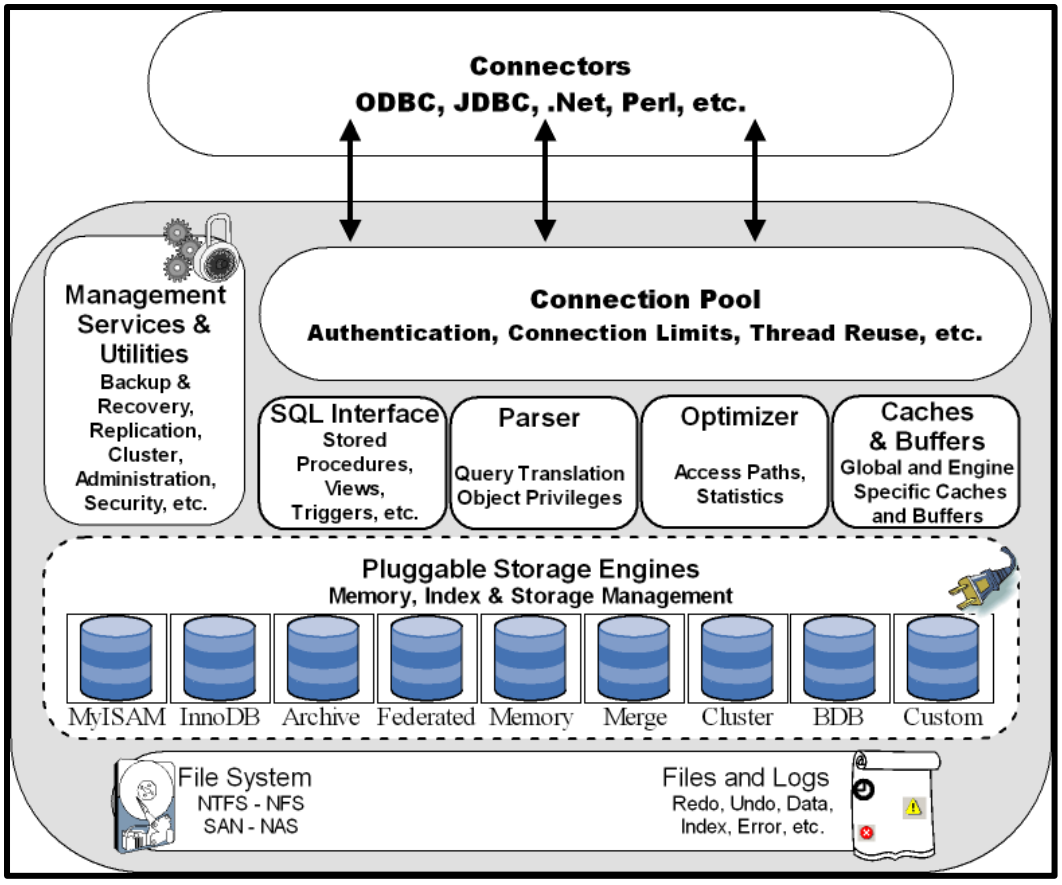


Figure 2.3 The MySQL Pluggable Storage Engine Architecture [2]

2.2.2.1 MYISAM

MyISAM is the MySQL DBMS's default storage engine. MyISAM is a non-transactional storage engine. Tables based on the MyISAM storage engine have the following characteristics:

- High-speed writing and retrieval of data [**2**].
- Table level locking granularity [**12**].
- Support for B-tree and full-text indexes [**12**].

It is important to note that MyISAM only caches indexes in memory and relies on the operating system (OS) to cache the data [**2**]. MyISAM tables can be stored in one of three formats: fixed length (static), dynamic, and compressed [**2**]. MyISAM tables are ideal for data marts, traditional data warehousing [**12**] and logging application [**13**].

2.2.2.2 **MEMORY**

The MEMORY storage engine creates tables with contents stored in memory [**2**]. MEMORY tables store only a definition (or format) file on disk. MEMORY tables support both hash and B-tree indexes [**2**]. These tables lose their data when the server is shut down, but the tables themselves will continue to exist.

2.2.2.3 **ARCHIVE**

ARCHIVE is designed to store historic and seldom referenced data with a very efficient footprint [**14**]. When compared to compressed MyISAM tables, ARCHIVE offers several advantages. ARCHIVE automatically compresses data stored in its tables [**15**], so there is no need for the use of a command-line utility as with MyISAM tables. Compressed MyISAM tables are read-only, where as ARCHIVE tables support both INSERT and SELECT statements. In this regard the only limitation is that data cannot be updated or deleted by using the UPDATE, DELETE or REPLACE statements [**2**]. ARCHIVE also supports multi-version concurrency control (MVCC) and row-level locking [**12**]. ARCHIVE does not support FOREIGN KEY constraints.

As of version 5.1 of MySQL, ARCHIVE has been enhanced to have faster I/O operations and lower memory requirements [**14**]. Support for auto-increment columns has also been added; unlike other columns that do not support indexing, they support either unique or non-unique indexes [**2**].

ARCHIVE is ideal for storing and retrieving large amounts of seldom-referenced historical and archived data or security audit information [**11**], i.e. sensitive information that should not be altered in any way [**12**].

2.2.2.4 InnoDB

InnoDB is one of MySQL's partner-developed storage engines. It is developed by Innobase OY, a subsidiary of Oracle [**11**]. InnoDB is a transaction safe storage engine and the most popular transactional storage engine for MySQL. InnoDB has been distributed with MySQL since 2001[**16**]; it is included in the binary distribution of MySQL.

Unlike MySQL's native storage engines (those developed by MYSQL AB), InnoDB stores its tables and indexes in a table-space [**2**]. This table-space may consist of several files. All tables and their indexes are stored in this one table-space. For flexibility one can configure InnoDB to use multiple table-spaces, sometimes referred to as per-table table-spaces.

With multiple table-spaces, each InnoDB table and its indexes are stored in its own file or table-space [**2**]. When using multiple table-spaces the table data and indexes are stored in an *.ibd* file, while the table definition is stored in an *.frm* file [**2**]. The names of both files start with the table name. Multiple table-spaces are useful in scenarios where you wish to place different InnoDB tables on different physical disks or wish to quickly restore a backup of specific tables without disrupting the use of the other InnoDB tables.

Besides providing MySQL with transaction-safe tables it also offers commit, rollback and crash recovery capabilities [**10**]. Unlike most other storage engines it has row level locking granularity and FOREIGN KEY constraint support. It also has support for B-Tree and clustered indexes. Unlike with MyISAM tables, both the data and indexes of InnoDB tables are cached; this is possible since InnoDB maintains its own buffer pool.

2.2.3 Table and Index partitioning

As of version 5.1 of MySQL, user-defined partitioning [**17**] is supported. Even before partitioning was supported the InnoDB storage engine supported the notion of table-spaces and the MySQL server itself could be configured to store different databases in different physical directories. Partitioning allows an even greater flexibility, by allowing you to store different portions of a table across a file-system according to user set rules. Partitioning potentially offers increased performance and simplified data management [**18**].

The table sections are referred to as table partitions (or fragments [**19**]) and the user-selected rules that define how the data is divided are referred to as the partitioning function (or expression) [**17**]. The function must either return an integer value or NULL. For date or time based partitioning, most partition types require you to use a function that operates on a DATE, TIME or DATETIME column and returns a valid value.

There are two main categories for partitioning: horizontal partitioning and vertical partitioning. With horizontal partitioning a relation (or table) is divided along its tuples (or rows) [**19**]; meaning different rows are assigned to different physical partitions [**17**]. With vertical partitioning different columns of a table are assigned to different physical partitions. MySQL supports only horizontal partitioning.

MySQL offers several types of horizontal partitioning. These include [**2**]:

- RANGE partitioning
- LIST partitioning
- HASH partitioning
- KEY partitioning

2.2.4 Important Server System Variables

Even though MySQL has many system variables, only a few are important for the performance of the server [**20**]. When setting up or fine-tuning system variables you need to consider the workload, the storage engines used, and the OS and hardware used. Following below is a list of the most important variables with a description of each.

- **key_buffer_size**

  This variable determines the size of the key buffer (or key cache) that is used to cache the indexes of MyISAM tables in memory. For databases where the majority of the tables are based on MyISAM, it is recommended to set the key buffer to approximately 25% of the system memory [**2**]. Some experts recommend setting the key buffer up to 40% of the system memory [**20**]. The maximum size of the key buffer is 4 GB on 32-bit platforms. On 64-bit platforms it is limited by the capabilities of the OS [**2**].

- **query_cache_size**

  This variable determines the amount of memory allocated to the query cache. By default the query cache is set to 0, which also disables it. Its size must be a multiple of 1024, and a minimum size of 40 KB is required to allocate its structures [**2**]. It has a 4 GB size limit on 32-bit platforms, and an approximate $(2^{64} - 1 \text{ bytes})$ for 64-bit platforms.

- **thread_cache_size**

  Enables the thread cache. With the thread cache enabled, MySQL caches the threads of disconnecting clients for reuse. Threads of disconnected clients are places in the thread cache, while the number of cached threads is less than the *thread_cache_size* [**2**]. New threads will only be created when the thread cache is exhausted. The default value of the *thread_cache_size* is 0.

- **table_open_cache**

  The *table_open_cache* variable determines how many tables can be kept open by the MySQL server for all threads [**2**]. It has a default value of 64 and maximum value of 524288. Note that with larger values you encounter open file limits imposed by the OS. It is recommended that this variable be set to at least $(max\_connections \times N)$, where N is the maximum number of joins in any of the queries to be executed [**2**]. There is currently a limit of 2048 open files for MySQL 5.0 and 5.1 on Windows-based systems [**21**]. This problem has been addressed in MySQL 5.5 (currently in Beta).

- **sort_buffer_size**

  This buffer influences the performance of ORDER BY and GROUP BY operations. Each thread that needs to do a sort is allocated a buffer of this size [**2**]. The default size of the sort buffer is approximately 2 MB. On 32-bit systems it has a maximum value of 4 GB and approximately $(2^{64} - 1 \text{ bytes})$ on 64-bit systems. If only simple queries are executed this variable is of little importance [**20**].

- **innodb_buffer_pool_size**

  The size buffer pool is very crucial to performance of the InnoDB storage engine. It is used to cache both data and indexes in memory [**2**]. On dedicated database server, with a database that primarily uses InnoDB based tables, the buffer pool can be set to approximately 70% of the available system memory [**20**].

- **innodb_log_file_size**

  This variable determines the size of each log file in the log file group [**2**] of the InnoDB storage engine. The default size is 5 MB. Larger log files need less checkpoint flush activity, meaning less disk I/O [**2**]. The disadvantage of large log files is that crash recovery will be slower. The combined size of the log files must be less than 4 GB [**2**]. The number of log files is determined by the *innodb_log_files_in_group* variable.

- **innodb_log_buffer_size**

  This variable sets the size of the buffer used by InnoDB to write to the log files on disk [**2**]. The default size is 1 MB and the maximum size is 4 GB. Larger values will be beneficial to large transactions.

- **innodb_flush_log_at_trx_commit**

  The *innodb_flush_logs_at_trx_commit* variable controls InnoDB's log flush behaviour. By default it is set to 1. This means that the log will be flushed to disk on each transaction commit. It is quite costly in terms of performance, but is required for ACID (Atomicity, Consistency, Isolation, and Durability) compliance [**10**]. In cases where ACID compliance is not that important, i.e. when migrating from MyISAM to InnoDB for foreign key support, its value can be set to 0 or 2. When set to 0 no flush to disk operation will be done at the end of a transaction. Instead the buffer is written out to the log and flushed to disk once per second. This can lead to the loss of the last second's transactions, in the event that MySQL crashes [**2**]. When it is set to 2 the buffer is written to the log file on each commit, but then no flush to disk operation is performed. The flush to disk operation is still performed once per second by InnoDB [**10**]. This setting offers better performance than option 2 and higher reliability that option 0, but it still has some drawbacks. An OS or power failure can still cause the loss of the last second's transactions. Using a battery backed disk cache in the disk controller can speed up disk flushes, making the operation safer [**10**].

2.3 Optimisation of the Database Server

It is crucial to have a proper hardware platform to accurately make use of the capabilities of the MySQL DBMS. The current database server is far from this ideal. This is illustrated in Figure 2.4. The database server doubles as an application server for hosting the OPC servers, client and other applications. Another problem is that it is basically a standard office computer. This means that MySQL has to share what little resources that are available with other applications. Also the current server, with only a single 150 GB hard disk, offers very little in terms of storage capacity, I/O capabilities and reliability. Migrating to a dedicated database server will address some of the problems, but it will not necessarily address the limitations concerning input/output (I/O) capabilities or reliability.

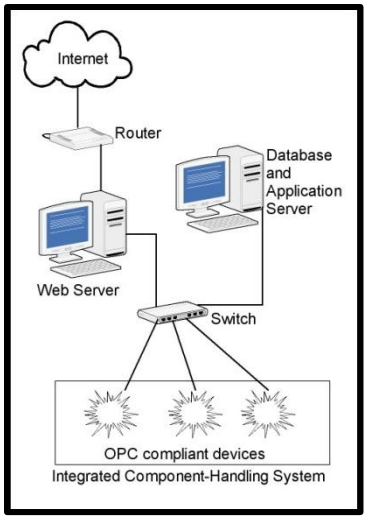


Figure 2.4 Illustration of the Old System

RAID is the traditional method of increasing both I/O capabilities and reliability of the storage medium. It configures several hard disks into an array of disks that are treated as a single volume by an OS. Data is then distributed across the disks, so that it can be accessed in parallel and thus increase the I/O rate. To ensure the safety of the data even further it is necessary to implement a backup scheme. The following section provides an in-depth review of RAID technologies and backup schemes and mediums.

2.3.1 **RAID**

RAID is an acronym defined as Redundant Array of Inexpensive Disks by Patterson *et al.* [**22**]. It was later redefined as Redundant Array of Independent Disks [**23**]. RAID was originally developed, in the 1980s, to address what was at the time considered the pending I/O crises [**22**]. The I/O crises had to do with the rapid pace at which the performance and capacity of Central Processing Unit (CPU) and Random Access Memory (RAM) technologies improved, while those of the I/O system like magnetic disks or networks were lagging behind. This would eventually lead to programs becoming I/O bound, making faster CPUs and RAM useless. To counter this problem Patterson formally defined several levels of RAID, one to five.

Patterson *et al.* suggested that by replacing a single large disk with an array of inexpensive disks could be a solution to the I/O problem, as small requests can be serviced independently, while larger requests can be split over several disks to transfer in parallel. The biggest problem with an array of disks is the reduction in reliability [**22**]. With each disk that is added the reliability of the array is significantly reduced. The Mean Time To Failure (MTTF) of an array can be calculated as in expression (2:1) below [**22**]:

$$MTTF\ of\ a\ Disk\ Array = \frac{MTTF\ of\ a\ Single\ Disk}{Number\ of\ Disks\ in\ the\ Array} \qquad (2:1)$$

This means if one large disk were replaced with 10 smaller disks, the reliability of the array would be reduced by a factor of 10 when compared to the single disk. Patterson suggests that by adding some form of redundancy to the array of disks, forming a RAID, can improve the reliability of the array beyond that of a single disk [**22**]. This means that RAID has the advantage of improved performance and reliability, when compared to a single disk.

Today the standard RAID levels include RAID 0 to 6, although RAID 2 and RAID 3 are rendered obsolete by RAID 4 [**24**] and RAID 5 has mostly replaced RAID 4. In addition to the standard RAID levels there are also several nested (or hybrid) and non-standard RAID levels. The nested RAID levels are combinations of standard RAID levels, while the non-standard RAID levels tend to be enhanced vendor-specific versions on the standard RAID levels. The following sections discuss the most popular RAID levels.

2.3.1.1 **RAID 0**

RAID 0 is also known as striping or non-redundant disk array [**23**]. It is similar to concatenation [**24**], in that a RAID 0 array is constructed out of multiple smaller disks. RAID 0 differs from concatenation [**25**] in that the data is striped (distributed) over all the disks, increasing the I/O throughput. RAID 0 was not part of the original RAID levels defined by Patterson *et al.* [**22**], but was later included. Figure 2.5 shows how a RAID 0 array is implemented.

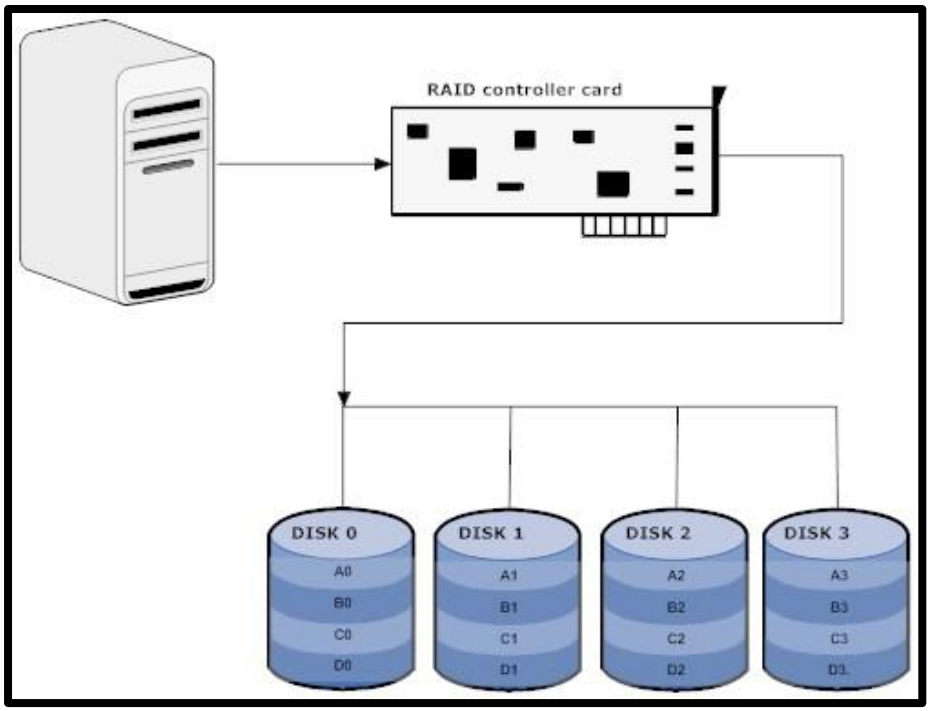


Figure 2.5 A Four Disk RAID 0 Configuration

It is important to note that RAID 0 offers no redundancy whatsoever. This has some advantages and disadvantages. In terms of cost RAID 0 is considered as the least expensive RAID level [**23**], as all of the disks in the array are used for data storage. No disks are necessary for redundant data. The lack of redundancy means that RAID 0 offers the best write performance. In terms of read performance, RAID 0 is surprisingly not the fastest, and it can be outperformed by configurations implementing mirroring [**23**]. The lack of redundancy is also RAID 0's greatest disadvantage. With no redundancy, if any one disk in the array fails all data is lost. RAID 0 is recommended for environments where capacity is more important than reliability.

2.3.1.2 **RAID 1**

RAID 1 is also referred to as mirroring. It is the traditional and easiest way to get redundancy. With RAID 1 the data of one disk is mirrored on to another. It offers great reliability and availability; you always have an exact replica of your data available. If one of the two disks were to fail, the system can continue normal operation from the other with little or no data loss. Figure 2.6 shows how a RAID 1 array is implemented.

In terms of performance RAID 1 offers no increase in write performance [**26**], however it allows reads to occur in parallel [**22**], increasing the read performance. In terms of cost RAID 1 is the most expensive, since twice as much disk is needed when compared to other RAID or non-RAID systems. This leaves you the choice between doubling the cost of the system's storage or only using 50% of the available storage. It is important to note that RAID controllers, that only handle RAID 0 and RAID 1, are normally cheaper than those that handle the other RAID levels as well.

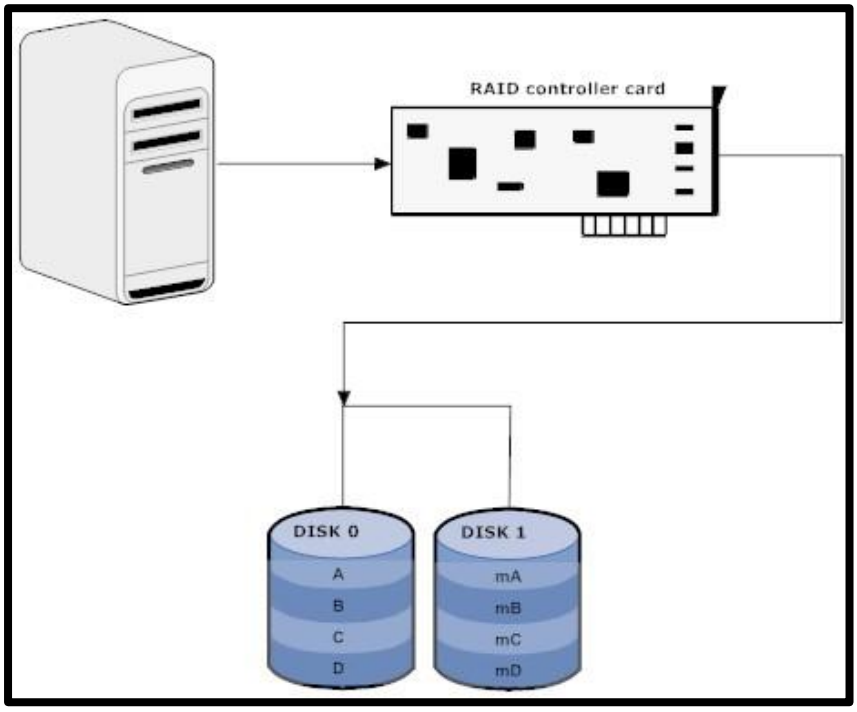


Figure 2.6 A Two Disk RAID 1 Configuration

RAID 1 is recommended for environments where availability is more important than performance [**23**]. It is however important to never think of RAID 1 as a replacement for a proper backup system.

2.3.1.3 RAID 2 to 5

The shortcomings of RAID 1 inspired RAID levels 2 to 5 [**22**]. To fully understand how RAID 5 works one must first understand it predecessors. RAID 2 distributed the data across a set of disks by using bit-interleaving (bit level striping [**26**]), combined with Hamming code for the error correcting code (ECC) [**22**]. The data and ECC information is stored on separate disks [**26**]. By using multiple check disks it is not only possible to detect an error, but it is also possible to determine which disk is erroneous [**22**]. For a configuration using 25 data disks, five ECC disks are required. The Hamming codes are calculated and written to the ECC disks when the data is written to the disks, and is again calculated when the data is read [**26**].

RAID 2 was never successful due to the complex and expensive RAID controllers, and the minimum number of disks needed to implement it [**26**]. RAID 3 was offered as an improvement over RAID 2, and focused in reducing the cost.

RAID 3, like RAID 2, used bit-interleaving, but with a parity instead of Hamming code [**23**]. Since parity is used to check the data integrity only, one dedicated check (parity) disk is necessary in the disk array. This is possible since most disk controllers can detect which disk failed [**22**]. RAID 3 succeeded in reducing the redundancy cost to the minimum, but left room for improvement in small access times [**22**]. RAID 4 aimed to improve performance without sacrificing reliability or increasing the cost.

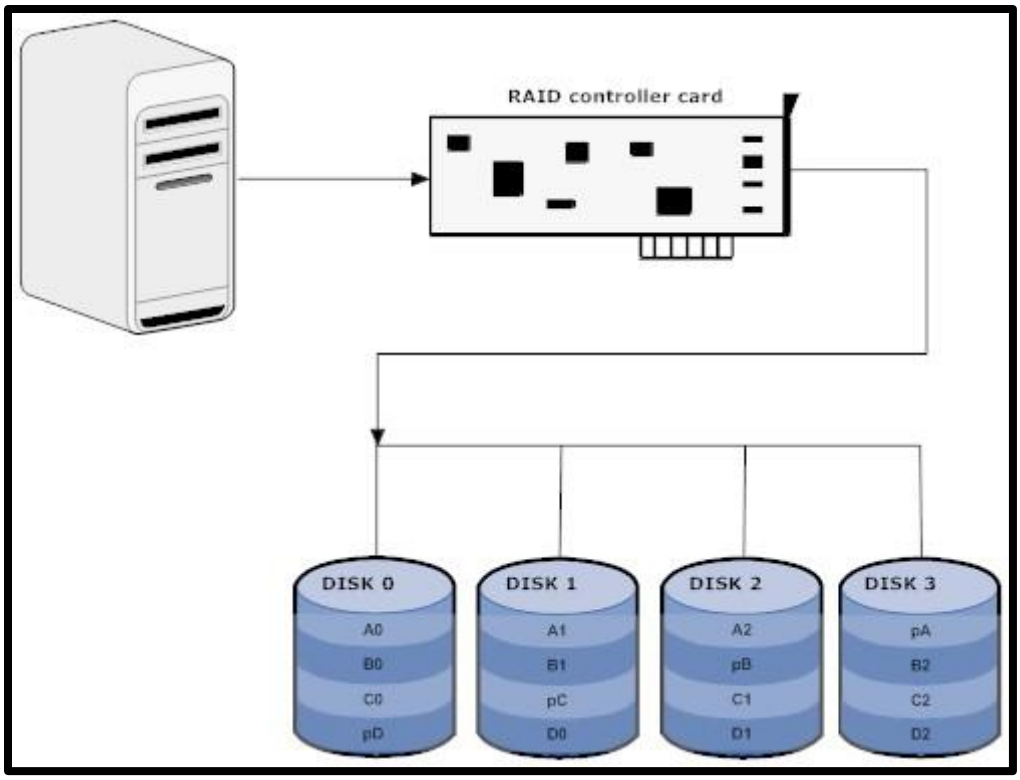


Figure 2.7 A Four Disk RAID 5 Configuration

RAID 4 was very similar to RAID 3 in that it used striping with parity and a single check disk. The difference being that RAID 4 uses sector level striping (block-interleaving [**23**]) instead of bit level striping. This allows independent disk reads at the maximum disk rate, while maintaining ECC capabilities [**22**]. Using sector level striping also gives you the advantage of being able to change the stripe size to suit the needs of the applications [**26**]. RAID 4 still had one problem, which is: the check disk remained the final bottleneck [**22**]. RAID 5 was the answer to this problem.

RAID 5 attempts to solve the bottleneck of RAID 4 by distributing the parity across all disks. Figure 2.7 illustrates this. The advantage of this is that RAID 5 now allows multiple writes [**22**]. This also means that if one disk fails all missing data can be calculated from the remaining disks [**26**]. RAID 5 thus offers the best balance between performance, cost and reliability. Note that RAID 5 has the effective storage capacity of N-1 disks and require at least three disks to be implemented.

2.3.1.4 RAID 1+0 and RAID 0+1

RAID 1+0, like all nested RAID levels, is not part of the standard RAID levels. Also like all nested RAID levels it attempts to use the advantages of two or more standard RAID levels. RAID 1+0, sometimes referred to as RAID 10 [**25**], combines the attributes of striping (RAID 0) and mirroring (RAID 1). RAID 1+0 can be described as a *stripe of mirror sets* [**25**]. Figure 2.8 shows how RAID 1+0 is implemented. A RAID 1+0 configuration allows tolerance for multiple drive failures, as long as the secondary failures do not occur in the same mirrored pair.

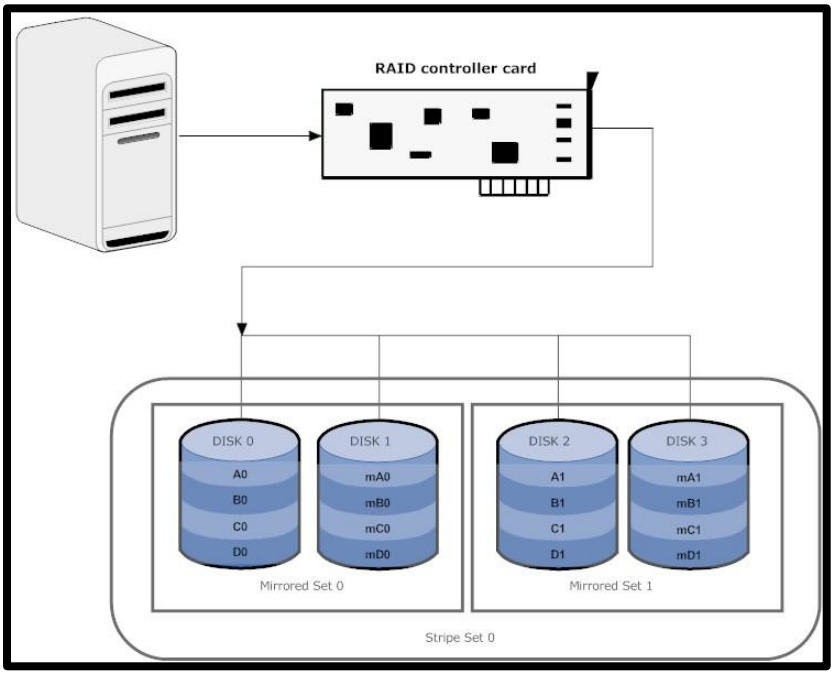


Figure 2.8 A Four Disk RAID 1+0 Configuration

RAID 1+0 is sometimes confused with RAID 0+1 that also combines the attributes of striping and mirroring [**25**]. RAID 0+1, as implemented in Figure 2.9, can be described as a *mirror of stripe sets*. It is considered less reliable, because in the event of a disk failure the array is in essence reduced to a RAID 0 array. Secondary disk failures can only be tolerated if it occurs in the same stripe set as the first.

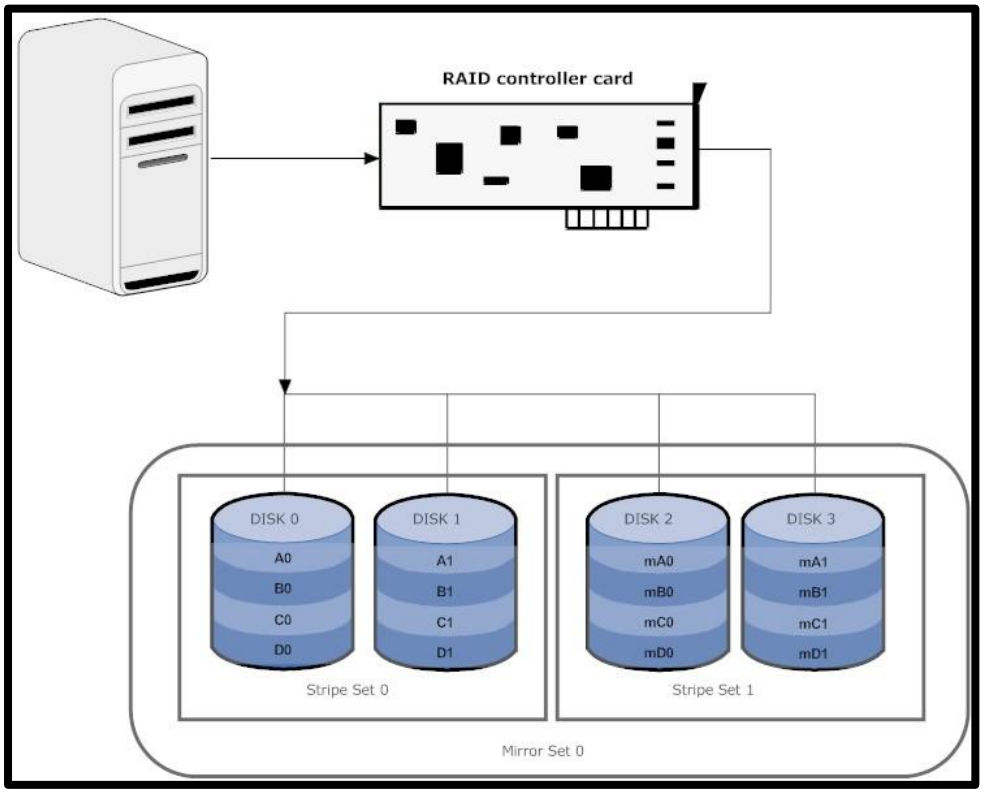


Figure 2.9 A Four Disk RAID 0+1 Configuration

RAID 1+0 and RAID 0+1 configurations are expensive since 50% of the disks are used to ensure redundancy through mirroring [**25**]. RAID 1+0 is most commonly used in four-drive scenarios as it offers better performance than RAID 5 and RAID 6 configurations, whilst maintaining fault tolerance similar to that of a RAID 6 configuration [**27**]. RAID 1+0 and RAID 0+1 both require a minimum of four disks to be implemented [**25**].

2.3.1.5 Other RAID Levels

Several other RAID levels also exist, although only RAID 6 is accepted as one of the standard levels. The others fall either into the category of nested RAID levels or proprietary RAID levels, the latter is normally only available by the company that developed it. Some popular nested RAID levels include RAID 1+0, RAID 0+1, RAID 5+0, RAID 0+5 and more recently RAID 6+0 and RAID 0+6.

RAID 6, or striping with double parity, works on the same principle as RAID 5 (which distributes the parity across all disks), but implements a second parity [**27**]. This adds an additional level of redundancy as the failure of up to two disks can be tolerated, increasing reliability. The disadvantage being that a RAID 6 array suffers from an additional decrease in performance. Benchmarks have shown that there can be a performance drop of up to 30% with a RAID 6 configuration, when compared to a RAID 5 configuration [**27**].

A RAID 6 configuration is also less efficient than a RAID 5 configuration, in terms of storage. A RAID 5 configuration has the effective storage capacity of N-1 drives, where with a RAID 6 configuration it is N-2 drives [**27**]. Note that the effective storage capacity of a RAID 6 configuration increases as the number of disk in the array increases. For instance in a RAID 6 array with four disks the effective storage capacity is 50% of the available space, while a RAID 6 array with eight disks the effective capacity will be 75% of the available space. RAID 6 controllers are in general more expensive than RAID 5 controllers. RAID 6 is advised in scenarios where high redundancy in a large RAID array is required.

2.3.2 Data Backup

A survey on the cost of downtime, by Eagle Rock Alliance, LTD., in 2001 showed just how important data protection and a proper contingency plan are. The survey showed that the survival of 15% of the participating companies would be at risk within the first 24 hours in case of downtime [**28**]. A further 21% of the companies would be at risk in 48 hours and 40% in 72 hours.

RAID's offer increased data protection through redundancy, but still has limitations. RAID offers protection against a certain level of hardware failure, but is useless in cases of software corruption or a natural disaster, such as a fire or earthquake. Thus, it is still necessary to consider further data protection. There are many ways one can achieve an additional level of data protection, each offering their own advantages and disadvantages.

Creating a backup is the traditional way in which one can protect critical data. A backup is a copy of data, files, databases, etc. that can be easily accessed and quickly restored in case of corruption, deletion or catastrophe. Backups are normally stored on a separate storage media and are short-lived; they are overwritten daily, weekly or monthly. It is not to be confused with archiving. An archive is an indefinite retention of inactive or seldom accessed data. Archived data is normally not part of the backup process.

When designing a backup scheme there are several factors you need to consider. These include: the current and future backup requirements, and the characteristics of the backup media.

2.3.2.1 The backup requirements

The backup requirements are determined by the amount of data that needs to be backed up and the number of servers, workstations and other storage devices that needs to be backed up. When defining the backup requirements one can consider the following [29]:

- The data that needs to be backed up. Firstly it is necessary to decide what data needs to be backed up. This can be done by answering the question "What can we afford to lose?" Eliminating unimportant data from the backup process can save storage space, time and money. This entails how the data should be backed up. The different backup types are discussed in section 2.3.2.3.
- The data environment. Secondly consider where all the data is located, the type of data, the estimate backup and restore frequencies, the period the backups must be retained, the time frame that is available to complete the backup, and the required security.

- If possible use backup techniques and technologies that can automate the larger part of the backup process. One can consider using backup software that comes with the backup solution or OS. In some cases it might even be best to outsource the backup process entirely.
- Create a process that will ensure that the backups are completed successfully and are valid. This process includes assigning the responsibility to make sure that the backups are done and monitored for problems; ensuring that there is an adequate time window to perform the backup and understand the time that is required to restore the data; and regularly checking and testing the backup equipment, media and process.
- Properly dispose of old backup media. This involves that old data media is physically destroyed before it is deposed. This will prevent any unauthorised access data retrievals from these mediums.

2.3.2.2 The Backup Media

2.3.2.2.1 Optical Discs

Optical discs have come a long way since the release of the CD-R disc in the 1990s. DVD discs improved on both the capacity and data access rates of CDs. A DVD writer is a standard component in most computers today. Unfortunately the unreliability and limited life of optical discs [30] make them unsuitable for long-term backups. Automation of the backup process, when using optical discs, is very difficult since the discs have to be changed manually.

CDs and DVDs, even when bought in bulk, may seem like an inexpensive backup media, but when the time that is required to backup or restore large amounts of data from these discs are brought into the cost calculation, optical disks become much more expensive [31]. Storage of a large number of optical discs can also be problematic. Optical discs are viable for short-term backups, but are not recommended for long term or very large backups.

2.3.2.2.2 External hard disks

With hard disk capacities rivalling and surpassing the capacities of magnetic tape cassettes, external hard disks are more and more considered as a backup medium. Most external hard disks come in a single disk package with a storage capacity of up to 2 TB. Some models offer multi-disk packages with basic RAID support [32].

External hard disks are more cost effective when compared to optical discs and can be more cost effective than an equivalent tape backup system [33]. It is very easy to automate the backup process when using external hard disks. The reason for this being that once a hard disk is plugged in it can run unattended. No insertion of additional tapes or discs is required and no expensive tape autoloader needs to be purchased.

USB 2.0 external hard disks offer data transfer rates of up to 480 MB/s [34] and can be used with any computer that has a USB port. Newer External hard disks offer External Serial Advanced Technology Attachment (eSATA) connectivity. eSATA offers transfer speeds similar to the SATA II (up to 3 000 MB/s), but is more expensive to implement. Most computers do not have an integrated eSATA port and require either a SATA to eSATA converter header or a dedicated eSATA expansion card [35].

Network attached storage (NAS) hard disk systems are also an option. NAS systems like the Seagate BlackArmor NAS 440 [36] and the WD ShareSpace [37] offer transfer rates up to 1 GB/s through Ethernet. They also offer capacities of several terabytes and more advanced RAID configurations.

2.3.2.2.3 Magnetic tape drives

Magnetic tape disks are one of the traditional backup mediums. Originally tape disks offered capacities above that of hard disks, with the capability to easily spread the data across multiple tapes. The latest generation of cassettes and drives, like the TS1130 tape drive [38] and IBM 3592 JB/JX Tape Cartridges [39] from IBM, offer up to 1 TB of uncompressed storage.

When combined with autoloaders or tape libraries capacities of several petabytes are possible. Tape drives also offer much higher sustained transfer speeds, up to 160 MB/s [38], when compared to other backup mediums. Unfortunately a tape drive solution is expensive when compared to hard disks. A tape drive solution, with an autoloader and two sets of medium, can cost up to five times more than an equivalent external hard disk system with two hard disks [33].

2.3.2.3 Backup schemes

A backup scheme defines the parameters of the backup process. It defines what media is used to store the backup, how often the backup is performed and what type of backup is done. Most modern backup software allows you to set-up a backup scheme, allowing you to specify these parameters.

When designing a backup scheme it is important to select the correct media to store the backup. This can normally be achieved with just one type of media, although in some cases the best solution may be to use a combination of media to satisfy the specific backup needs. Backups have a limited lifetime, as they are periodically overwritten. It is thus important to determine how long a specific backup should be retained.

There are also several different types of backups that can be performed. The most popular backup types include: full backup, incremental and differential backups. Each of the backup types will now be discussed.

- A full backup entails making a copy of the system files, software files and data files [40]. With a full backup an entire system can be restored in case disaster strikes. It is considered good practice to perform a full backup either weekly, bi-weekly or on a monthly basis.
- A partial backup will copy only the file that has changed or added since the last backup [40]. It is normally performed on a daily basis and combined with a full backup. Following is a description of the two main types of partial backups:

- Incremental backups copy only the files that have been changed or added since the last full or partial backup [40]. This means that fewer files need to be copied with each backup session, in effect reducing the time needed to perform the backup. A complete system restore can be slowed if there are many incremental backups.
- Differential backups copy only the files that have changed since the last full backup [40]. This means that backups will take progressively more time to complete, but a full system restore will be faster. When using a differential backup strategy it is important to make a full backup more regularly.

2.4 Enhancement of the software applications

2.4.1 The Data Acquisition and Logging Client (DALC)

The current DALC, although operational, still has many limitations. Currently to log the data of several OPC items one must manually configure the information concerning the OPC server and then manually add all the items of that server. With the new version, functionality will be implemented to browse for OPC servers and their items.

The current DALC is designed to connect to a local database and to connect to both local and remote OPC servers. This will be changed so that it will be able to connect to either a local or remote database, making it much more flexible.

2.4.2 Automation of administration tasks

Originally it was considered to create an application that would automate some of the administrative tasks. These administrative tasks include: archiving older data, regularly running OPTIMIZE and ANALYZE on heavily used tables, backing up the database schemas and data.

It proved unnecessary as the backup can be automated using Navicat [41], while the other tasks can be automated by using a new feature, called EVENTS.

2.4.2.1 MySQL's EVENTS

Since the release of MySQL 5.1.6, events are supported. An event is called a database object, whose instructions are executed at one or more regular time intervals. Events should not be confused with triggers. Triggers are similar to events, but are executed in response to an event on a specific table [**2**], e.g. before an INSERT is executed. Events allow a DBA to automate certain routine tasks, saving considerable amounts of time.

An event's instructions can consist of one or more SQL statements. These statements can either be normal SQL or compound statements. The SQL statements that can be used in events are limited to those that are allowed in stored procedures [**2**].

```
CREATE EVENT simple_event
ON SCHEDULE EVERY 1 SECOND
DO INSERT INTO test.t VALUES (CURRENT_TIMESTAMP);
```

Figure 2.10 Creating a simple event and starting the event scheduler

An event can execute once or on a recurring schedule. A recurring event schedule starts as soon as it is created and continues indefinitely. For more control you can assign a starting and/or ending date and time [**2**]. A standard set of SQL statements are included to create, modify and delete events. Figure 2.10 shows how to create a basic event that executes every second. Using the ALTER EVENT SQL statement you can set most a scheduled event's properties [**2**].

Before events can run or be executed the event scheduler thread, referred to as the Event Scheduler, needs to be started. The Event Scheduler can have one of three states. These include OFF, ON and DISABLED. The default is OFF. In this state the Event Scheduler is stopped and does not appear in MySQL's process list. To start the Event Scheduler its state needs to be set to ON. When the Event Scheduler is started it will execute all evens accordingly.

The ON/OFF states can be set dynamically at runtime [2]. The third state, DISABLED, can only be set at start-up. DISABLED stops the Event Scheduler and prevents it from being started at runtime. Figure 2.10 shows how to create a very basic event and enable the Event Scheduler.

To create, modify or delete events a user requires the EVENT privilege [2]. To start or stop the Event Scheduler the SUPER privilege is required. Users can also include statements that they do not have the privilege to use in an event. The event will be created successfully, but its execution will fail.

2.5 Summary

This chapter reviewed the relevant technologies and techniques that can be used to optimise the data-logging system. Attention was given to the following:

- The optimising of different types of SQL statements that result in performance increases.
- Several features of MySQL that would assist with the redesign of the database schema. These included supported data types, the storage engines and table and index partitioning.
- Evaluation of several MySQL system variables that influence the performance of the server.
- Investigating RAID and backup technologies that can be used to improve the performance and reliability of the database server.
- Reviewing the limitations of the current DALC application.

The following chapter will focus on how these will be evaluated and ultimately implemented.

3 Methods and Techniques

The study is focused on improving the reliability and performance of the OPC data acquisition and logging system as a whole. In most real world scenarios it is difficult to find the right balance between performance and reliability, often sacrificing one for the other. This is normal due to limited funds and/or personnel assigned to the information technology (IT) department of a company.

To achieve the correct balance between performance and reliability, the optimisations to the system will be divided into three parts: the hardware, the DBMS and DALC application. Each of these areas as well as the methods used to improve and measure the changes are discussed in the subsequent sections.

3.1 Hardware Optimisations

The first optimisation to the data-logging system will be hardware orientated. The hardware is the foundation of the system. The focus will be to improve the performance and reliability of the database server. When defining the hardware requirements of the system it is always important to consider the current and future operational parameters. In the case of a database server it is important to consider the size of the database, and the number of users that will be allowed to connect at once. The proposed hardware optimisations are discussed below.

3.1.1 Overview of the Proposed Database, Application and Web Servers

As mentioned previously, in the current system, the database server also acts as an application server. This means that its resources are shared between the DBMS and all the applications that are executed on the server. It also has very limited resources in terms of memory, physical storage capacity and processing power.

To address these problems three new dedicated servers will be implemented to replace the current database/application server and web server. One will be setup as a database server and another as an application server. The old web server will also be replaced to match the specification of the application server.

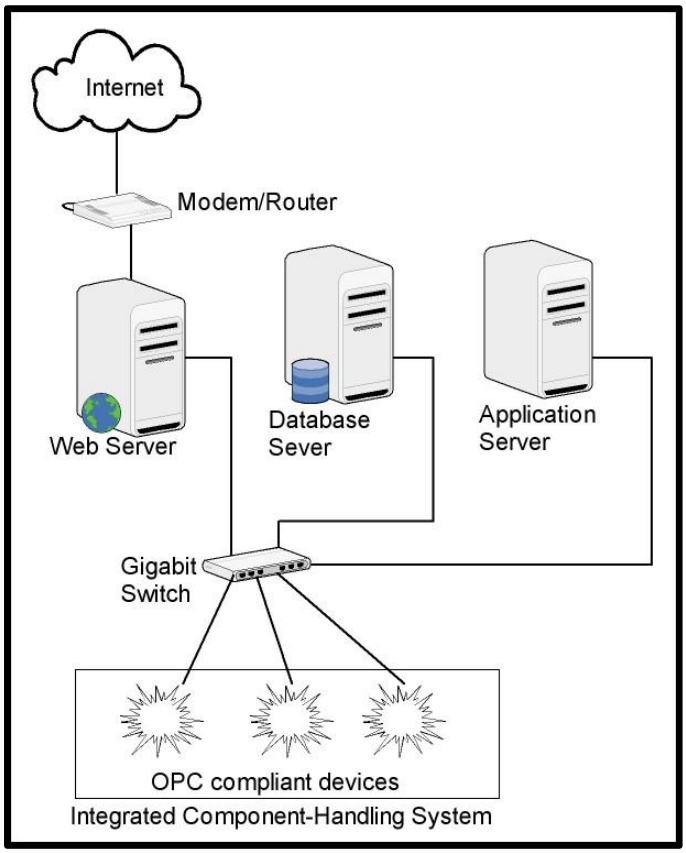


Figure 3.1 Network Diagram of the Proposed System

When selecting the hardware components for the new servers, the following will be taken into consideration:

- The central processing unit (CPU): MySQL is a multi threaded DBMS and can scale well across multiple CPUs or cores. This makes multi-core CPUs a good option for the new database server; especially since most desktop platform motherboards only support one processor. The Intel Q6000 series processors [**42**] are an ideal solution. It offers four processor cores, each capable of executing a process thread independently. They also support Intel's Virtualisation Technology (VT), which will be of great value to the application server, as it will host several virtual machines.
- Memory: Memory capacity is of high importance, especially to the database server. More memory will allow the DBMS to cache more data and indexes in memory; this will also prevent paging and increase performance. With the current desktop computer technology and 64-bit OSs, such as the Windows Server 2008, a system with 4 GB or more memory is viable.
- RAID controller: A dedicated RAID controller card will be added to the database server. This RAID controller card will be used to create and manage the RAID configuration that will store the data files of the MySQL database. The Adaptec 3405 RAID controller card [**43**] is an excellent candidate. It has support for both Serial ATA (SATA) and Serial Attached SCSI (SAS) disks. In addition it allows up to four hard disks to be connected in a RAID array. This makes RAID 5 and 1+0 configurations very viable, as both should produce increased reliability and I/O rates.
- The Motherboard (MOBO): It is the heart of any computer system. Selecting a MOBO that will support all the hardware is vital. Additional features that are also required include: integrated display, Gigabit networking capabilities, integrated RAID controller. Desktop platform MOBOs normally have a limited RAM capacity, when compared to the server equivalent. It is therefore vital to select a MOBO with support for at least 8 GB of memory.
- Primary storage: Each server will be equipped with three hard disks. These disks will be used with the motherboards integrated RAID controller to create a RAID configuration that will be used as the system disk. Four additional disks will be added to the database server. These disks will be used in conjunction with the dedicated RAID controller card, to configure the RAID configuration that will store the data files of the database.

3.1.2 **RAID Configurations and Benchmark Methodology**

As mentioned, the Adaptec 3405 RAID controller has a drive capacity of up to four hard disks. It has support for RAID levels 0, 1, 5, 6, 1+0, Just a Bunch Of Disks (JBOD) and several other non-standard RAID levels [**43**]. RAID offers increased I/O rates and reliability over a single hard disk, with the exception of RAID 0 that actually has reduced reliability.

RAID levels 5 and 1+0 are of interest. RAID 1+0 combines the advantages of striping and mirroring, but the effective storage space is only 50% of the total capacity. In a four-disk array it can tolerate up to two disk failures. RAID 5 offers redundancy through parity and can tolerate one disk failure. The effective storage capacity of a RAID 5 configuration can be calculated using equation (3:1). Note that D_{cap} represents the effective storage capacity, and N is the number of disk in the array.

$$D_{cap} = \frac{N-1}{N} \times 100\% \qquad (3:1)$$

The four disks connected to Adaptec 3405 will be configured in the following RAID levels:

- RAID 0
- RAID 5
- RAID 1+0

Each of these configurations will be benchmarked by using Sisoftware Sandra Lite [**44**]. A control benchmark will be done using a single hard disk. The RAID 0 benchmark is included as a reference, as it should theoretically provide the best write performance. The three disks will be used in conjunction with the MOBO's integrated RAID controller. They will be used to create a RAID 5 configuration that will store the OS and other applications. Here the performance is of little importance, with the focus being on reliability. RAID 1 and 5 configurations will be evaluated over a period over time, on all three servers.

3.1.3 Design and Implementation of a Backup Scheme

The importance of a proper backup scheme has already been discussed in section 2.3.2. The following sections will discuss the steps that need to be taken to design and implement the backup scheme. These steps include:

1) Identify the backup requirements.
2) Selecting the backup media.
3) Planning the backup schedule.
4) Implementing of the backup scheme.

The details of each step are discussed in the following sections.

3.1.3.1 Identifying the Backup Requirements

The most important data that must be backed up, are the data files of the database. These files will change constantly as new data is logged. These files will have to be backed up regularly.

A full system backup of each server is also recommended. Restoring a system backup can be significantly faster by reinstalling the server OS and all the relevant applications and drivers. Also since there will be little changes to the system disk after the initial installation and configuration, these backups do not need to be performed as regularly as the database backups.

The data that needs to be backed up, is distributed across the three severs. The servers will be interconnected via a high speed Gigabit Ethernet network. This makes a NAS solution a very viable option as central backup media. Also since all the servers will have USB 2.0 connectivity, external hard disks are a good option as an offsite backup media.

The backup routine needs to be highly automated with the minimum interaction needed. Also the administrators need to be informed whether the backup succeeded or not.

3.1.3.2 Selecting the Backup Media

The primary backup media will be a combination of internal and external hard disks. CD-R and DVD-R will be used as a secondary media, for backing up specific data that only need to be available for a few days (short-term backups). The different backup media include:

- A Western Digital ShareSpace NAS system.
- Three 500 GB external hard disks (one for each server).
- Six internal hard disks (two per server).

The ShareSpace NAS system has a total of four 1 TB hard disks, and has RAID capabilities. It also supports Gigabit Ethernet. A RAID 5 configuration will be used (RAID 1+0 is not supported), resulting in 3 TB of effective storage space. However it will offer improved reliability over a JBOD configuration.

The external hard disks can interface with the servers through either eSATA or USB. They contain a single hard disk, trading redundancy for mobility. Each external disk has a capacity of 500 GB.

It was decided to abandon the RAID array for the OS of each server (see Chapter 4 for the explanation), and rather to incorporate the two extra disks per server into the backup strategy. Each disk has a capacity of 250 GB.

3.1.3.3 Scheduling the Backups

In the following section the external hard disks will be referred to as EXT0, EXT1 and EXT2, the Share Space NAS system will be referred to as NAS0, and the two internal disks of each server will be referred to as INT0 and INT1. The backup schedule will make use of a daily, weekly and monthly backup routine. Here follows an explanation of the backup schedule, as depicted in Figure 3.2. Note that this schedule assumes operational hours from 7 a.m. to 6 p.m., Monday to Friday.

The daily backups will be interleaved between INT0 and INT1; with INT0 being used for the backups of Mondays, Wednesdays, Fridays and Sundays, and INT1 for Tuesdays, Thursdays and Saturdays. On the first day, assumed a Monday, a full backup will be made to INT0. On the second day, assumed a Tuesday, a full backup will be made to INT1. After the initial full backups, daily incremental backups will be made to INT0 and INT1 on their appointed dates. Daily backups will be performed at 11 p.m. daily.

The weekly backups will be made to the external disk, EXT0, EXT1 or EXT2, assigned to the specific server. As with the daily backups, an initial full backup will be made following a series of incremental backups. Weekly backups will be performed on Fridays at 4 p.m. The external disks will be kept offsite during the week.

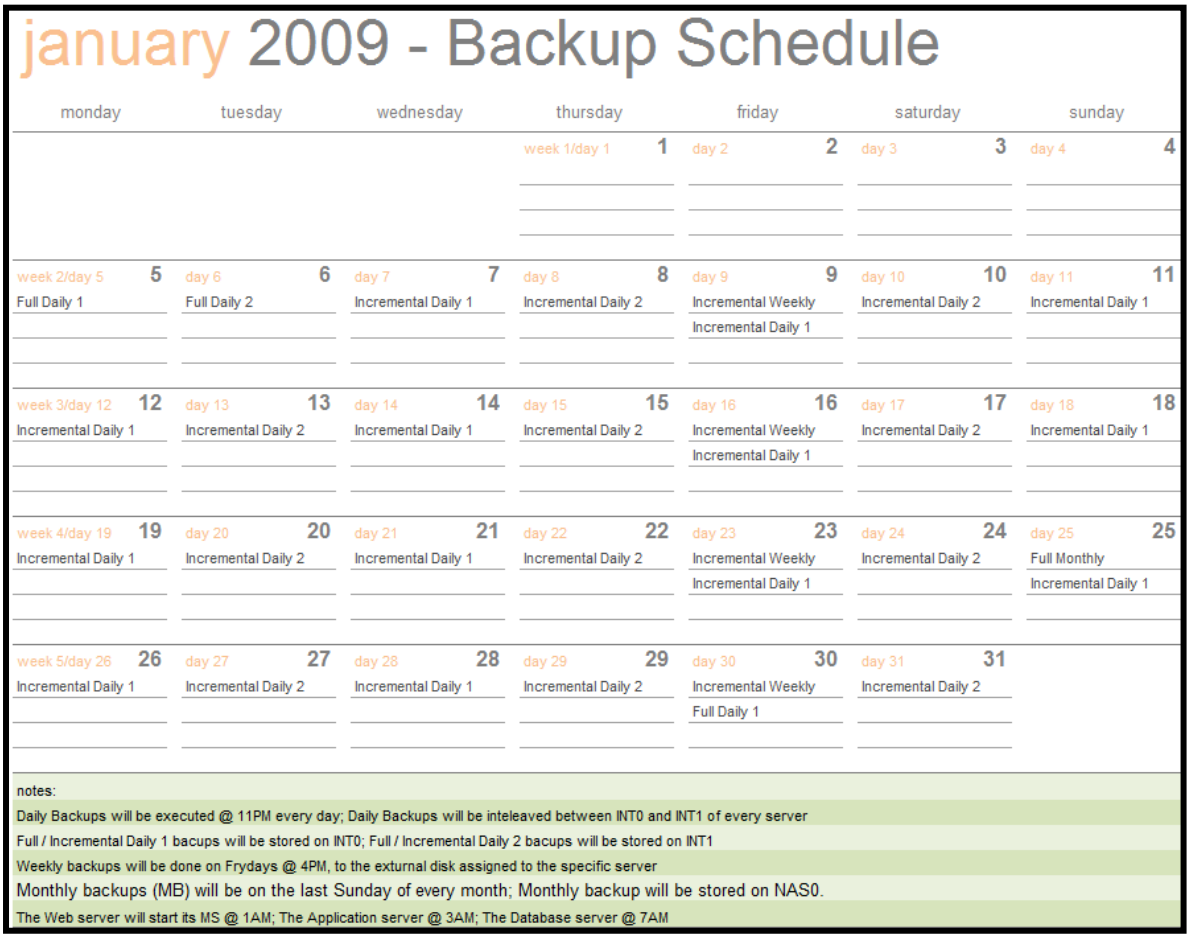


Figure 3.2 Backup Schedule – for January 2009

The monthly backups will be made to NAS0. Due to limitations of the backup software only full backups can be made to network locations. The monthly backups will be performed on the last Sunday of every month. The web server will be the first to do its monthly backup, as it has the least amount of data. It will start at 1 a.m. and will have a three-hour backup window. The application server will be the second to do its monthly backup. It will start at 4 a.m. and have a three-hour backup window. The database server will be the last to do its monthly update, seeing that it has the most data to backup and will take the most time to perform the backup. It will start at 7 a.m. with very large backup window of 16 hours (it needs to be done before the scheduled daily backup starts). As illustrated in Figure 3.2, a new full backup will be made after every 14th backup (1 full backup + 13 incremental backups). This applies to both daily and weekly backups.

3.1.3.4 Implementing the Backup Strategy

The system backup will be implemented with command-line scripts, which will utilise the WBADMIN command-line utility [45] included with the OS. These scripts will be scheduled for execution with the Task Scheduler. The database data will be backed up in two different ways. The full backups will be scheduled with Navicat [46] to execute on a monthly base, while the daily and weekly backups will be integrated into the system backups. This will be achieved by making use of MySQL's binary log (binlog) files, which are created on the system disk. The following two sections will discuss how the backups will be scheduled.

3.1.3.4.1 Scheduling the System Backups

The Task Scheduler will be configured to execute the system backup scripts at the appointed intervals. Figure 3.3 illustrates the daily backup script that will be used as template for all the other system backup scripts.

```
@ECHO OFF
REM +-------------------------------------------------------------------------------------------+
REM | Command              |     Decsription                                                     |
REM |----------------------+-------------------------------------------------------------------|
REM | WBADMIN START BACKUP|     will start the backup process                                   |
REM +----------------------+-------------------------------------------------------------------|
REM | -backupTarget:       |     speecifies where the backup will be stored                     |
REM +----------------------+-------------------------------------------------------------------|
REM | -allCritical         |     automatically includes all critical volumes. It is useful      |
REM |                      |     for making full system backups                                 |
REM +----------------------+-------------------------------------------------------------------|
REM | -vssFull             |     Makes backups, by using the Volume Shadow copy Service(VSS)    |
REM |                      |     VSS tracks changes to files and is used as the basis for       |
REM |                      |     the incremental backups                                        |
REM +----------------------+-------------------------------------------------------------------|
REM | -quiet               |     runs the backup with no prompts to the user                    |
REM +----------------------+-------------------------------------------------------------------|
REM | >>                   |     Out redirect ouput to a specified file                         |
REM +-------------------------------------------------------------------------------------------+

REM get the year, month and day components of the current date
SET /A yy = %date:~0,4%
SET /A mm = %date:~5,2%
SET /A dd = %date:~8,2%

REM recombine the date componentes for the file name
REM Format: DD.MM.YYYY
SET NAME=%dd%.%mm%.%yy%

REM Execute the backup, create a log file
WBADMIN START BACKUP -backupTarget:E: -allCritical -vssFull -quiet >> C:\backuplogs\%NAME%-DailyBackup.log

REM Delete the CurrentBackupLog, Copy the latest backup log and rename the copy to CurrentBackupLog
REM CurrentBackupLog will be mailed to the person in charge of the backups
DEL C:\backuplog\CurrentBackupLog.log
COPY C:\backuplogs\%NAME%-DailyBackup.log C:\backuplog\CurrentBackupLog.log
```

Figure 3.3 Backup Script Template

The first time this script is executed a full backup of the system disk will be made. Thereafter incremental backups will be made. As stated previously a new full backup will be made after every 14th backup; this is done automatically by the Windows Server Backup system. After a script has been executed the backup log file, which is automatically generated by the backup script, will be emailed to the relevant administrators.

A new task must be created in the Task Scheduler for each script. The Task Scheduler is integrated into the Microsoft Management Console as a snap-in, and is located under *Configuration* in the Console tree. Each task will be given a unique, but descriptive name. Note that the task is configured to run with highest privilege, as this is required to perform the backup [47]. Scheduled tasks make use of the concept of triggers and actions. Triggers specify the conditions that will cause the task to be executed. Actions specify the actions that should be taken when the trigger conditions are met. A task can have multiple triggers and/or actions.

The triggers for the system backups will be set on a repeating schedule. Figure 3.4 illustrates how the trigger for the first of the two daily backups will be configured. Because a bi-daily schedule is used for the daily system backups, the trigger will be configured to execute on certain days and repeat on a weekly base.

After the trigger for the task has been created, an action needs to be specified. The first action will be created, as in Figure 3.5, to execute the batch file, containing the script in Figure 3.3. Similarly a second action will be defined to send the backup log file, which is generated by the backup script, to the relevant administrators.

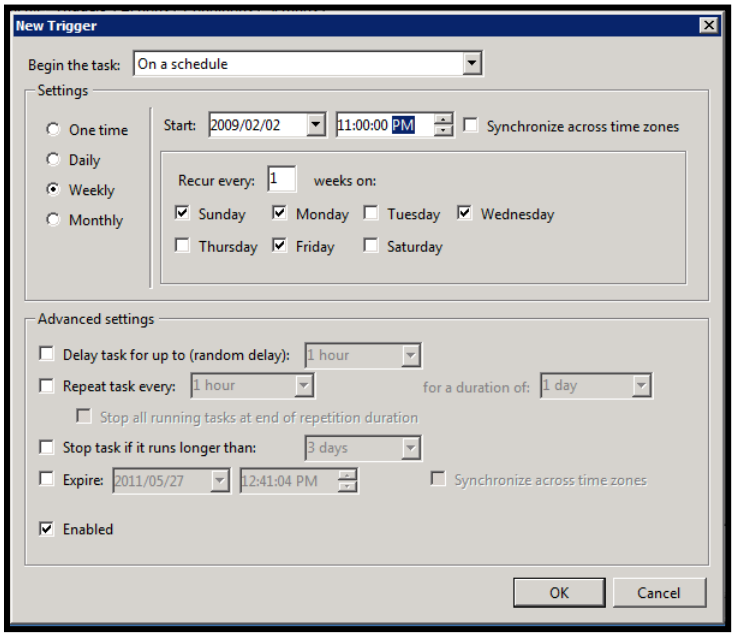


Figure 3.4 The New Trigger Dialogue

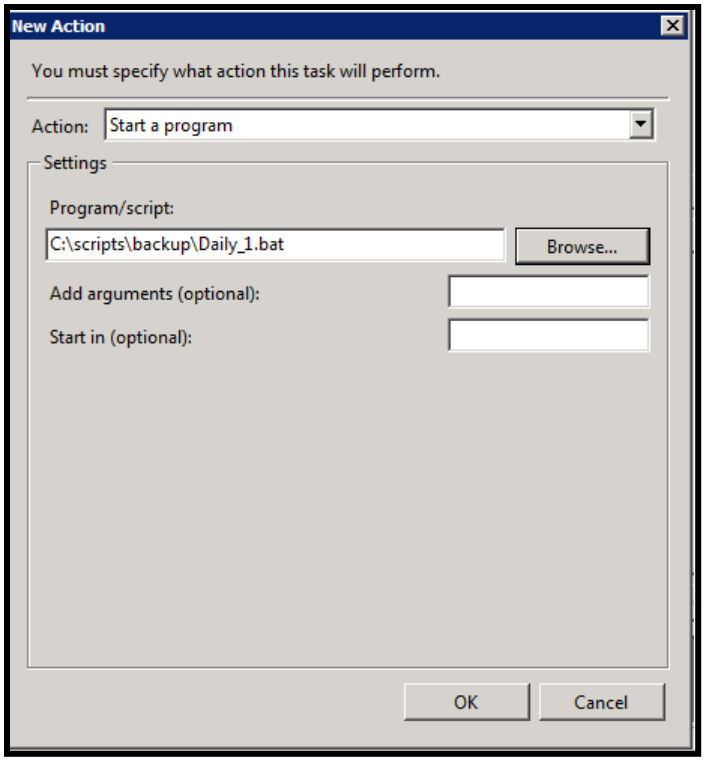


Figure 3.5 The New Action Dialogue configured to execute the backup script

3.1.3.4.2 Scheduling the database backups

The daily and weekly backups of the database will automatically be included in those of the system disk. The reason for this is that the binlog files, which are used as incremental backups of the database, are created on the system disk. The binlog files track all transactions that modify or create database objects and data. A new binlog file is created when the logs are flushed, the MySQL server is restarted or when the *max_binlog_size* has been reached or exceeded. The binlogs will be flushed on a daily basis and reset on a monthly basis.

As previously mentioned, the full backups of the database will be scheduled and done with Navicat. Before the database's full backup can be scheduled, a backup job must first be created for each of the schemas that must be backed up. This can be done from the Backup pane.

The backup jobs will be configured to lock the tables and compress the backups, as in Figure 3.6. The table lock will prevent any data to be modified while the backup is in progress. Compressing the backups may increase the time required to backup or restore the database, but will greatly reduce the space required to store the backups. Each backup job will also be given a unique, but descriptive name. Note that Navicat stores the backup in the *Setting Save Path* [48], in this case it is configured to a directory on NAS0.

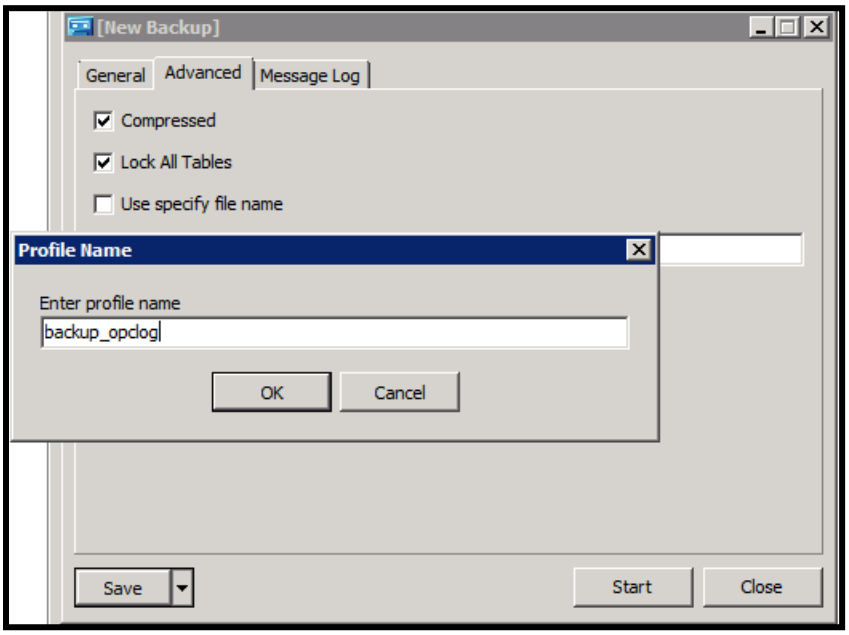


Figure 3.6 New Backup Dialog in Navicat

A batch job will be created to sequentially execute the backup jobs, illustrated in Figure 3.7. This batch job will then be scheduled to execute with the monthly system backups, on the last Sunday of each month. The database backup will be set to start three hours after the system backup. Before the backup task is performed the binlogs will be flushed and reset by a MySQL event. Several versions of the monthly backup will be kept.

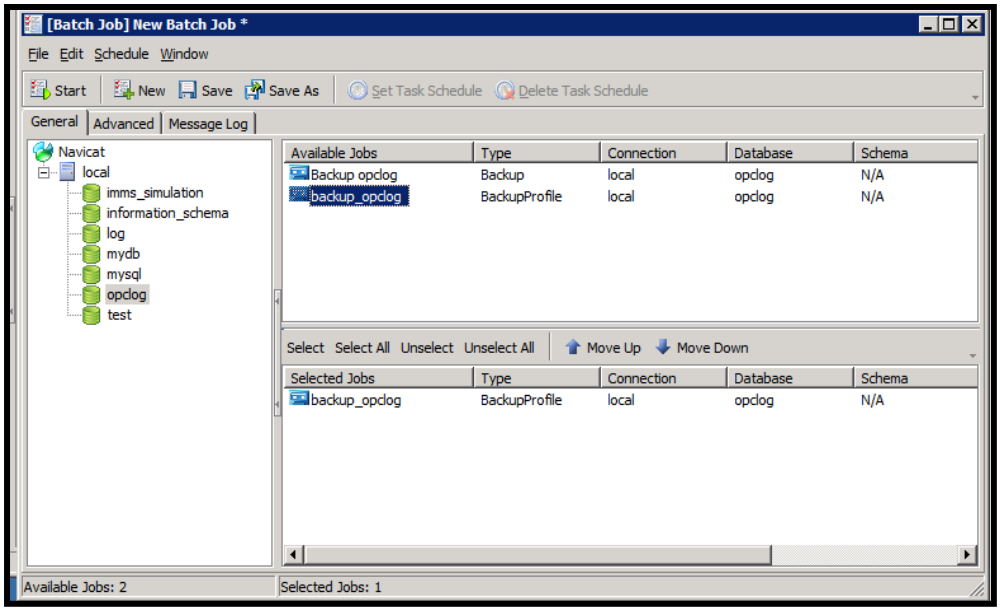


Figure 3.7 Creating a New Batch Job in Navicat

The backups that are made with WBADMIN can easily be restored by using either the Windows Server backup or the Windows Recovery Environment that can be launched from the Windows Setup disc. The full database backups can also be quickly restored with Navicat. The process of doing a point-in-time recovery from the binlogs is well documented in the MySQL reference manual.

3.1.4 Additional Reliability Improvements

To further improve the reliability of the server and security of the data, two more optimisations are necessary. The first optimisation to the system is the addition of an uninterruptible power supply (UPS) and the configuration of its power management software. The second optimisation involves properly configuring the user rights of the database users.

3.1.4.1 Power Management Configuration

A 3 KVA UPS will be added to the system to protect the servers and other devices, such as external hard disks, from power surges and failures. UPSMON, a software package that is distributed with the UPS, will be used to monitor the power status of the mains, as well as the battery levels of the UPS. UPSMON can be configured to take predetermined actions in case of a prolonged power failure.

UPSMON will be configured as illustrated in Figure 3.8. *Power Failure Windows Display Delay* determines the length of the delay before the power failure message is displayed. After the warning message is displayed, UPSMON will wait for a period of time before starting the shutdown process; the period it determined by the *Power Failure Windows Shutdown Delay* variable. If the AC power is restored during this period the shutdown process will be cancelled and normal operation will continue.

The UPS Shutdown Delay starts directly after the windows shutdown delay ends. As its name suggests it is the time the UPS waits before it shuts down. This time must be long enough to allow all the devices connected to the UPS to power down. The shutdown process of the computer that is running UPSMON, the database server, can be delayed shortly to execute any Command-Line script before the computer shuts down; the Time to Execute Command file specifies this delay time. UPSMON will be configured to execute a Command-Line script to shutdown the Web and Application servers.

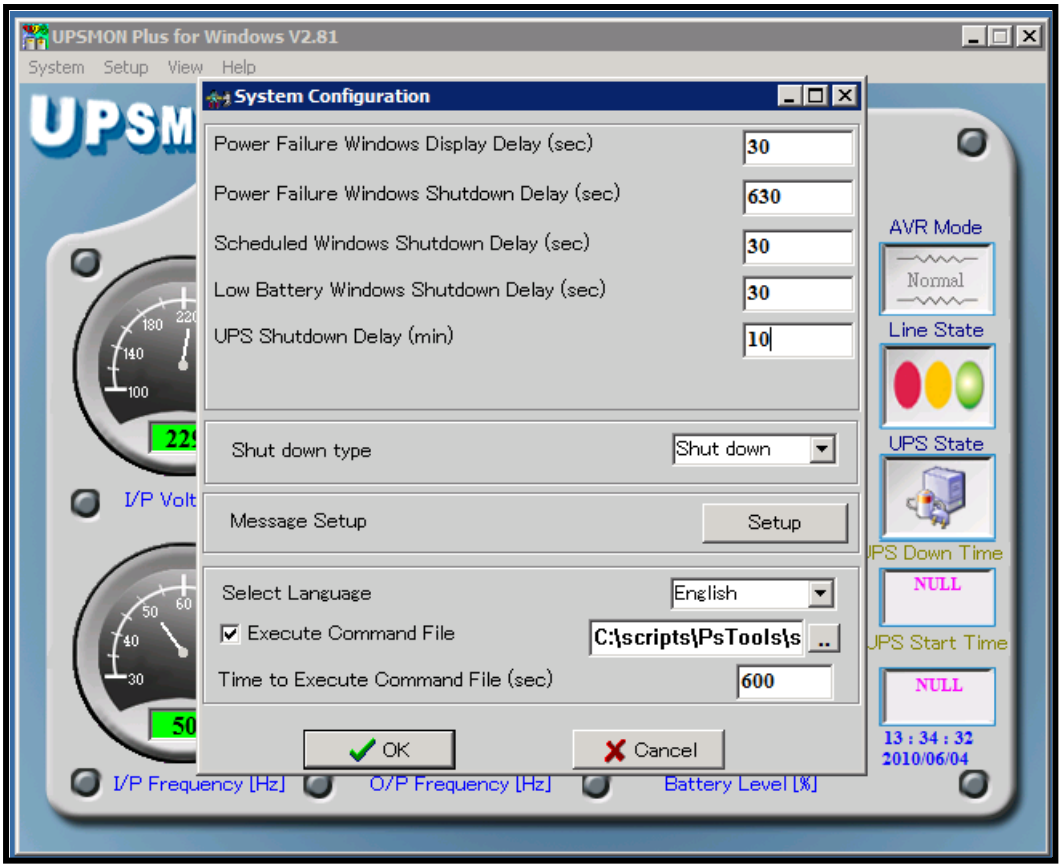


Figure 3.8 UPSMON Configuration

After the Command file delay time has passed the shutdown process will continue. Note that the Time to Execute Command does not influence the UPS Shutdown Delay. The former must be shorter than the latter to allow the computer to successfully shutdown before the UPS shuts down. If the AC power is restored during the UPS Shutdown Delay, the shutdown process will complete and the start-up process will commence immediately thereafter. Once the UPS has shut down it will wait for the AC power to be restored before it powers up again. The timeline in Figure 3.9 illustrates the flow of this process.

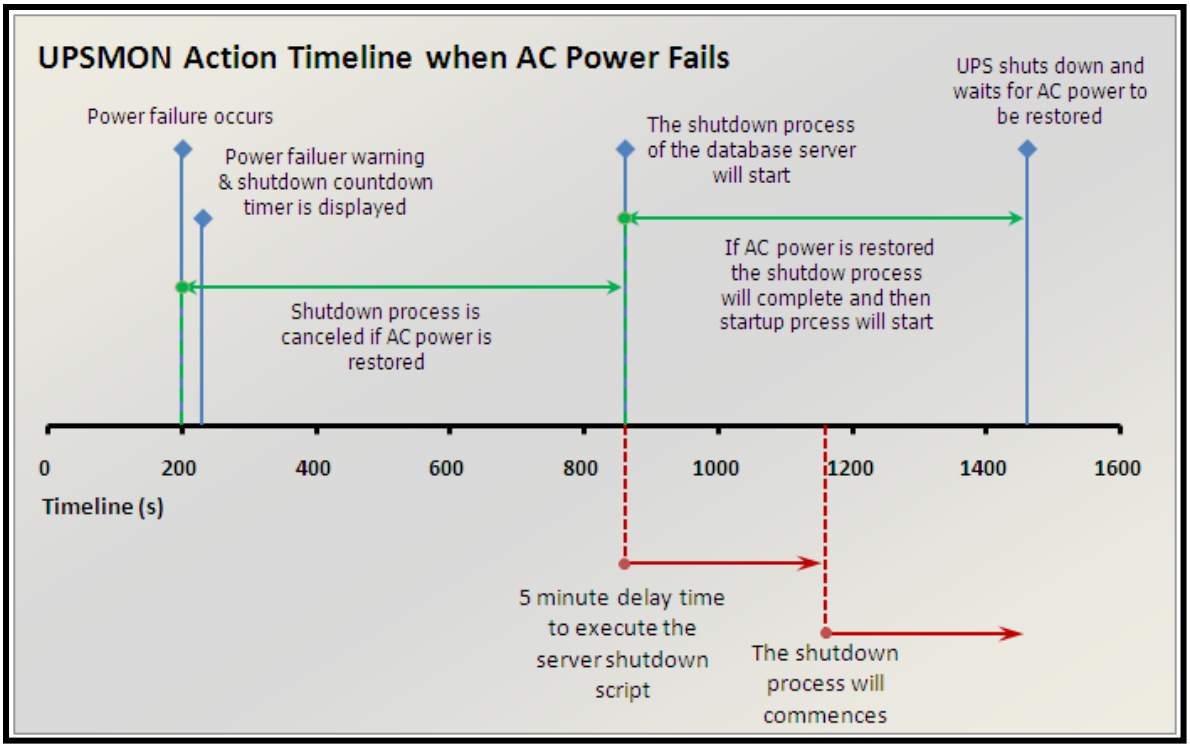


Figure 3.9 Timeline of the Actions Taken by UPSMON When AC Power Fails [49]

The Command-line script responsible for safely shutting down the Web and Application servers was created using PsShutdown. PsShutdown is part of the PsTools' [**50**] Command-line tools suite, which allows you to manage both the local and a remote server through the command prompt. PsShutdown allows you to shutdown or restart the local or remote server.

The start-up process will be fully automated. This is possible because the UPS automatically starts-up once the AC power is restored. The Power On setting in the BIOS of the servers determines how the server automatically resumes (starts-up) after a loss of power. Configuring it to Restore on AC Power Loss will cause the servers to start-up in the event that AC power fails and is later restored. Once the UPS has come back online it will automatically restore power to the servers, triggering this event.

3.1.4.2 User Rights Configuration

Properly configuring the access rights of each of the database user accounts will greatly increase the security of the server and data. Dedicated user accounts, with limited rights, will be created for all users and applications. The rights will be assigned according to each user or application's needs. The default administrator account (root) of the database will also be disabled and a new administrator account will be created. The user accounts will be created using Navicat.

3.2 Optimisations of the Database Schema and DBMS

There are several optimisations that will be done to improve the performance and reliability of the database. In terms of performance, the focus will be on improving the write speeds of the server. The first optimisation will be a complete redesign of the database schema. The second optimisation will focus on the performance of INSERT and SELECT queries. The third optimisation will be to configure the MySQL server (*mysqld*) to properly make use of the available resources.

3.2.1 Redesign of the Database Schema

The database schema will be completely redesigned. Several events will also be implemented to automate some routine administrative tasks. Following is a discussion on each of the schema optimisations.

3.2.1.1 Selecting the Primary Storage Engine

The first step in designing the new database schema will be to select the primary storage engine. While the performance of the storage engine is certainly important, the features it supports are equally important. The engines that will be considered for the primary engine are MyISAM and InnoDB. Archive and Memory are excluded as options due to their very unique features; the former is specialised for storing data that is rarely accessed and the latter for temporary in-memory tables or caching data that is frequently accessed. The results from the benchmarks in section 3.2.2.1 will be used as a write performance indicator for the storage engines. Important features that will be considered include: index support, lock granularity, transaction support, foreign key support and possible optimisations to the storage engine.

3.2.1.2 Table design and relations

After selecting the primary storage engine, the data-logging database schema will be redesigned. The data that needs to be stored, as well as how they relate, will be considered. One of the main goals is to keep the row length of the data that will be stored in the OPC log table to a minimum to ensure fast writes, seeing that larger rows will take longer to write to disk.

3.2.1.3 Selecting the data types for the table columns

After the table layout and relations have been determined, it is necessary to select the proper data types for each of the columns in the tables. The focus will be on using the smallest possible data types, as this will ensure a smaller row size and overall a smaller database. Smaller row sizes will help to minimise data retrieval times, thereby increasing query performance.

3.2.1.4 Implementing the basic indexes

With the tables designs completed, relationships identified and data types selected, basic indexes will be implemented based on the table relations. Indexes will be created on columns identified as primary and foreign keys; this is actually a requirement of the InnoDB storage engine when implementing a primary key – foreign key relation. This will also serve as a basic optimisation for SELECT queries as these columns are likely to be used when table joins are performed.

3.2.1.5 Events

As mentioned, several events will also be designed and implemented to automate some administrative tasks. The first will flush the binlogs and tables on a daily basis, just before the daily backup starts. The second event will be a bit more complex, but will only execute once a month.

The second event will perform several tasks. First it will copy all the OPC logs older than a specified period, three months in this case, to the log archive table. Secondly it will delete those logs from the active log table. Next it will execute the OPTIMIZE TABLE statement against the active log table to defragment it and reclaim the space of the deleted records. The fourth task will be to update the key (index) distribution statistics of the entire database by means of the ANALIZE TABLE statement. The last task will be to flush the tables and reset the binlogs. This event will be scheduled to execute well before the monthly backup is scheduled as it might require some time to complete; it will however not influence the backup process of the other servers.

3.2.2 SQL optimisations

The two main types of queries that will be executed against the database will be INSERT and SELECT queries. The INSERT queries that will be executed against the OPC logging database will almost exclusively be done by the DALC. SELECT queries will mostly be executed by users and other applications. Following is an explanation of the optimisation process for both INSERT and SELECT queries.

The performance of all the SQL queries will be measured with *mysqlslap* [**2**]. *Mysqlslap* is a multi-threaded command-line utility, developed by MySQL AB, which can measure query performance at different user loads.

3.2.2.1 Performance comparison of single record and bulk record INSERT statements

The DALC will collect a large amount of data, from several OPC servers, and write it to the database at one-second intervals. This makes bulk record inserts a viable option. As mentioned in Chapter 2, bulk inserts can be up to ten times faster than multiple single inserts statements. To investigate this performance difference, and further evaluate the viability of using bulk inserts in the DALC, several benchmarks will be performed.

The benchmarks will measure the performance difference between a series of single inserts and the equivalent bulk insert. The benchmarks will be repeated several times, each time with a different amount of records. The benchmarks will be performed on several identical tables based on the InnoDB, MyISAM and Archive storage engines.

Figure 3.10 illustrates the SQL code for the table that will be used for the benchmark. After each benchmark the table will be truncated and its storage engine changed.

```
#Delete the table and its data if it exists
DROP TABLE IF EXISTS inserttest;
CREATE TABLE inserttest (
        int_col INT(11) DEFAULT NULL,
        varchar_col VARCHAR(20) DEFAULT NULL,
        char_col CHAR(10)DEFAULT NULL,
        DateTime_col DATETIME DEFAULT NULL
```

Figure 3.10 The Script for Creating the Table Used in the INSERT benchmarks

The template of the *mysqlslap* script that will be used for the benchmarks is illustrated in Figure A-1. The script has four major steps:

1. Execute the benchmark of the specified stored procedure.
2. Write the results to a file.
3. Change the storage engine that the table is based on.
4. Truncate the table.

These steps are repeated for the Archive, MyISAM and InnoDB storage engines. The structure of the stored procedures (singleinsert10 and bulkinsert10) can be seen in Figure A-2 and Figure A-3. Similar stored procedures to insert 100, 1 000, and 10 000 records respectively will also be created. The results of the benchmarks will be written to a log file that will later be processed to determine the performance difference between single and bulk record inserts.

3.2.2.2 SELECT Optimisation Process

It is impossible to determine all the different SELECT queries that will be run against the OPC logging database. Instead one plausible query will be used to illustrate the optimisation process. This process can be repeated to improve the performance of most SELECT queries.

The query will be executed against a test database based on the final design of the OPC logging database. The database will be populated with test data according to the following specifications:

- 10 OPC servers.
- 1 000 OPC items randomly assigned to the OPC servers.
- 10 000 records of randomly generated test data in the "active" log table, with timestamps ranging over three months.
- Approximately 6 000 000 record in the archived log table, with timestamps ranging over two years.

The tables of the database will be optimised, with the OPTIMIZE TABLE command, before the benchmark and optimisation process begins and after every structural change, like addition indexes.

A *mysqlslap* script will be created and executed to measure the execution times of the query. Figure B-1 illustrates the template of the *mysqlslap* script that will be used. Following the *mysqlslap* benchmark the query will be executed normally using the EXPLAIN or EXPLAIN PARTITIONS statement, to gain some understanding of the query execution plan used by MySQL. EXPLAIN PARTITIONS will return the same information as EXPLAIN; only adding information concerning the partitions used by the query. Based on the information returned, the query will be modified in an attempt to optimise it. After each optimisation the query will be benchmarked again and the execution times compared.

Optimisations will focus on the following:

- Using indexes, primary and foreign keys were possible.
- Proper use of the indexes, i.e. using *indexed_col* = 20 instead of *indexed_col* * 2 = 40.
- Removing unnecessary joins.
- Using static date and time values, where possible, instead of the NOW() function.
- Temporarily increasing the size of session variables like the *join_buffer_size*.
- Adding additional indexes *if* deemed necessary.

Stored procedures based on the query above will also be created. The stored procedures will be benchmarked with the same *mysqlslap* configuration as is used for the normal query.

3.2.3 **Configuration of the Server System Variables**

The last step in the optimisation process DBMS will be to configure the system variables of the MySQL server to make optimal use of the available resources. The MySQL server (*mysqld*) will be tuned for maximum write (INSERT) performance. Several server variables will be tuned and benchmarked to find the optimal settings. The variables that will be configured will be chosen based on the primary storage engine that will be used.

Each of the variables will be adjusted several times and the change in performance will be measured using the benchmarks' processes discussed below. Two separate benchmarks processes will be used to measure the effect that each change will have on the write performance of the DBMS. The tables that will be used for the benchmarks are based on those that will be developed for the OPC data-logging database. The MySQL Administrator [51] client application will be used to adjust the values of *mysqld*'s system variables. The *msqlslap* application was used to execute the benchmarks.

The first benchmark process will simulate 50 users/applications simultaneously inserting 5 000 records, for a total of 250 k records [52], using single insert transactions. The *msqlslap* application will be configured to repeat this part of the process (subsequently referred to as Part A) three times. The execution times of Part A will be recorded and the tables will be truncated. Part A will then repeat ten times to average out any possible anomalies.

The second benchmark process will be similar to the first; with the exception that it will make use of bulk inserts transactions instead of single insert transactions. Here each user/application will execute five bulk record inserts, each consisting of a thousand records. Other than this the process will be the same as the first. The *mysqlslap* scripts for these benchmarks are illustrated in Figure C-1 and Figure C-2.

After each set of benchmarks, on a system variable, is completed the DBMS configuration will be adjusted by setting the system variable to its optimal size or setting. A control benchmark will also be run before each benchmark set, to take into account any inconsistencies that might occur. Each set of benchmarks was completed in the same day. The *NormalInsert* and *BulkInsert* stored procedures are similar to those used in section 3.2.2.1.

3.3 Improving the DALC

The DALC will be completely rewritten, splitting its functionality into two parts. The first part will be a configuration program. The configuration program will provide the following functionality:

- Browse for OPC DA servers, either locally or on a remote computer on the network.
- Browse the OPC items on a selected OPC server.
- Select which items should be logged.
- Store the list of items to be logged in a file configuration file.
- Make an entry in the database for each new OPC server and item.

The second part of the DALC will read the configuration file, connect to the OPC DA servers and log the item information into the database. It will also be multi-threaded, separating the data acquisition and logging into separate threads. This will allow it to make better use of modern multi-core processors. To increase the performance of the DALC and reduce redundant data, the values of OPC items will only be logged when they change.

Since it can be difficult to configure OPC communication to work over a network, both parts of the DALC will be designed so that they can either connect to OPC servers and the database on the local computer or a computer on the network.

3.4 Summary

This chapter discussed the methods, techniques and equipment that will be used in this study. The main topics of this chapter include:

- Identifying the specifications for the new servers.
- Selecting the proper RAID configurations.
- The design and implementation of the backup scheme.
- Installation of a UPS and configuration of its software.
- Redesign of the database schema and configuration of the DBMS.
- Improving the DALC.

Where applicable, the benchmarks methodologies that will be used were also discussed. Additionally the scripts that will be used as part of the benchmarks methodologies, backup scheme and power management scheme were explained. The following chapter will discuss the benchmark results and final configuration of the different components of the system.

4 Results

This chapter will discuss the final implementation of the data-logging system. The results of the different benchmarks, as described in Chapter 3, are presented and discussed where relevant. Section 4.1 focuses on the hardware and infrastructure component of the system. Section 4.2 discusses the improvements to the database schema and other related topics. The configuration of the MySQL server variables are discussed in Section 4.3 and Section 4.4 presents and discusses the improved DALC.

4.1 Hardware Infrastructure

When this project started, the hardware infrastructure of the system consisted of two office computers that were connected through a dated 10 MB network switch. It offered no protection against hardware failure and little against power failures. Additionally, one of the computers had to fulfil the functionality of both a database and application server. This further decreased the performance of the data-logging system, seeing that the already inadequate resources had to be shared.

In response to these problems three new servers were implemented. The specifications of the servers are as follows:

- A 2.4 GHz Intel Q6600 Quad core Central Processing Unit (CPU)
- 4 GB (2 X 2 GB) Dual Data Rate (DDR) 2 800 Low latency Random Access Memory (RAM) modules
- Three 250 Gigabyte (GB) hard disks
- A Gigabyte GA-EG45M-DS2H motherboard
- A 500 W power supply
- DVD writer
- Stiffy drive

The hardware configuration of the database server differed slightly from the other two servers. These differences are as follows:

- An Adaptec 3405 RAID controller was added.
- A 700 W Power Supply Unit (PSU) was used instead of the 500 W PSU.
- Four additional hard disks were added.
- Two 5.25" hard disk bays.

The selected motherboard is based on the Intel G45 Express chipset [**53**]. It features an integrated display, an integrated RAID controller, Gigabit LAN capabilities, USB 2.0 support, eSATA support, and a PCIe (Peripheral Component Interconnect express) x4 slot (needed for the Adaptec RAID controller).

During the course of the study the memory of all the servers was upgraded by adding two more RAM modules, identical to the initial modules, to each server. This increased the memory capacity of the servers to 8 GB each. The power supply of the database server was also upgraded to a 1200 W unit. This was necessary since the server would become unstable when placed under load and in some cases will simply reset due to lack of power. A backup battery was also added to the RAID controller. This was necessary to prevent data loss or corruption when the RAID controller's buffered write is enabled.

The hardware components of all three servers were installed into 4U rack mountable chassis. These chassis were then mounted in a 40U cabinet. Additionally two-gigabit switches, a cable management unit, and UPS were mounted below the servers. A 19" LCD display, mouse and keyboard, and a keyboard, video and mouse (KVM) switch was placed on a 1U shelve that was installed above the server cases. The routers, NAS, multi-plugs and power extensions are placed on another shelve towards the top of the cabinet.

The Gigabit Ethernet switch, combined with Cat-6 cables, ensures fast data transfers between the servers. Most of the current devices and equipment on the network only support 10/100 MB Ethernet; they are connected using normal Cat-5 cables. Figure 4.1 shows a detailed representation of the network layout. Note that the diagram also includes several changes to the network layout, like the addition of the second switch that was done by fellow students. These changes are outside the scope of this study and will not be discussed here.

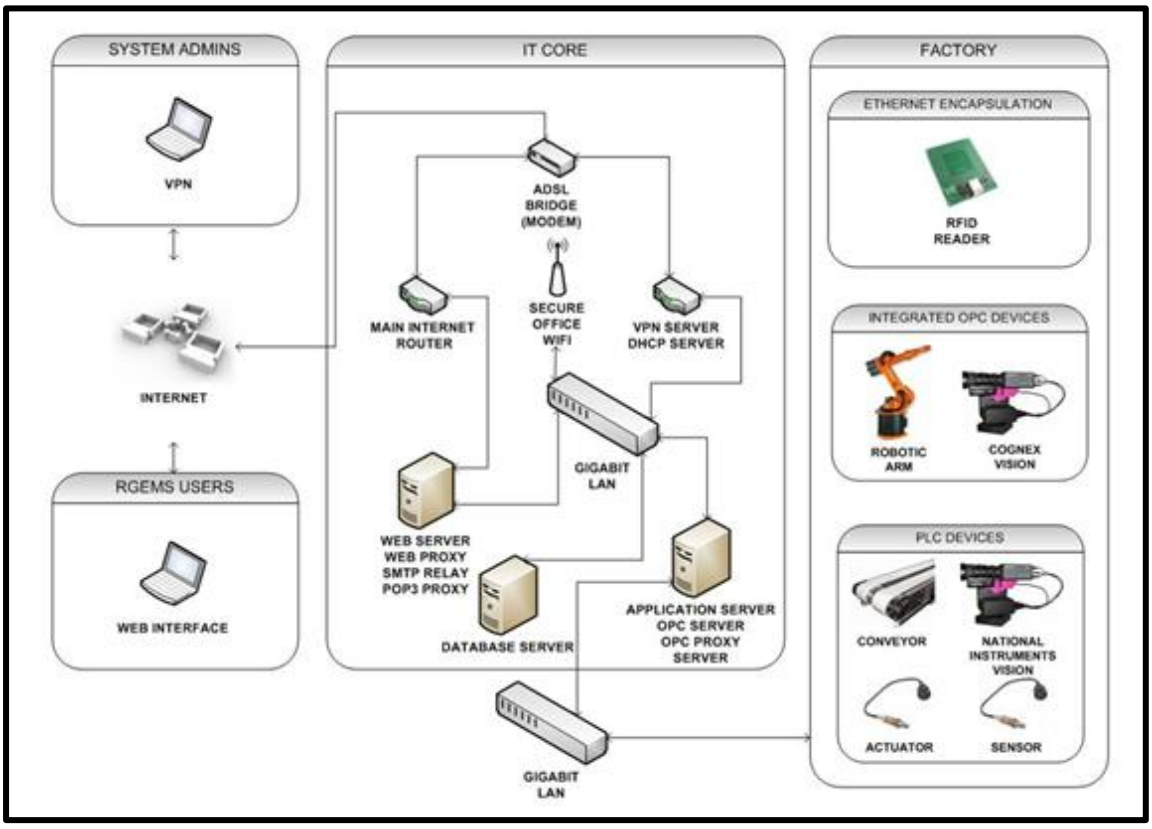
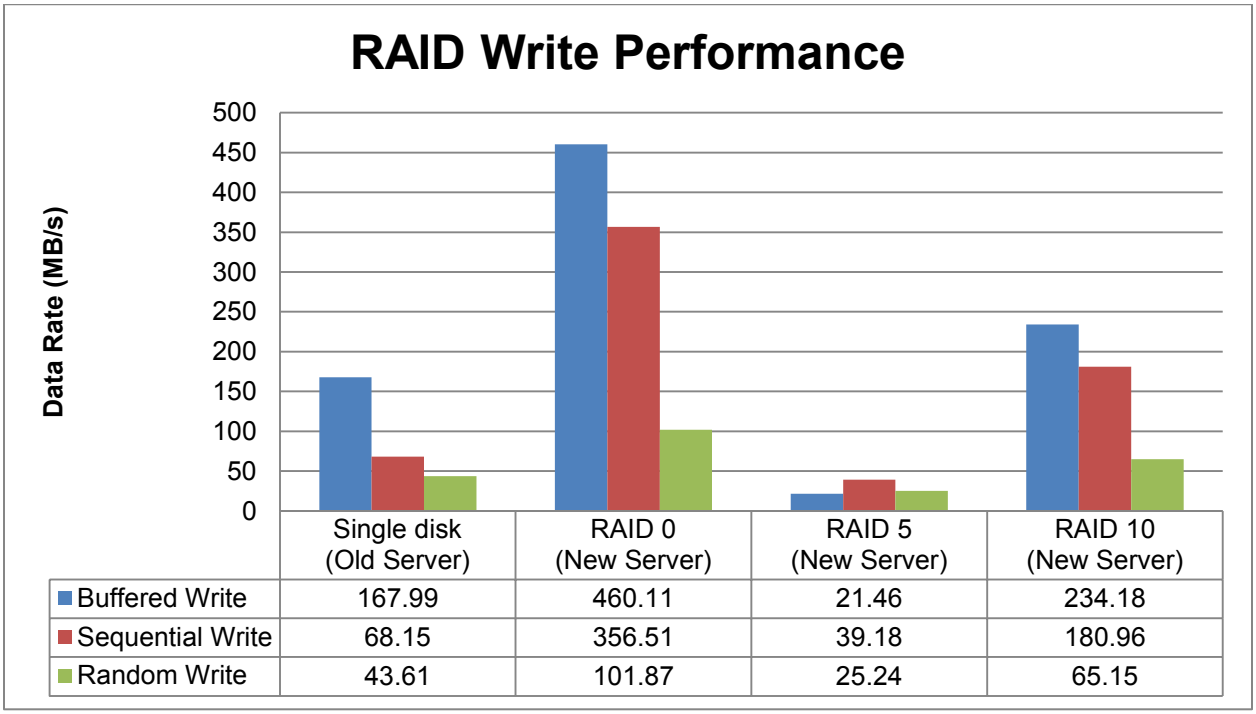


Figure 4.1 Detailed Network Layout [54]

4.1.1 **RAID Benchmarks and final RAID configuration**

The results of the RAID benchmarks, the same as in Figure 4.2 and Figure 4.3, were similar to what was expected. RAID 0 proved to be the fastest overall. RAID 5 performed significantly worse than a single disk with the write benchmarks. It also performed significantly better with the read benchmarks. The RAID 10 configuration offered a modest improvement in both read and write performance when compared to the single disk.



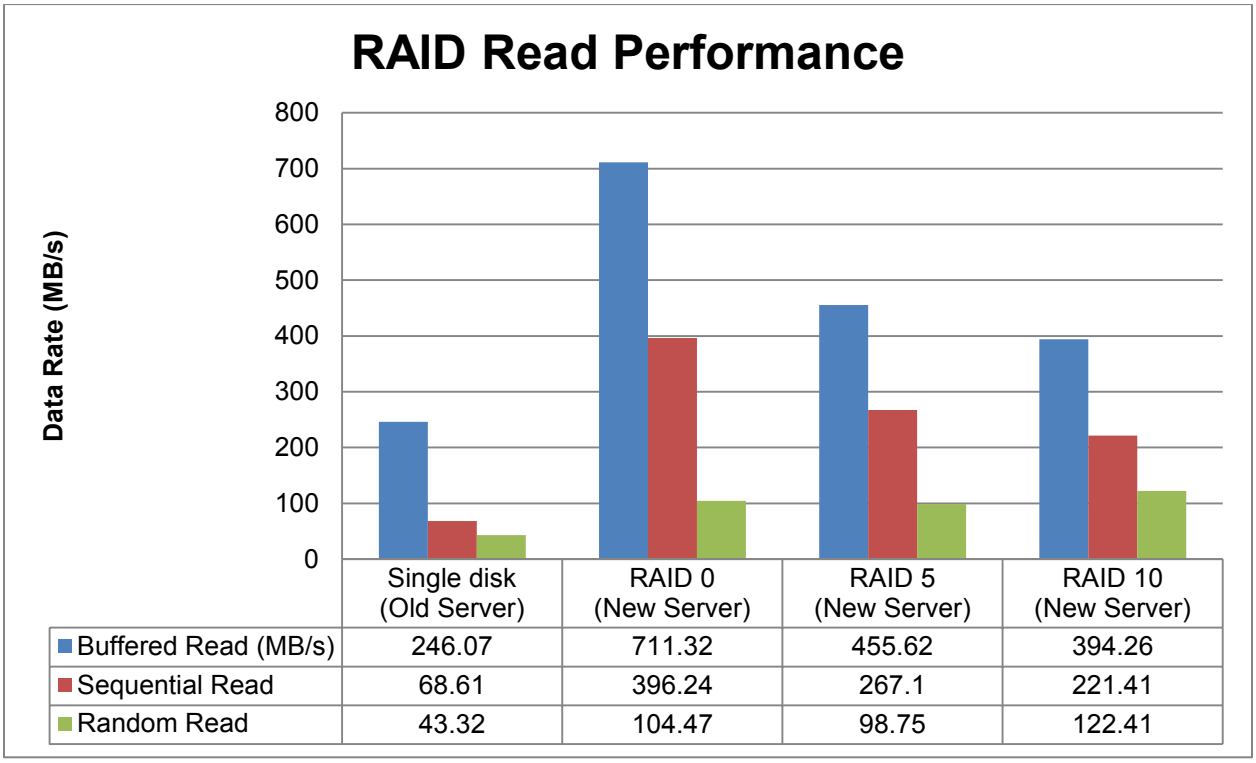
	Single disk (Old Server)	RAID 0 (New Server)	RAID 5 (New Server)	RAID 10 (New Server)
Buffered Write	167.99	460.11	21.46	234.18
Sequential Write	68.15	356.51	39.18	180.96
Random Write	43.61	101.87	25.24	65.15

Figure 4.2 RAID Write Performance

After taking both the redundancy and real-world performance of each RAID configuration into account, RAID 10 was selected for the final configuration. Offering increased read and write performance in addition to up to two simulation disk failures; it is simply the most robust of the RAID levels tested.

During the course of the RAID benchmarks and database related benchmarks, several hard disks failed. The RAID array of the Adaptec controller suffered the most hard drive failures, with a total of three disk failures, of which two failed at once (one in each mirror-set). This meant that the data could still be accessed, even though the array was crippled. In each case the faulty disks were replaced and the array recovered automatically.

The RAID arrays that were created using the motherboard's integrated RAID controller were very unreliable. Initially a two disk RAID 1 configuration was used as a system disk to install the OS on. This solution worked initially, but one of the mirrors continually lost synchronisation. This resulted in the array having to be constantly rebuilt, slowing operations on the server.



	Single disk (Old Server)	RAID 0 (New Server)	RAID 5 (New Server)	RAID 10 (New Server)
Buffered Read (MB/s)	246.07	711.32	455.62	394.26
Sequential Read	68.61	396.24	267.1	221.41
Random Read	43.32	104.47	98.75	122.41

Figure 4.3 RAID Read Performance

As an alternative a three disk RAID 5 configuration was implemented. This proved to be more reliable and stable than the RAID 1 configuration, until one of the disks in the array failed. The integrated RAID controller turned out to be a hybrid RAID controller, instead of a pure hardware controller. This meant that while a basic RAID 1 array could be recovered by the controller itself, a more advanced RAID 5 configuration could only be recovered in a Windows environment.

This lead to the secondary problem encountered with each disk failure. Unlike the Adaptec controller, the integrated controller could not reconstruct the data of the missing disk on the fly. This proved to be problematic, since the OS itself was installed on the array. As a result of these problems with the integrated RAID controller it was decided to rather run the OS on a single disk and incorporate the redundant disks into the backup strategy.

4.1.2 Backup strategy in Action

The backup strategy was successfully implemented as discussed in Chapter 3. Only one problem was encountered while implementing it. It was found to be nearly impossible to do the monthly backups directly to the NAS. For some unknown reason WBADMIN failed to authenticate with the NAS; and as a result it could not detect the shared folder created for the monthly backup. As an improvisation to this it was decided to simply use a copy of the latest daily backup as the monthly backup. A simple command-line *xcopy* script was implemented to copy it to the NAS.

The final backup strategy was thoroughly tested to identify any problems that might occur during the backup and restore process. The backups proved to be invaluable, as all the servers failed at least once during the course of this project. The failures were mainly due to software corruption and hardware failures. In every case the backups could be quickly restored and operation continued.

4.1.3 UPS and Power management

UPSMON itself proved to be quite limited in functionality. However, when combined with the scripts that utilises *PsShutdown*, to shutdown the application and web servers it proved to be adequate.

The Power management strategy was also thoroughly tested in controlled conditions to ensure proper operation. It was also unexpectedly "field tested" during a series of blackouts caused by load shedding. In all cases it proved to be very reliable, shutting down the servers in the given time frame and then restarting after the power has been restored.

4.1.4 Security and Database user accounts

Several database user accounts were created. The accounts were created using Navicat. Three developer accounts were created with limited administrative rights. These accounts have full rights to schemas that they create, and read access to related schemas that other users created. The developer accounts can also create new users, but can only grant access rights to their database schemas. These accounts will be disabled once they are not needed any more.

An account was also created for the DALC that has very limited rights. It has read access to the OPC servers, and limited read and write access to the database schema. The default super administrator account, root, was also disabled and a new one was created with a stronger username and password. In addition a normal administrator account was also created for general administration of all databases.

The following rules were also defined concerning the creation of new user accounts:

- The user names must be at least eight characters long, but no longer than 16 characters (this is a MySQL limitation).
- User names must contain at least one capital letter and one numerical digit.
- Passwords must be at least eight characters long for basic users and 16 for developers and other limited administrative users.
- Passwords for normal administrative users must be at least 20 characters long.
- The password for the super administrative must be at least 24 characters long.
- All passwords must contain at least one capital letter, one numerical digit and one special character.
- Regularly used accounts must change their passwords once a month.

Unfortunately the MySQL DBMS cannot enforce these rules by itself, and it is up to the administrators to carry out this task.

4.2 The MySQL DBMS

4.2.1 The Database schema

The database schema, as in Figure 4.4, was designed around the InnoDB storage engine. Despite being the slowest in terms of write performance, InnoDb was chosen as the primary storage engine due to several important factors. The first, and most important, being that it is a transactional storage engine; this means that foreign keys and relational integrity enforcement are supported. All the other storage engines are non-transactional. Secondly InnoDB supports row level locking allowing for multiple updates and inserts to be executed simultaneously. The third factor being that InnoDB caches both data and indexes. MyISAM only caches the table indexes and relies on the OS to cache its data. Most other storage engines rely purely on the OS to cache their data. InnoDB also offers a variety of system variables that can be fine-tuned to improve performance.

Archive was selected as the secondary storage engine, due to its fast writing speeds and high compression rates. As ARCHIVE based tables do not enforce FOREIGN KEY constraints it cannot be linked to the InnoDB tables. The relationship must therefore be enforced on application level. The table design and events will be discussed in more detail in the next section.

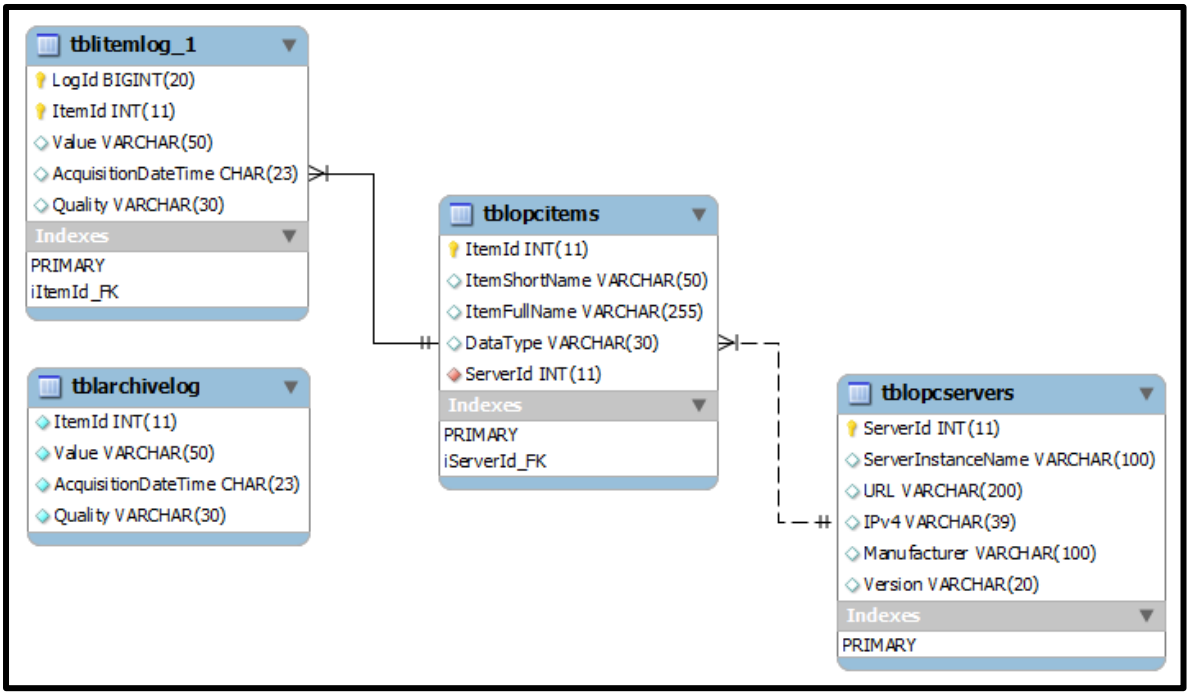


Figure 4.4 The OPC data-logging database schema

4.2.1.1 Tables and indexes

The schema was designed with the concept of a compact database in mind. Most of the static data columns were moved to the OPC items and OPC server tables. These included the server and item information that is unlikely to change. The result being a small logging table containing only the most necessary data columns.

The relationship between the table of the OPC server and the table of the OPC items was setup so that each OPC server entry can be linked to multiple OPC item entries; but any given item can only be linked with one OPC server. The same principle applies to the relationship between the OPC item's table and the data-logging table. The archive log table is not in an enforced relation with the other tables, as the Archive storage engine is non-transactional, and does not support foreign keys and referential integrity.

The smallest possible data type that could store the relevant information was also selected when designing the tables. Note that the *AcquisitionDateTime* column makes use of a CHAR(23) data type instead of a DATETIME data type. The reason for this is that millisecond resolution is required on this column and MySQL's date and time data types do not support this.

Indexes were created on all columns that were used in table relations, i.e. all primary and foreign keys. A composite primary key, using an auto-increment log ID and the item ID columns, was created to ensure that each log entry would have a unique identifier. A separate index was also created on the item ID column to ensure that table JOINs would be performed more efficiently. Due to the limitations of the Archive storage engine, no indexes were placed on the archived log table.

One limitation concerning partitioning in MySQL was also encountered. Currently MySQL does not support foreign keys in partitioned tables, including tables based on InnoDB. For this reason partitioning could not be applied to the main tables of the schema.

The archived log table, however, is not affected by this limitation and was partitioned. The partitioning function made use of the TO_DAY() method and the *AcquisitionDateTime* column to partition the table accordingly. This will allow queries that make use of the *AcquisitionDateTime* column, in a WHERE statement, to take advantage of partition pruning.

Note that a special partition, pNULL, was created. As the name suggests this partitions will store all the records of which the partitioning function returns NULL; in this case, records with invalid date time values. If this partition were not created these records would be stored in partition p0. The result will be that p0 will always be included in the list of partitions to be accessed after partition pruning has been applied. This might be detrimental to query performance, as p0 will become quite large over time. The SQL code used to create the tables of the schema can be found in Appendix D.

4.2.1.2 Events

Figure 4.5 shows the SQL code used to create the events. While both events form part of the backup strategy, the prior also archives all the old log records. Both events have been tested and confirmed to work as desired. In addition to creating the event the event scheduler itself was also enabled with the *SET GLOBAL event_scheduler = ON* command.

```
DELIMITER $$
#Event 1
CREATE  EVENT `opclog`.`eArchive_opclogs`
ON SCHEDULE EVERY 1 MONTH
STARTS CONCAT(LAST_DAY(NOW()),' 05:00:00')
DO
    BEGIN
        SET @tempdate = NOW();
        SELECT SLEEP(1);
        INSERT INTO tblarchivelog

        SELECT NULL, ItemId_fk, ItemValue, AcquisitionDateTime, ValueQuality
        FROM tblitemlog_1
        WHERE AcquisitionDateTime < @tempdate;

        DELETE FROM tblitemlog_1
        WHERE AcquisitionDateTime < @tempdate;
    END$$
#Event 2
CREATE EVENT `opclog`.`eFlush_binlogs`
ON SCHEDULE EVERY 1 DAY
STARTS CONCAT(LAST_DAY(NOW()),' 23:00:00')
DO
    BEGIN
        FLUSH TABLE;
        FLUSH LOGS;
    END$$

DELIMITER ;
```

Figure 4.5 SQL Code for the Two Events

4.2.2 **Bulk INSERT and Single INSERT Comparison**

Figure 4.6 illustrates the performance difference of bulk record inserts and single record inserts on several storage engines. For MyISAM and Archive the performance difference was negligible with 10 and 100 records; it became clear that bulk record inserts only offer a significant increase in performance with large amounts of records. For InnoDB, however, the bulk inserts made a significant difference in every benchmark. With 10 records, the batch inserts were approximately three times faster; and with 10 000 records they were approximately 404 times faster.

Note that the time taken to execute single INSERT statements scale almost linearly with the amount of records. This drastic performance difference of single and bulk record inserts is partially due to the way InnoDB is configured to flush its transactions. By default *innodb_flush_log_at_trx_commit*, the server variable that controls this behaviour is set to flush the data to disk after every transaction. While a bulk insert with any number of records is accounted for as one transaction, each single record insert statement on its own, accounted for as one transaction. This means that 10 single record inserts are treated as 10 transactions.

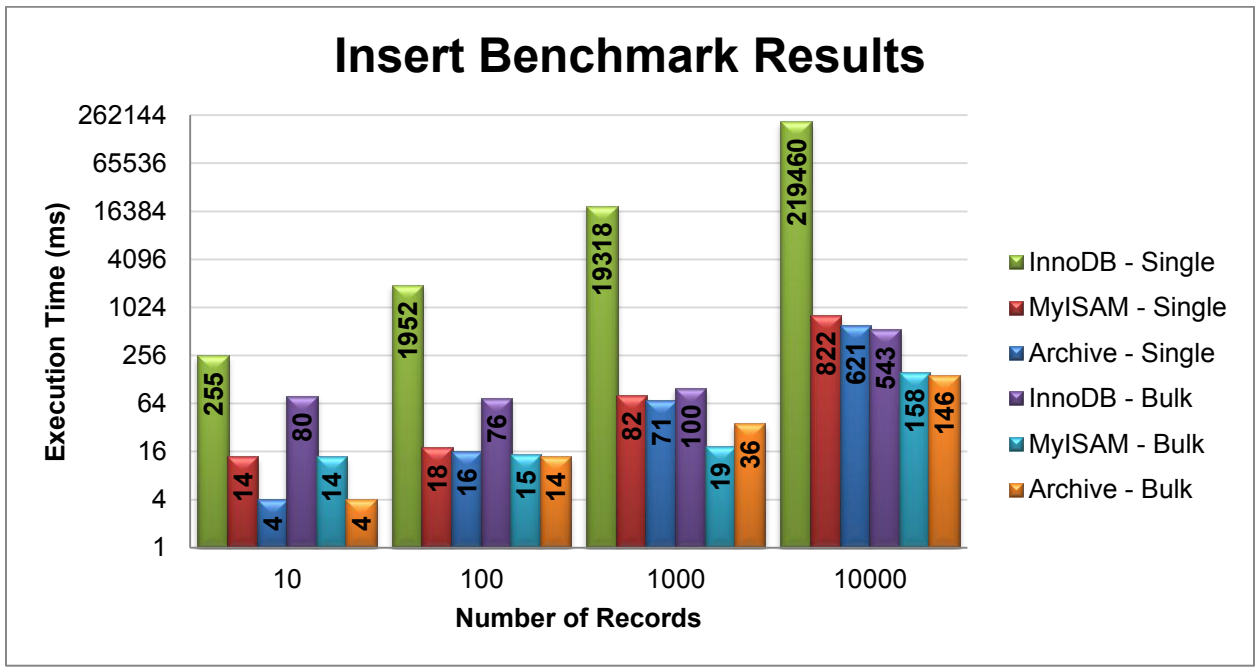


Figure 4.6 Performance Comparison of Bulk Record and Single Record INSERT statements

Another important issue to note is that the bulk record inserts was never slower than single record inserts. With this information it was decided that the new DALC would be designed to take advantage of bulk record inserts.

4.2.3 **SELECT Optimisations**

The query selected to illustrate the Select optimisation process is illustrated in Figure 4.7. It consists of two queries, the primary (outer) query and the derived (inner) query. The inner query selects a list of the OPC item IDs and their calculated average value between the two dates. The outer query then uses this information to select all the recorded values, and some Meta data, of the items whose average is less than zero and belong to TestServer4.

```
SELECT a.ItemShortName, c.Value, c.AcquisitionDateTime
FROM tblopcitems a, tblopcservers b, tblarchivelog c,
(SELECT ItemId, AVG(VALUE) AS Avg_Vlaue
FROM tblarchivelog
WHERE AcquisitionDateTime
        BETWEEN '2008-07-15'
        AND '2009-05-15'
GROUP BY ItemId) d
WHERE d.Avg_Vlaue < 0
AND a.ItemId = c.ItemId
AND a.ItemId = d.ItemId
AND a.ServerId = b.ServerId
AND b.ServerInstanceName = 'TestServer4';
```

Figure 4.7 The SELECT query to be optimised

Mysqlslap was configured to simulate 10 concurrent connections and to repeat each benchmark 10 times, seeing that the query was expected to be rather slow. The benchmark results confirmed that the query was slow, with the average time to complete a single iteration of the benchmark being 73.659 seconds. This translates to approximately 7.366 seconds per query.

After executing this un-optimised query with EXPLAIN PARTITIONS the following information was obtained:

- The derived query performed a full table scan on *tblarchivelog*, but due to partition pruning only partitions p0, p1 and pNULL were scanned. It also made use of a temporary table and a file sort.
- The primary query had to perform a full table scan on the result set of the derived query, although the result set only contains 100 records.
- The primary query could perform an *eq_ref* join on *tblopcitems* and the result set of the derived query. This was possible due to the fact that the primary key of *tblopcitems* could be used to satisfy the relevant WHERE clause.
- The primary query could perform an *eq_ref* join on *tblopcservers* and *tblopcitems*, due to *tblopcservers*'s primary key.
- The primary query performed full table scan on *tblarchivelog* to retrieve the relevant information. All partitions were scanned and the Join buffer was used to assist with the join on *tblopcitems* and *tblarchivelog*.

The derived query's table scan was as expected, since *tblarchivelog* contains no indexes. Fortunately the date range allowed the query optimiser to use partition pruning to reduce the table scan to only a few partitions. As for the file sort and the temporary table, they were caused by the GROUP BY statement and the lack of an index on the *ItemId* column. Unfortunately no optimisation can be done here, since no indexes can be added to this table.

For the primary query, however, there are multiple possible optimisations. The first being the unnecessary join on *tblopcservers* and *tblopcitems*. It is unnecessary because no information is retrieved from *tblopcservers*. This join can easily be replaced by comparing the *tblopcitems*'s *ServerId* column to a constant value equal to the *SeverId* of TestServer4. Figure 4.8 shows the modified query implementing this optimisation.

```
EXPLAIN PARTITIONS
SELECT a.ItemShortName, c.Value, c.AcquisitionDateTime
FROM tblopcitems a, tblarchivelog c,
(SELECT ItemId, AVG(VALUE) AS Avg_Vlaue
FROM tblarchivelog
WHERE AcquisitionDateTime
        BETWEEN '2008-07-15'
        AND '2009-05-15'
GROUP BY ItemId) d
WHERE d.Avg_Vlaue < 0
AND a.ItemId = c.ItemId
AND a.ItemId = d.ItemId
AND a.ServerId = 4;
```

Figure 4.8 The First revised version of the query

The benchmarks of this query were faster with more than six seconds. The average iteration execution time was 67.218 seconds, approximately 6.722 seconds per query. EXPLAIN PARTITIONS revealed no new information. The only change was that the join on *tblopcservers* and *tblopcitems* was no longer listed.

The second optimisation to the query has to do with the *Avg_Value* < 0 condition. If this filter condition is moved to the derived query, it should decrease the records in the derived result set. This should speed up the join on *tblopcitems* and *tblarchivelog*. Figure 4.9 shows the modified query implementing this optimisation.

```
EXPLAIN PARTITIONS
SELECT a.ItemShortName, c.Value, c.AcquisitionDateTime
FROM tblopcitems a,tblarchivelog c,
(SELECT ItemId, AVG(VALUE) AS Avg_Vlaue
FROM tblarchivelog
WHERE AcquisitionDateTime
        BETWEEN '2008-07-15'
        AND '2009-05-15'
GROUP BY ItemId
HAVING Avg_Vlaue < 0) d
WHERE a.ItemId = c.ItemId
AND a.ItemId = d.ItemId
AND a.ServerId = 4;
```

Figure 4.9 The Second Revised Version of the Query

Benchmarks of this query showed that it was on average 4.753 seconds faster than the previous. The average iteration execution time was 62.465 seconds, approximately 6.247 seconds per query. EXPLAIN PARTITIONS revealed that the result set of the derived query now contained only 52 records, instead of 100.

The optimisation would be adjusting the size of the join buffer session variable, setting it to 1 MB. This should theoretically speed up the join on *tblopcitems* and *tblarchivelog*. This can be achieved by executing *SET SESSION join_buffer_size = 1*1024*1024,* before the query.

Benchmarks showed this actually slowed the query down. The average iteration execution time was 67.193 seconds, approximately 6.719 seconds per query. One reason for this could be that there is not enough available memory for the larger buffers, causing the OS to start paging. This is very unlikely as these benchmarks were done on an un-optimised *mysqld* configuration, which utilises only 4 GB of the 8 GB available RAM. A more reasonable explanation could be that the time it takes to increase the session's join buffer, causes the increase in overall execution time. A better solution would be to set the global size of the join buffer to a larger size if queries utilises it frequently. Figure 4.10 shows a comparison of the minimum, maximum and average benchmark iteration times of all four scenarios.

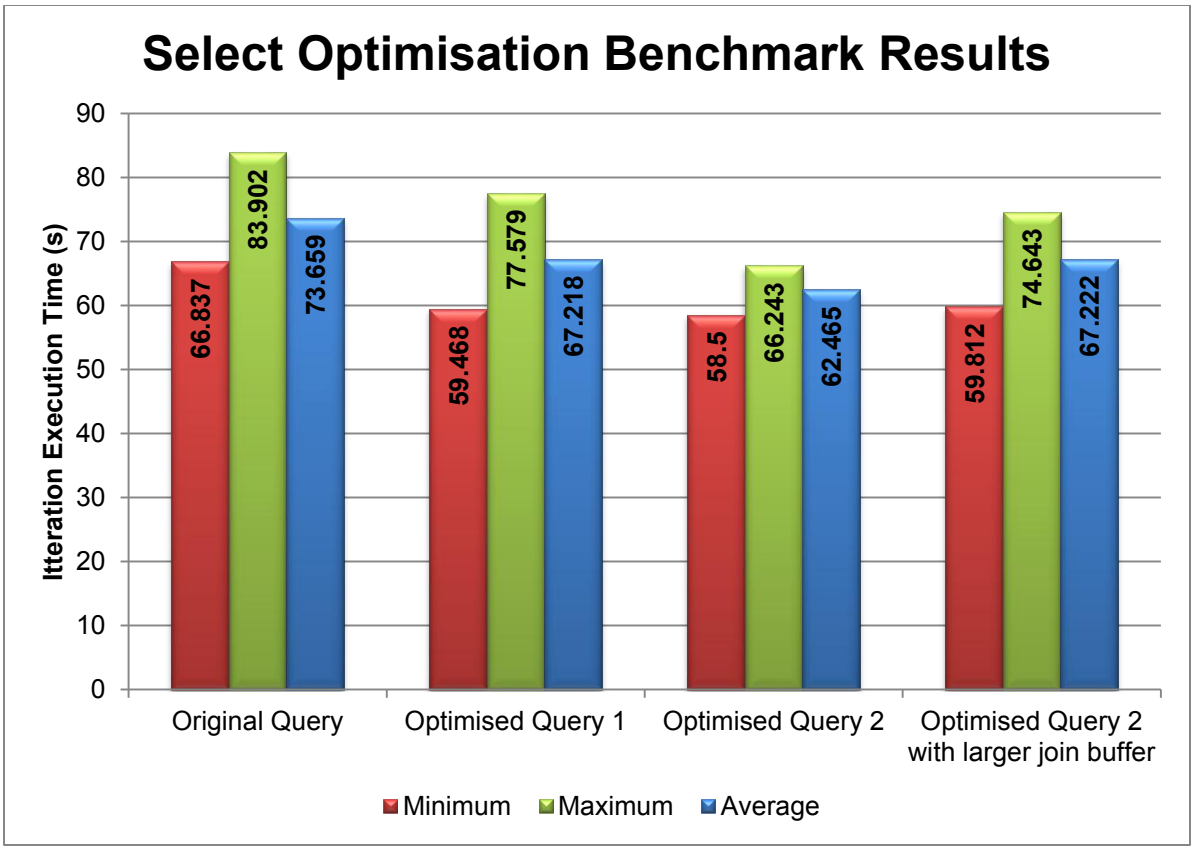


Figure 4.10 Select Optimisation Benchmark Results

4.3 Final Configuration of the MySQL Server

The initial configuration of the MySQL server was tuned with the write performance of the DBMS in mind. The performance of both single and bulk insert transactions were taken into consideration as both might be utilised by other users and applications. Note that these changes will affect the entire DBMS and not just the OPC data-logging schema. Also note that these results of these benchmarks are based on 50 concurrent connections. The approximate execution time of an individual transaction can be calculated using Equation (3:2).

$$t \approx {}^{T}\!/_{50} \qquad (3:2)$$

This the approximate execution time of a single transaction and T is the averaged execution time of all 50 concurrent connections.

Before the adjustment of each system variable, a benchmark was run on the current settings. The results of this benchmark are indicated as the default performance in the tables. After each variable was tuned, and the settings in the *my.ini* file were adjusted, a new base benchmark was run to take into account any inconsistencies that might occur between benchmark sets. Each set of benchmarks was completed in the same day.

As InndoDB would be used as the primary storage engine, the variables that have the greatest influence on performance were selected for fine-tuning. They include [**52**]:

- innodb_buffer_pool_size
- innodb_log_file_size
- innodb_flush_log_at_trx_commit

The *innodb_buffer_pool_size* variable was the first to be adjusted and benchmarked. Benchmarks were performed at the following values: 2 048 MB (the default size), 4 915 MB (60% of the total memory), 5 734 MB (70% of the total memory) and 6 554 MB (80% of the total memory). The averaged results for this benchmark set are shown in Figure 4.11 and Figure 4.12.

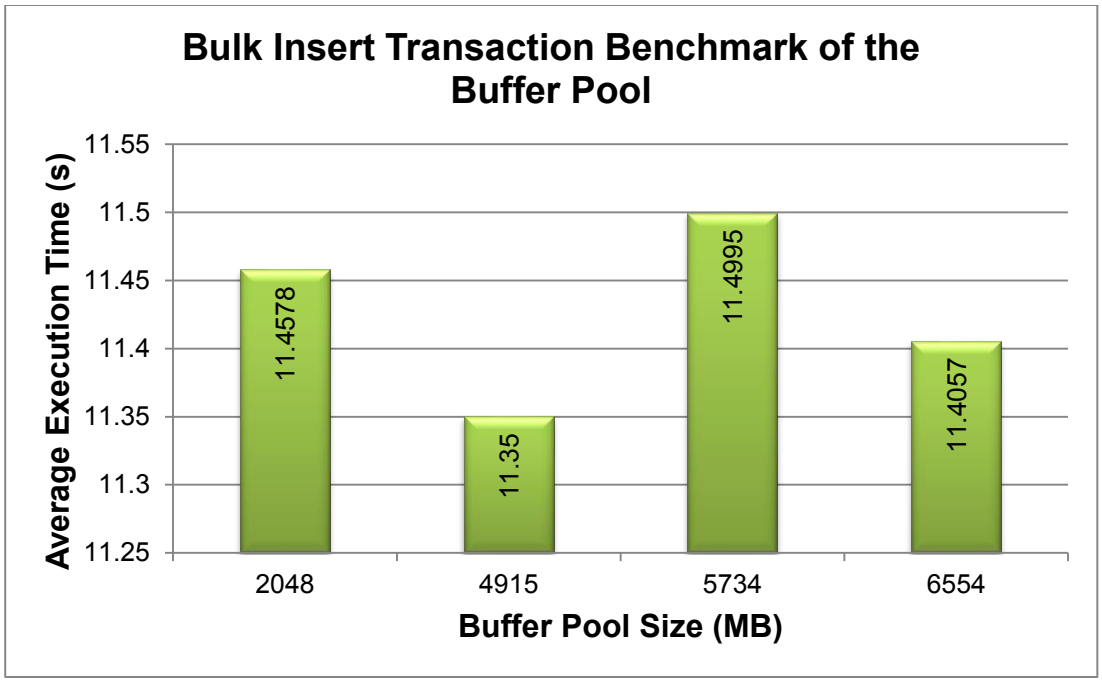


Figure 4.11 Bulk Insert Transaction Benchmark of the Buffer Pool

The optimum setting was found to be 70% of the total memory. Bulk insert transactions were 41 milliseconds slower, but the single insert transactions were 4.92 seconds faster.

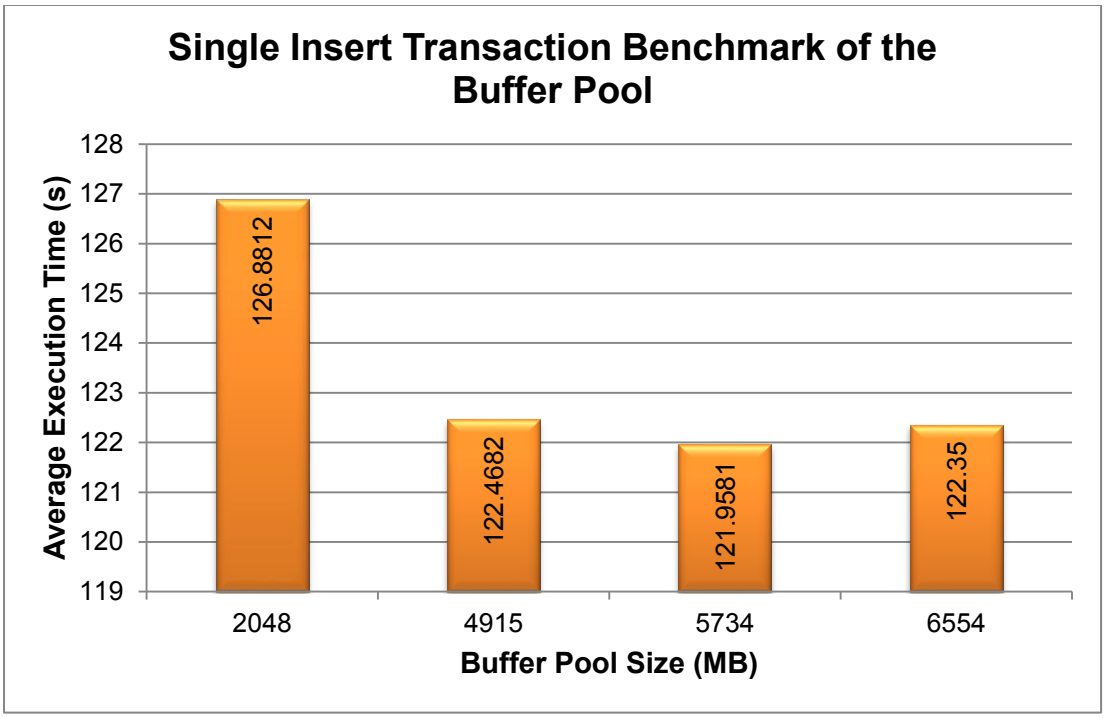


Figure 4.12 Single Insert Transaction Benchmark of the Buffer Pool

innodb_log_file_size was the second system variable that was tuned. Benchmarks were performed at the following values: 222 MB (the default size), 512 MB, 1 024 MB, and 2 047 MB. The averaged results for this benchmark set are shown in Figure 4.13 and Figure 4.14. The optimal log file size was found to be 512 MB. At this size the single insert transaction benchmarks were 608 milliseconds slower, but the bulk insert transaction benchmarks were 179 milliseconds faster.

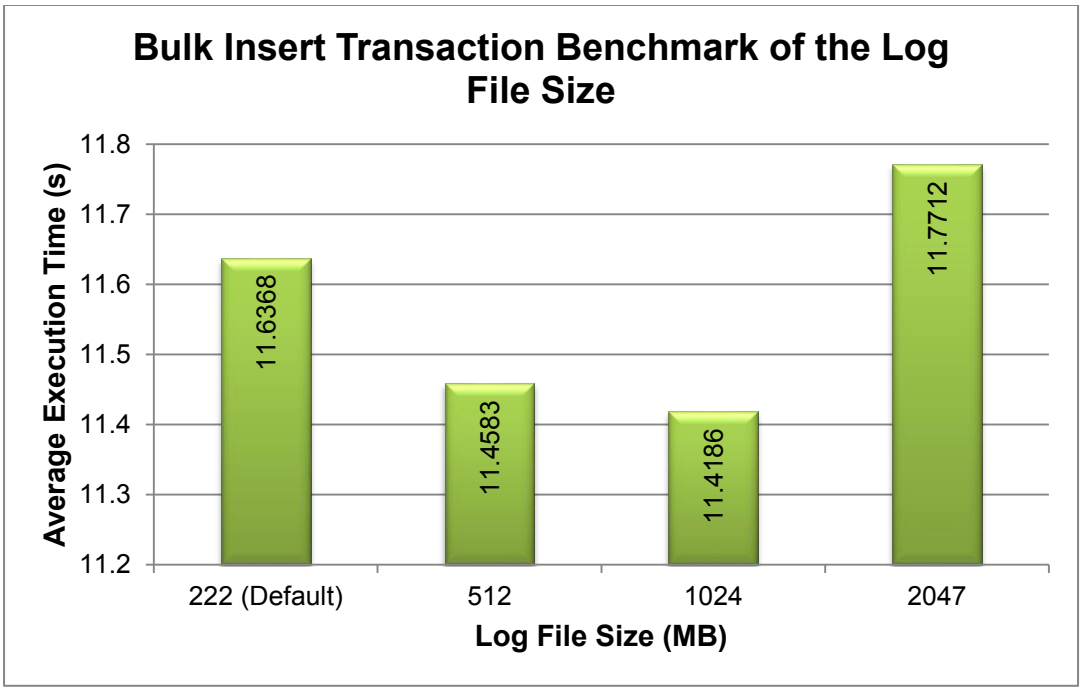


Figure 4.13 Bulk Insert Transaction Benchmark of the Log File Size

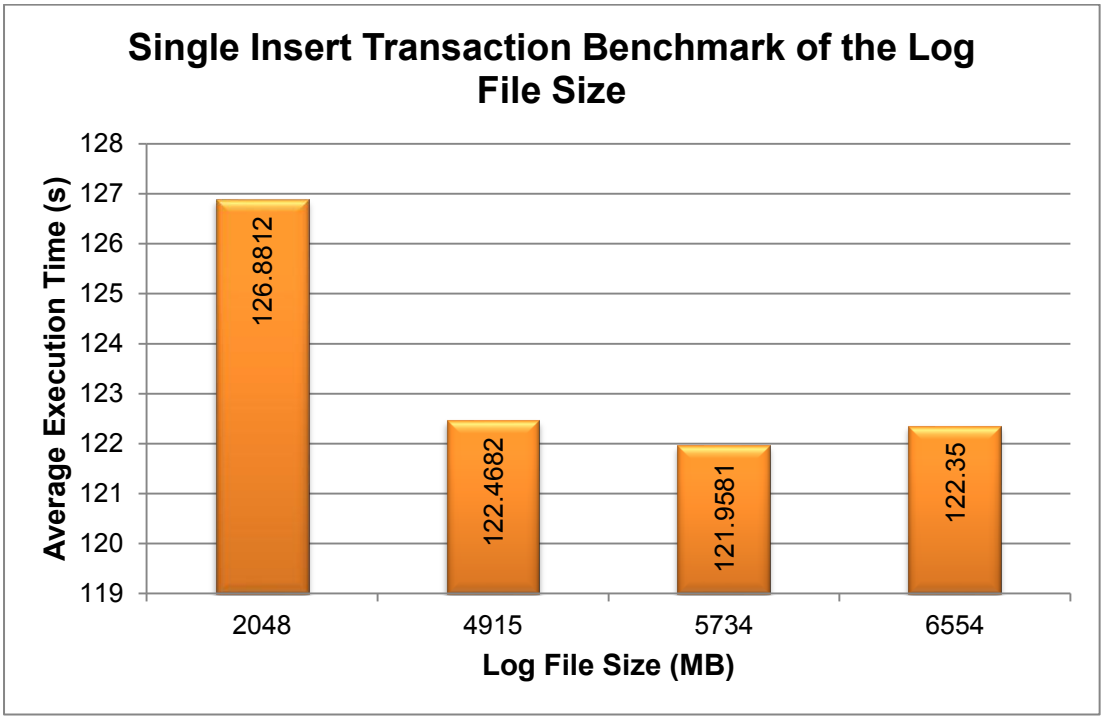


Figure 4.14 Single Insert Transaction Benchmark of the Log File Size

The third variable that was tuned was *innodb_flush_log_at_trx_commit*. It was benchmarked in conjunction with the RAID controller's battery backed write cache settings. The benchmarks were performed in the following order:

1. *innodb_flush_log_at_trx_commit* = 1 with the write cache disabled.
2. *innodb_flush_log_at_trx_commit* = 1 with the write cache enabled.
3. *innodb_flush_log_at_trx_commit* = 2 with the write cache disabled.
4. *innodb_flush_log_at_trx_commit* = 1 with the write cache enabled.

The averaged results for this benchmark set are shown in Figure 4.15 and Figure 4.16.

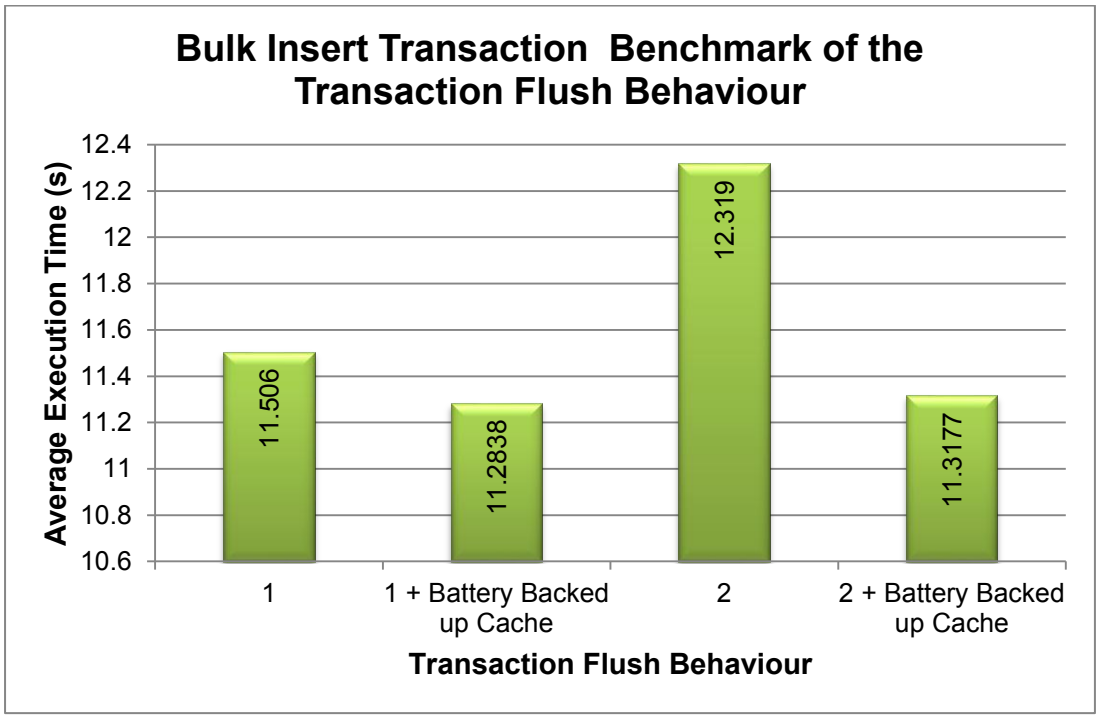


Figure 4.15 Bulk Insert Transaction Benchmark of the Transaction Flush Behaviour

The final configuration proved to be the best, with the single insert transaction benchmarks being 85 seconds faster and the bulk insert transaction benchmarks being 188 milliseconds faster.

The fine-tuning of these variables did not have a big influence on the performance of bulk insert transactions; decreasing the average execution time only by 140 milliseconds, which is approximately a 2.8 millisecond decrease in the execution time of each bulk insert transaction. For the single insert transactions it proved to have a significant effect. The average execution time was decreased by 85 seconds, which is approximately a 17 second decrease in the execution time of each single insert transaction.

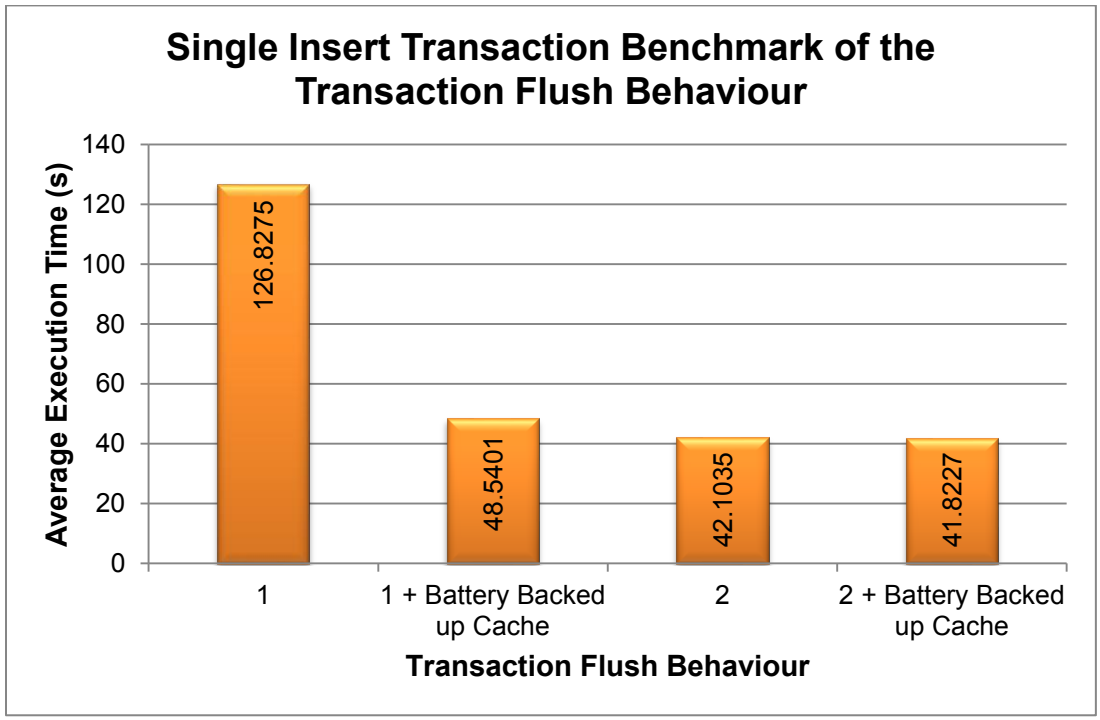


Figure 4.16 Single Insert Transaction Benchmark of the Transaction Flush Behaviour

Several other variables do not influence the performance of the DBMS, but rather its behaviour, has also been adjusted as follows:

- The *datadir* variable [1]: It was set to point to the RAID array, so that all tables would be created there.
- The *innodb_file_per_table variable* [1]: It was enabled so that the data for each table would be stored in separate files.
- *max_heap_table_size* = 366 M.

- *join_buffer_size* = 128 k.

4.4 The New DALC

As mentioned, the new DALC was designed as two separate applications. The first application is a configuration tool. It is used to configure the OPC server and database information, as well as to browse the servers and items and select which items to log. Figure 4.17 shows the "Configure Server Information" tab of the configuration tool. Under Host Information one can specify the IP address of the computer that hosts the OPC servers and the OPC data access specification. Data Access (DA) 1, 2 and 3 are supported. The database login credentials are specified under the Database Information section. The schema that contains the OPC data-logging database must also be specified here. After all the information is specified one can then add the server configuration to the list on the right-hand side. Multiple Server configurations can be added.

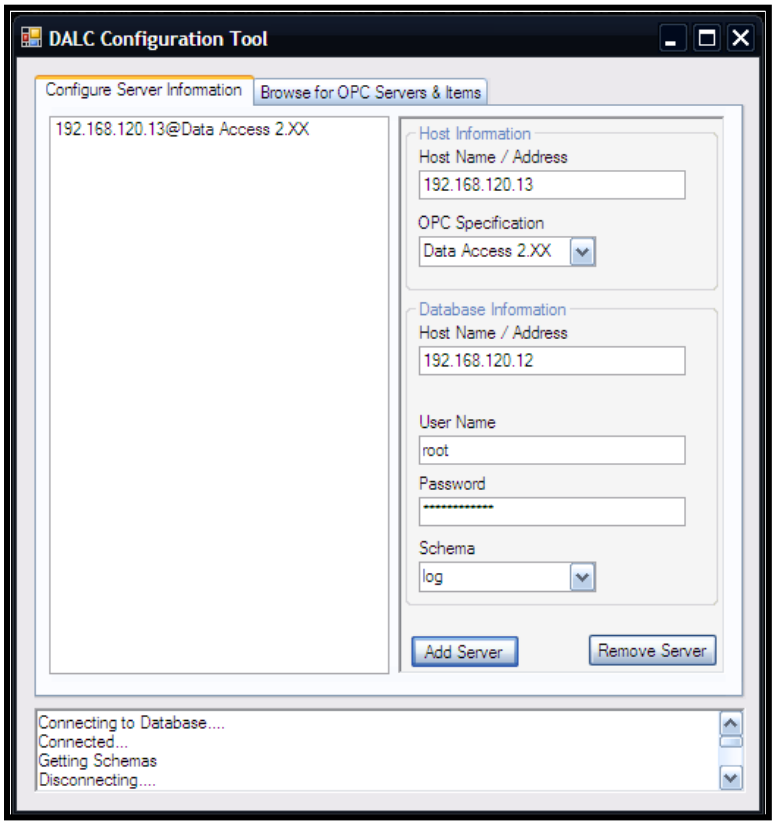


Figure 4.17 DALC Configuration Tool – Server Information

The "Browse for OPC Servers & Items" tab, refer to Figure 4.18, allows the user to browse all the available OPC servers, and their items, of the specified host computers. Individual items or entire groups can be selected to be logged, by selecting the checkbox next to its name. Before adding an item or a group one can also specify the rate at which the item values should be updated. Once all the items that must be logged have been selected, a configuration file containing all the preferences can be saved.

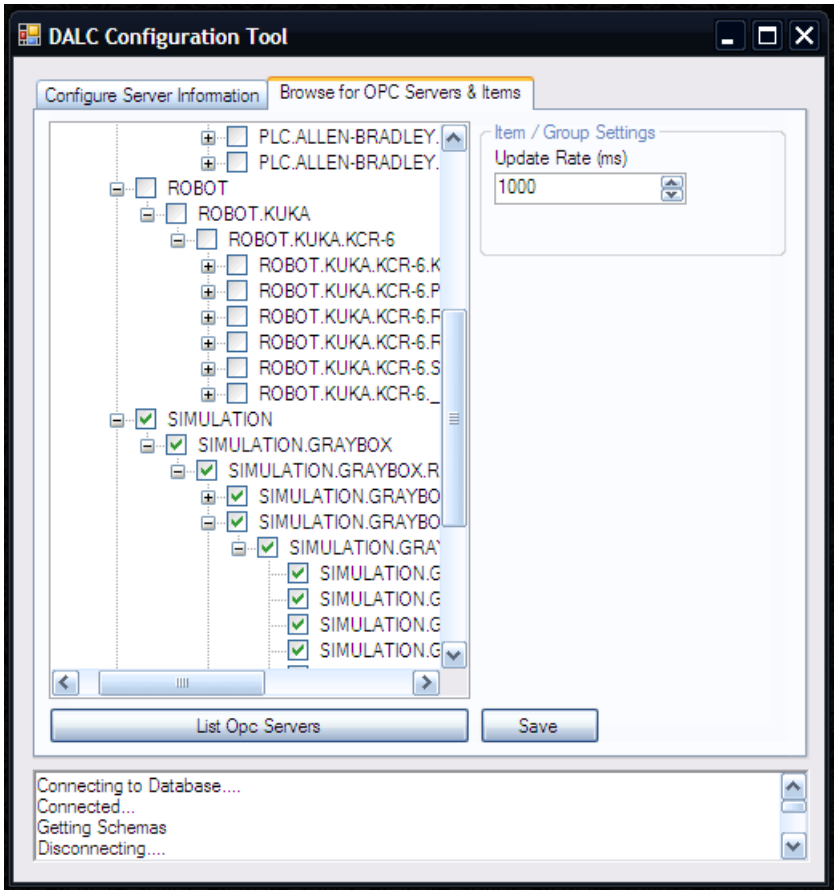


Figure 4.18 DALC Configuration Tool – Browse for OPC Servers and Items

The second program, refer to Figure 4.19, actually acquires the item data from the OPC servers and writes it to the database server. When it starts it will read the configuration file, generated by the configuration tool. After it has read the configuration file it will connect to the OPC servers and start logging the item data.

Note that when running these programs on a different computer than the one hosting the OPC servers one must comply with the following guidelines:

- On the client side the programs have to be executed using an account that mirrors an account on the OPC server host, i.e. if you run the programs with a user account *OpcUser* on a client machine, then that account must exist on the host computer as well.
- Both accounts must have the exact same username and password.
- The DCOM security settings must be configured as per the updated OPC DCOM white paper [55].

Figure 4.19 The Data-logging and Acquisition Client

4.5 Summary

This chapter focused on results obtained during the completion of the project. The chapter consists of four main sections. The first section discussed the final specifications of the new servers and other parts of the hardware infrastructure were introduced, as well as the results of the RAID benchmarks were reviewed. The operation of the backup strategy and power management system were also briefly discussed.

The second section discussed the redesigned database schema, the INSERT benchmarks, and SELECT query optimisations. The third section focused on the configuration of the MySQL server and the related benchmarks. The fourth section reviewed the new DALC application.

The next chapter concludes this study. It provides a short overview of the project and an analysis of the results.

5 Conclusions

This project aimed to improve an existing OPC data-logging system. The main points of focus were on improving the performance and reliability of the system. The initial system was lacking in several areas. The hardware component was lacking in terms of performance and reliability, the database schema was not properly designed, the DBMS was not properly configured, and the DALC was not very user-friendly and lacked some key features. In response to these problems this study was performed and the rest of the chapter is devoted to discussing the outcomes and conclusions thereof.

5.1 The Contribution of the Project

When this project was initiated, several problems were identified. The problems are an insufficient hardware infrastructure, a DBMS that was not properly configured, a badly designed database schema, and a difficult to use DALC application.

The last two problems only influenced the data-logging system; however the first two problems affected the entire information system of the component-handling platform. The new servers, upgraded network, UPS, and backup system improved the overall performance and reliability of the entire information system. The new database server, with its RAID 10 configuration, provides a reliable platform for hosting the DBMS. It also allowed for the proper configuration of the DBMS, ensuring that the DBMS made proper use of the available resources. The backup system ensures that the entire information system is backed-up regularly and that the data can be recovered quickly.

The redesigned database schema not only resulted in a more compact database, but also serves as a guide to designing future schemas. Lastly the new DALC is an easy-to-use application that can quickly be configured to log the data from multiple OPC servers. The use of bulk record INSERT statements also proved to be an effective means of writing many records to disk; thus effectively increasing the performance of the DALC.

5.2 Contributions by other students

The Web server and Application server were built, installed and managed by fellow students of the research group. These servers were briefly discussed due to the integrated nature of the implemented system. Management tasks included setting-up the backups, creating various virtual machines that would be used to develop and test various applications, and the design and creation of the RGEMS website. As mentioned in section 4.1, several additional changes were also made to the network by fellow students.

5.3 Recommendations

To enhance the performance and reliability of the OPC data-logging system new dedicated servers were built and installed, the database was redesigned, the DBMS was properly configured, and the data-logging client was rewritten. Several benchmarks were also performed to confirm the validity of the changes.

In light of the obtained results the following recommendations are made:

- By using a dedicated database server, one can configure the DBMS to take full advantage of the available resources. One example is that more data and indexes can be cached in memory, reducing disk access; this in turn will result in faster query turnaround time.
- In a four-disk RAID configuration RAID 10 offers the best balance between performance and reliability. However, if effective storage capacity is more important than performance, a RAID 5 configuration might be considered. In a rare scenario where performance is of the utmost importance and data redundancy and integrity is not important, one may consider a RAID 0 configuration.
- Avoid using integrated hybrid RAID controllers in critical systems where downtime is not accepted.
- Separating the database's data and the OS and storing them on separate disks or disk arrays has several advantages:

- If the OS were to fail due to hardware failure or software corruption, the database's data would still remain intact.
- In the above-mentioned scenario restoring the system disk from backups should, in most cases, be faster than restoring the entire database.

- Having a proper backup strategy is absolutely vital for any environment where data is involved. It will ensure that data can be easily recovered in case of accidental deletion, corruption or hardware failure.
- Windows Server Backup can be very flexible if it is used with command-line scripting and the Task Scheduler. However it remains only a basic backup tool and for certain scenarios a professional third-party solution may be required. One such scenario may be where multiple NAS devices form a critical part of the backup strategy.
- A UPS is a vital component of any critical computer system. It is unlikely that a UPS will be used to allow continued operation during an extensive blackout; furthermore a UPS will be expensive. In most cases it will be configured to safely shutdown the servers, or keep the servers running for a short period of time until a generator has started up. In both cases it will also prevent data loss, due to a sudden power failure.
- MySQL offers great flexibility with its different storage engines. However if relational integrity is required, InnoDB is the only option at this time.
- Bulk record insert transactions do offer some improvement over single transactions; however for MyISAM and Archive tables the performance difference is negligible with smaller batches. InnoDB tables stand to gain the most from bulk inserts, with a significant performance difference apparent in all four benchmarks.
- Optimising queries can take some time, but for those queries that are frequently run, it can significantly reduce execution time. For queries that are seldom run it may not be worthwhile to go through the entire optimisation process.

- The *innodb_flush_log_at_trx_commit* variable has the greatest effect on the write performance. It is recommended to set its value to 2, as it offers a good compromise between performance and integrity. The UPS and backup battery of the RAID controller compensates for the slight loss of reliability, as they prevent data loss due to power failures. This leaves the unlikely event of an OS failure as a possible weakness. A value of 0 is to be avoided at all cost, as this will cause data loss in the event of a *mysqld* crash.
- For security reasons the DALC will be run on the same computer or virtual machine as the OPC servers and connected remotely using a TCP/IP connection to the database.

5.4 Future Work

There are several more possible improvements that can be made to the system. These are discussed below:

The next version of the DALC can be migrated to a windows service so that it can start automatically even if a user is not logged into the computer. Another possible improvement may be to migrate it from DA to the newer and cross-platform compatible XML DA or UA. This will make it possible to create a version of the DALC that is OS independent.

For the database there are only a few possible improvements. The latest release of MySQL 5.1 includes the new InnoDB plug-in that offers a smaller table format and increased performance. If the archived logs are accessed regularly, it may be wise to either increase the period that data remains in the active log table or add a third MyISAM based table to store more recent data. A third option, for servers with a lot of RAM, is to preload frequently accessed data into a MEMORY base's table and then to access it from there. It has been noted in the MySQL manual that future versions may allow table partitions to use different storage engines; this will allow the current log table implementation and the above-mentioned schemes, with exception to the last, to be implemented using one table.

In future the MARIA storage engine could also be an option as a primary storage engine. The MARIA is available as part of the MARIA Database (DB) project [**56**], a fork of the main MySQL branch. MARIA aims to offer a storage engine with the advantages of both MyISAM and InnoDB.

5.5 Conclusion

In an attempt to increase the performance and reliability of the data-logging system as a whole and its individual component, the shortcomings of the data-logging system were identified. Relevant technologies that could be used to improve the performance and reliability were evaluated, and a new and improve system was designed and implemented.

The new hardware infrastructure that was installed during the course of this study provided a stable, high performance and reliable platform for the entire information system. The new database server with its dedicated RAID configuration allows fast and reliable data storage. The RAID 10 configuration also proved itself on more than one occasion. When one or even two disks in the array failed, operation was able to continue.

The new backup and power management strategies further improved the reliability of the overall system. Backups ensure that critical data can easily be restored. The majority of the backup system is automated, and only requires the offsite backup disk to be plugged-in once a week. The fully automated power management system protects the system from power failures and fluctuations.

The database schema is now much more compact. The main tables are based on the InnoDB storage engine, as it enforces referential integrity and offers better data security than MyISAM. The Archive storage engine, used for the archived log table, ensures that older log files will be stored efficiently and are easily accessible. Unfortunately table partitioning could not be applied to three main tables, due to a limitation concerning InnoDB's foreign key support. However table partitioning was applied to the archive log table.

Using the InnoDB storage engine improved the reliability of the database schema, but it proved to be the slowest of the three engines that were evaluated. After configuring the MySQL DBMS for an InnoDB based load, the engine's performance increased significantly.

The new DALC now allows for OPC server and item browsing on a specified computer. This made the application much more user-friendly. It offers slightly better performance by making use of bulk record INSERT statements.

It can be concluded that any information system requires a proper hardware platform, and that one can significantly improve the reliability of the entire system by implementing a backup strategy. Additionally, by adding a UPS of sufficient capacity and properly configuring its software, you can prevent data loss due to power failures. It is also vital to properly configure the DBMS to allow it to properly make use of the available resources.

Appendix A INSERT Query Benchmark Scripts

```
@ECHO OFF


ECHO Archive >> insert_benchmark.log
ECHO Query = bulkinsert10 >> insert_benchmark.log


mysqlslap --user=<username> --password=<passwaord> --create-schema=<schemaname> --query="call
bulkinsert10();" --iterations=10 >> insert_benchmark.log


mysql -u<username> -p<password> <schemaname> -e "ALTER TABLE inserttest ENGINE = MYISAM;"
mysql -u<username> -p<password> <schemaname> -e "truncate inserttest;"


ECHO MyISAM >> insert_benchmark.log
ECHO Query = bulkinsert10 >> insert_benchmark.log


mysqlslap --user=<username> --password=<passwaord> --create-schema=<schemaname> --query="call
bulkinsert10();" --iterations=10 >> insert_benchmark.log


mysql -u<username> -p<password> <schemaname> -e "ALTER TABLE inserttest ENGINE = INNODB;"
mysql -u<username> -p<password> <schemaname> -e "truncate inserttest;"


ECHO InnoDB >> insert_benchmark.log
ECHO Query = bulkinsert10 >> insert_benchmark.log


mysqlslap --user=<username> --password=<passwaord> --create-schema=<schemaname> --query="call
bulkinsert10();" --iterations=10 >> insert_benchmark.log


mysql -u<username> -p<password> <schemaname> -e "truncate inserttest;"
mysql -u<username> -p<password> <schemaname> -e "ALTER TABLE inserttest ENGINE = ARCHIVE;"


ECHO Archive >> insert_benchmark.log
ECHO Query = singleinsert10 >> insert_benchmark.log


mysqlslap --user=<username> --password=<password> --create-schema=<schemaname> --query="call
```

Figure A-1 Benchmark Script Template Used to Compare the Performance of Single Record and Bulk Record INSERT Statements

```
DELIMITER $$

USE `test`$$

DROP PROCEDURE IF EXISTS `singleinsert10`$$

CREATE DEFINER=`root`@`%` PROCEDURE `singleinsert10`()
BEGIN
    INSERT INTO inserttest VALUES ('61','1274862245406A0','jnoyhwhca',NOW());
    INSERT INTO inserttest VALUES ('100','1274862245406B1','pajzl',NOW());
    INSERT INTO inserttest VALUES ('82','1274862245406C2','fanioctpu',NOW());
    INSERT INTO inserttest VALUES ('5','1274862245406D3','jtfdpkrv',NOW());
    INSERT INTO inserttest VALUES ('56','1274862245406E4','aqdcoey',NOW());
    INSERT INTO inserttest VALUES ('77','1274862245406F5','ewhwbrisr',NOW());
    INSERT INTO inserttest VALUES ('91','1274862245406G6','tqdhafefx',NOW());
    INSERT INTO inserttest VALUES ('44','1274862245406H7','vpzwm',NOW());
    INSERT INTO inserttest VALUES ('87','1274862245406I8','onnilr',NOW());
    INSERT INTO inserttest VALUES ('20','1274862245406J9','lfzgqw',NOW());
    END$$

DELIMITER ;
```

Figure A-2 Stored procedure to execute 10 separate single record INSERT statements

```
DELIMITER $$

USE `test`$$

DROP PROCEDURE IF EXISTS `bulkinsert10`$$

CREATE DEFINER=`root`@`%` PROCEDURE `bulkinsert10`()
BEGIN
    INSERT INTO inserttest VALUES
    ('61','1274862245406A0','jnoyhwhca',NOW()),
    ('100','1274862245406B1','pajzl',NOW()),
    ('82','1274862245406C2','fanioctpu',NOW()),
    ('5','1274862245406D3','jtfdpkrv',NOW()),
    ('56','1274862245406E4','aqdcoey',NOW()),
    ('77','1274862245406F5','ewhwbrisr',NOW()),
    ('91','1274862245406G6','tqdhafefx',NOW()),
    ('44','1274862245406H7','vpzwm',NOW()),
    ('87','1274862245406I8','onnilr',NOW()),
    ('20','1274862245406J9','lfzgqw',NOW());
    END$$

DELIMITER ;
```

Figure A-3 Stored procedure to execute a bulk record INSERT statement with 10 records

Appendix B SELECT Query Benchmark

```
@ECHO OFF
REM Replace Query_Path the location of the query file. i.e. C:\benchmarks.
REM Query.sql contains the query that is to be used in the benchmark.
REM Replace UserName and Password witH those of the database user account
REM that must be used
REM Change the value for --concurrency to change number of concurrent REM queries
REM Change the value for --iterations=1 to change number of times the REM benchmark must
be run.
```

Figure B-1 Select Query Benchmark Template

Appendix C Server System Variable Benchmark Scripts

```
@ECHO OFF
REM +-----------------------------------------------------------------------------------------+
REM | Command              |     Decsription                                                   |
REM |----------------------+-------------------------------------------------------------------|
REM | mysqlslap            |     will execute the mysqlslap benchmark application with the    |
REM |                      |     given parameters                                              |
REM +----------------------+-------------------------------------------------------------------|
REM | --user               |     specifies the database user to use                            |
REM +----------------------+-------------------------------------------------------------------|
REM | --password           |     specifies the password for the given user                     |
REM +----------------------+-------------------------------------------------------------------|
REM | --concurrency        |     The number of user connections to simulate                    |
REM +----------------------+-------------------------------------------------------------------|
REM | --create-schema      |     specifies the schema in wich to run the benchmarks            |
REM +----------------------+-------------------------------------------------------------------|
REM | --query              |     The query to execute                                          |
REM +----------------------+-------------------------------------------------------------------|
REM | --iterations         |     The number of time to repeat the process                      |
REM +----------------------+-------------------------------------------------------------------|
REM | mysql                |     executes the mysql command line client with the               |
REM |                      |     given parameters                                              |
REM +----------------------+-------------------------------------------------------------------|
REM | -u                   |     specifies the database user to use                            |
REM +----------------------+-------------------------------------------------------------------|
REM | -p                   |     specifies the password for the given user                     |
REM +----------------------+-------------------------------------------------------------------|
REM | test                 |     specifies the schema to use, in this case the "opclog" schema |
REM +----------------------+-------------------------------------------------------------------|
REM | -e                   |     executes the following statment and exits                     |
REM +----------------------+-------------------------------------------------------------------|
REM | >>                   |     Out redirect ouput to a specified file                        |
REM +-----------------------------------------------------------------------------------------+

mysqlslap --user=root --password=[illegible] --concurrency=50 --create-schema=opclog --query="CALL
NormalInsert(5000);" --iterations=3 >> mysqld_benchmark.log

mysql -uroot -p[illegible] opclog -e "truncate tblitemlog_1;"
```

Figure C-1 Script Used to Implement Part A of the First Benchmark Process

```
@ECHO OFF
REM +-------------------------------------------------------------------------------------+
REM | Command              |     Decsription                                                |
REM |----------------------+-------------------------------------------------------------|
REM | mysqlslap            |     will execute the mysqlslap benchmark application with the |
REM |                      |     given parameters                                          |
REM +----------------------+-------------------------------------------------------------|
REM | --user               |     specifies the database user to use                        |
REM +----------------------+-------------------------------------------------------------|
REM | --password           |     specifies the password for the given user                 |
REM +----------------------+-------------------------------------------------------------|
REM | --concurrency        |     The number of user connections to simulate                |
REM +----------------------+-------------------------------------------------------------|
REM | --create-schema      |     specifies the schema in wich to run the benchmarks        |
REM +----------------------+-------------------------------------------------------------|
REM | --query              |     The query to execute                                      |
REM +----------------------+-------------------------------------------------------------|
REM | --iterations         |     The number of time to repeat the process                  |
REM +----------------------+-------------------------------------------------------------|
REM | mysql                |     executes the mysql command line client with the           |
REM |                      |     given parameters                                          |
REM +----------------------+-------------------------------------------------------------|
REM | -u                   |     specifies the database user to use                        |
REM +----------------------+-------------------------------------------------------------|
REM | -p                   |     specifies the password for the given user                 |
REM +----------------------+-------------------------------------------------------------|
REM | test                 |     specifies the schema to use, in this case the "opclog" schema |
REM +----------------------+-------------------------------------------------------------|
REM | -e                   |     executes the following statment and exits                 |
REM +----------------------+-------------------------------------------------------------|
REM | >>                   |     Out redirect ouput to a specified file                    |
REM +-------------------------------------------------------------------------------------+


mysqlslap --user=root --password=[illegible] --concurrency=50 --create-schema=opclog --query="CALL
BulkInsert(5);" --iterations=3 >> mysqld_benchmark.log

mysql -uroot -p[illegible] opclog -e "truncate tblitemlog_1;"
```

Figure C-2 Script Used to Implement Part A of the Second Benchmark Process

Appendix D SQL Scripts to Create the Database Schema

```
CREATE DATABASE IF NOT EXISTS`log`;

USE `log`;

/*Table structure for table `tblarchivelog` */

CREATE TABLE `tblarchivelog` (
  `ItemId` int(11) unsigned NOT NULL,
  `Value` varchar(50) NOT NULL,
  `AcquisitionDateTime` datetime DEFAULT NULL,
  `Quality` varchar(30) NOT NULL
) ENGINE=ARCHIVE DEFAULT CHARSET=utf8
PARTITION BY RANGE ( TO_DAYS(AcquisitionDateTime))
(PARTITION pNULL VALUES LESS THAN (0) ENGINE = ARCHIVE,
 PARTITION p0 VALUES LESS THAN (733588) ENGINE = ARCHIVE,
 PARTITION p1 VALUES LESS THAN (733772) ENGINE = ARCHIVE,
 PARTITION p2 VALUES LESS THAN (733953) ENGINE = ARCHIVE,
 PARTITION p3 VALUES LESS THAN (734137) ENGINE = ARCHIVE,
 PARTITION p4 VALUES LESS THAN (734318) ENGINE = ARCHIVE,
 PARTITION p5 VALUES LESS THAN (734502) ENGINE = ARCHIVE,
 PARTITION p6 VALUES LESS THAN MAXVALUE ENGINE = ARCHIVE);

/*Table structure for table `tblitemlog_1` */

CREATE TABLE `tblitemlog_1` (
  `LogId` bigint(20) unsigned NOT NULL AUTO_INCREMENT,
  `ItemId` int(11) unsigned NOT NULL,
  `Value` varchar(100) CHARACTER SET latin1 DEFAULT NULL,
  `AcquisitionDateTime` varchar(23) CHARACTER SET latin1 DEFAULT NULL,
  `Quality` varchar(30) CHARACTER SET latin1 DEFAULT NULL,
  PRIMARY KEY (`LogId`,`ItemId`),
  KEY `FK_tblrealtimelog` (`ItemId`),
  CONSTRAINT `FK_tblrealtimelog` FOREIGN KEY (`ItemId`) REFERENCES `tblopcitems` (`ItemId`)
) ENGINE=InnoDB AUTO_INCREMENT=16809 DEFAULT CHARSET=utf8;

/*Table structure for table `tblopcitems` */

CREATE TABLE `tblopcitems` (
  `ItemId` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `ItemShortName` varchar(50) CHARACTER SET latin1 DEFAULT NULL,
```

```
  `ItemFullName` varchar(255) CHARACTER SET latin1 DEFAULT NULL,
  `DataType` varbinary(30) DEFAULT NULL,
  `ServerId` int(11) unsigned NOT NULL,
  PRIMARY KEY (`ItemId`),
  KEY `FK_tblopcitems` (`ServerId`),
  CONSTRAINT `FK_tblopcitems` FOREIGN KEY (`ServerId`) REFERENCES `tblopcservers` (`ServerId`)
) ENGINE=InnoDB AUTO_INCREMENT=1015 DEFAULT CHARSET=utf8;

/*Table structure for table `tblopcservers` */

CREATE TABLE `tblopcservers` (
  `ServerId` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `ServerInstanceName` varchar(100) CHARACTER SET latin1 DEFAULT NULL,
  `URL` varchar(200) CHARACTER SET latin1 DEFAULT NULL,
  `IPv4` varchar(39) CHARACTER SET latin1 DEFAULT NULL,
  `Manufacturer` varchar(100) CHARACTER SET latin1 DEFAULT NULL,
  `Version` varchar(20) CHARACTER SET latin1 DEFAULT NULL,
  PRIMARY KEY (`ServerId`)
) ENGINE=InnoDB AUTO_INCREMENT=12 DEFAULT CHARSET=utf8;
```

References

[1] Bothma, B.C. and Vermaak, H.J., "Performance and Reliability Optimization of a Data Acquisition and Logging System in an Integrated Component-Handling Environment.," in *2nd Robotics & Megatronics Symposium*, Bloemfontein, South Africa, 2008, pp. 88-92.

[2] Sun Microsystems. (June 23, 2009). "MySQL 5.1 Reference Manual (Revision 15371)," Available: http://downloads.mysql.com/docs/refman-5.1-en.a4.pdf, last accessed in June 2009.

[3] *Setting Up the world Database*, http://dev.mysql.com/doc/world-setup/en/world-setup.html, last accessed in November 2009.

[4] Gilfillan, Ian. (Nov., 2001). "Optimizing MySQL: Queries and Indexes," Available: http://www.databasejournal.com/features/mysql/article.php/10897_1382791_1/Optimizing-MySQL-Queries-and-Indexes.htm, last accessed in June 2008.

[5] Schumacher, Robin. (May, 2006). "More on MySQL 5.1 Partitioning," Available: http://dev.mysql.com/tech-resources/articles/mysql_5.1_partitioning.html, last accessed in March 2010.

[6] Howard, Rob. (August 28, 2008). "Don't use stored procedures yet? Must be suffering from NIHS (Not Invented Here Syndrome)," Available: http://grokable.com/don-t-use-stored-procedures-yet-must-be-suffering-from-nihs-not-invented-here-syndrome/, last accessed in June 2009.

[7] Bouma, Frans. (November 18, 2003). "Stored procedures are bad, m'kay?," Available: http://weblogs.asp.net/fbouma/archive/2003/11/18/38178.aspx, last accessed in June 2009.

[8] *MySQL Stored Procedures problems and use practices*, http://www.mysqlperformanceblog.com/2007/06/12/mysql-stored-procedures-problems-and-use-practices/, last accessed in 2009 July.

[9] *How fast (or slow) is MySQL Stored Procedure language?*, http://mtocker.livejournal.com/45222.html, last accessed in March 2010.

[10] MySQL AB, *MySQL Administrator's Guide and Language Reference*, Second Edition ed., Seattle: MySQL Press, 2006.

[11] MySQL AB. (May, 2006). "MySQL 5.0's Pluggable Storage," Available: http://www.mysql.com/why-mysql/white-papers/mysql_wp_pluggable.php, last accessed in July 2008.

[12] MySQL AB. (2007). "Enterprise Data Warehousing," Available: http://www.mysql.com/why-mysql/white-papers/mysql_wp_for_data_warehousing.pdf, last accessed in June 2008.

[13] Zaitsev, Peter; Asplund, Tobias; , MySQL AB. (2005). "Advanced MySQL Performance Optimization," Available: http://www.mysqlperformanceblog.com/files/presentations/UC2005-Advanced-MySQL-Performance-Optimization.pdf, last accessed in June 2008.

[14] MySQL AB. (2008). "Inside MySQL 5.1 - A DBA's Perspective," Available: http://www.mysql.com/why-mysql/white-papers/whatsnew-mysql-51.php, last accessed in March 2009.

[15] MySQL AB. (October, 2005). "Inside MySQL 5.0," Available: http://mysql.com/why-

mysql/white-papers/mysql_wp_inside50.php, last accessed in July 2008.

[16] Tuuri, Heikki and Jacobs, Ken, "InnoDB: Fast, Reliable, Proven Transactional Storage for MySQL," presented at the *2008 MySQL Conference & Expo*, Santa Clara, California, USA, April 2008.

[17] MySQL AB. (June, 2008). "Guide to MySQL 5.1 Partitioning," Available: http://mysql.com/why-mysql/white-papers/mysql_wp_partitioning.php, last accessed in November 2008.

[18] Schumacher, Robin. (Feb. 22, 2006). "Improving Database Performance with Partitioning," Available: http://dev.mysql.com/tech-resources/articles/performance-partitioning.html, last accessed in August 2009.

[19] Özsu, M. Tamer and Valduriez, Patrick, *Principles of Distributed Database Systems*, 2nd ed., 1999.

[20] *What to tune in MySQL Server after installation*, http://www.mysqlperformanceblog.com/2006/09/29/what-to-tune-in-mysql-server-after-installation/, last accessed in June 2009.

[21] Bester, Shane. (Nov., 2006). "Bug #24509: 2048 file descriptor limit on windows needs increasing," Available: http://bugs.mysql.com/bug.php?id=24509, last accessed in February 2010.

[22] Patterson, David A, Gibson, Garth and Katz, Randy H, "A Case for Redundant Arrays of Inexpensive Disks (RAID)," *SIGMOD Record*, vol. 17, no. 3, pp. 109-116, 6 1988.

[23] Chen, P. M., Lee, E. K., Gibson, G. A., Katz, R. H. and Patterson, D. A., "RAID: high

performance, reliable seconday storage," *ACM Comput. Surv.*, vol. 26, no. 2, pp. 145-185, June 1994.

[24] Rockwood, Ben. (Apr., 2002). "RAID Theory: An Overview," Available: www.cuddletech.com/veritas/raidtheory/raidtheory.pdf, last accessed in November 2008.

[25] LaCie. (2010). "RAID Technology White Paper," Available: http://www.lacie.com/download/whitepaper/WP_RAID_EN.pdf, last accessed in March 2010.

[26] Persson, Jimmy and Evertsson, Gustav. (October, 2002). "RAID Systems," Available: http://www.guzzzt.com/files/coding/raid.pdf, last accessed in June 2008.

[27] AMCC STORAGE. (2005). "RAID 6 Techbrief," Available: http://www.3ware.com/products/pdf/RAID_6_techbrief_112206.pdf, last accessed in January 2009.

[28] Eagle Rock Alliance, LTD. (2001). "2001 Cost of Downtime Online Survey," Available: http://www.contingencyplanningresearch.com/2001%20Survey.pdf, last accessed in November 2009.

[29] Computer Associates International. (June, 2005). "Best Practices: Backup and Recovery Strategies," Available: http://ca.com/files/whitepapers/backup_recov_wp.pdf, last accessed in January 2010.

[30] Horlings, Jeroen. (July, 2004). "PC Active," Available: http://www.pc-active.nl/component/content/article/10508, last accessed in August 2009.

[31] Lauzet, Justin C. (August, 2006). "True Cost of Archiving to CDs/DVDs," Available:

http://www.wiebetech.com/whitepapers/true_cost_of_cds.php, last accessed in November 2009.

[32] *Desktop Storage Overwiev*, http://www.westerndigital.com/en/products/index.asp?cat=8, last accessed in February 2010.

[33] EMC Corporation. (October, 2006). "Backing Up to External Hard Drives," Available: http://www.retrospect.com/assets/wp_extrnlhds_en.pdf, last accessed in November 2009.

[34] *WD (Western Digital) Interface Guide*, http://www.wdc.com/en/products/resources/DriveCompatibilityguide.asp#external, last accessed in June 2009.

[35] Serial ATA International Organization. (2004). "eSATA," Available: http://www.serialata.org/technology/esata.asp, last accessed in January 2009.

[36] Seagate. (May, 2009). "BlackArmor® NAS 440/420 Technology Paper," Available: http://www.seagate.com/docs/pdf/whitepaper/TP604_1_blackarmor_NAS440.pdf, last accessed in November 2009.

[37] Western Digital. (January, 2009). "WD (Western Digital) ShareSpace Product Overview," Available: http://www.wdc.com/wdproducts/library/AAG/ENG/4178-705023.pdf, last accessed in November 2009.

[38] IBM Corporation. (October, 2008). "IBM System Storage TS1130 Tape Drive," Available: ftp://public.dhe.ibm.com/common/ssi/pm/sp/n/tsd03053usen/TSD03053USEN.PDF, last accessed in February 2010.

[39] IBM Corporation. (October, 2006). "IBM 3592 Tape Cartridge," Available:

ftp://public.dhe.ibm.com/common/ssi/pm/sp/n/tsd00063usen/TSD00063USEN.PDF, last accessed in February 2010.

[40] Tandberg Data Corporation. (2007). "Tandberg Data SMB Guide to Backup Best Practices," Available: http://www.exabyte.com/support/online/documentation/whitepapers/basicbackup.pdf, last accessed in July 2008.

[41] *Navicat for MySQL: Overview*, http://www.navicat.com/en/products/navicat_mysql/mysql_overview.html, last accessed in May 2009.

[42] Intel®. (August, 2007). "Intel® Core™2 Extreme Quad-Core Processor QX6000Δand Intel® Core™2 Quad Processor Q6000ΔSequence Datasheet," Available: http://download.intel.com/design/processor/datashts/31559205.pdf, last accessed in May 2009.

[43] Adaptec. (March, 2007). "Adaptec PCIe SATA and SAS RAID Controller Family Datasheet," Available: http://www.adaptec.com/NR/rdonlyres/94879AF5-C608-40AA-88AF-568826618162/0/4636_PCIeFamily_20.pdf, last accessed in May 2009.

[44] *SiSoftware Sandra*, http://www.sisoftware.net/, last accessed in November 2008.

[45] *Microsoft Technet - Wbadmin*, http://technet.microsoft.com/en-gb/library/cc754015%28WS.10%29.aspx, last accessed in March 2010.

[46] *navicat*, http://www.navicat.com/, last accessed in June 2008.

[47] *Microsotf Technet - Configure Automatic Backups with Task Scheduler*,

http://technet.microsoft.com/en-us/library/dd834883.aspx, last accessed in February 2010.

[48] PremiumSoft, Navicat for MySQL 8.X Help, 2009, Included with Navicat.

[49] Bothma, B.C. and Vermaak, H.J., "Final Implementation of an Improved OPC Data Logging System in an in a Automation Environment," in *13th Annual Research Seminar of the Faculty of Engineering and Information & Communications Technology, CUT*, Bloemfontein, South Africa, October 2010.

[50] *Windows Sysinternals - PsTools*, http://technet.microsoft.com/en-us/sysinternals/bb896649.aspx, last accessed in August 2009.

[51] Sun Microsystems, Inc. (July, 2009). "MySQL Administrator Manual," Available: http://downloads.mysql.com/docs/administrator-en.a4.pdf.

[52] Bothma, B.C. and Vermaak, H.J., "Configuring a MySQL database for a general purpose database in an Automation Environment," in *12th Annual Research Seminar of the Faculty of Engineering and Information & Communications Technology, CUT* , Bloemfontein, October 2009.

[53] *Intel® G45 Express Chipset Overview*, http://www.intel.com/products/desktop/chipsets/g45/g45-overview.htm, last accessed in October 2008.

[54] Vermaak, Herman, Weppenaar, De Ville, Bothma, Tian and Janse van Rensburg, Jean, "Information and Maintenace Management System," presented at the *AMTS Projects Symposium* , Midrand, Johannesburg, South Africa, September 2009.

[55] Deiretsbacher, Karl-Heinz, Luth, Jim, Mody, Rashesh and Haus, Kurt T. (March, 2009).

"Using OPC via DCOM with Windows XP Service Pack 2," Available: http://opcfoundation.org/DownloadFile.aspx?CM=3&RI=326&CU=1.

[56] *MariaDB*, http://mariadb.org/, last accessed in June 2010.