Mwai Karimi

Benchmarking of RDBMS and NoSQL Performance on Unstructured Data

2018

Mwai Karimi

Benchmarking of RDBMS and NoSQL Performance on Unstructured Data

Mestrado em Engenharia Informática
Trabalho efetuado sob a orientação da:
Professora Paula Ventura.

2018

BENCHMARKING OF RDBMS AND NOSQL PERFORMANCE ON UNSTRUCTURED DATA

Declaração de autoria de trabalho

Declaro ser o(a) autor(a) deste trabalho, que é original e inédito. Autores e trabalhos consultados estão devidamente citados no texto e constam da listagem de referências incluída.

Assinatura do candidato: _____

Abstract

New requirements are arising in the database field. Big data has been soaring.
The amount of data is ever increasing and becoming more and more varied. Traditional relational database management systems have been a dominant force in the database field but due to the massive growth of unstructured and multiform data, firms are now turning to architectures that have scale-out capabilities using open source software, commodity servers, cloud computing and services like Database as a Service. Due to this, relational databases ought to adopt and meet these new data requirements with easier and faster data processing capabilities and also provide multiple analytical tools that have the possibility of displaying analytics instantly. This study aims to benchmark the performance of relational systems and NoSQL systems on unstructured data.

Keywords
Relational database, NoSQL, big data, MySQL, MongoDB

Resumo

Novos requisitos estão surgindo na área das bases de dados. “Big data” permitiu avanços consideráveis em vários setores.
O volume de dados tem aumentado e torna-se cada vez mais variado. Os sistemas tradicionais de gestão de base de dados relacionais têm sido uma força dominante na área, mas devido ao crescimento massivo de dados não estruturados e multiformes, as empresas agora recorrem a arquiteturas que possuem recursos escaláveis usando software livre, servidores, computação em nuvem e serviços, tais como “base de dados como um serviço”. Nesse sentido, as bases de dados relacionais devem considerar e adotar novos requisitos de dados com maior agilidade no seu processamento e também fornecer múltiplas ferramentas analíticas com a possibilidade de mostrar análises em tempo real. Este estudo tem como objetivo avaliar o desempenho de sistemas relacionais e sistemas NoSQL em dados não estruturados.

Palavras-chave
Base de dados relacional, NoSQL, big data, MySQL, MongoDB

Acknowledgments

I want to thank the academic staff of the University of the Algarve, who have been of tremendous help during my learning period at the university. Special thanks go to my supervisor, Professor Paula Ventura, for her advice and guidance.

I would also love to thank portal 47 for affording me a glorious opportunity to do my traineeship at their company. Without them, this research work wouldn’t have been possible.

Last but not least, I thank my family for their unwavering and steadfast love and support during the whole period of my study.

Table of Contents

Index of Figures

Index of Tables

Abbreviations

ETL	Extraction, Transform, Load
API	Application Programming Interface
UX	User Interface
XML	Extensible Markup Language
JSON	JavaScript Object Notation
RDBMS	Relational Database Management System
SQL	Structured Query Language
ACID	Atomicity, Consistency, Isolation, Durability
BASE	Basically Available, Soft State, Eventual Consistency
CAP	Consistency, Availability, Partition Tolerance
OTTR	Observe, Think, Test, Review
YCSB	Yahoo! Cloud Serving Benchmark

Introduction and Motivation

With the current massive increase in data, it becomes inevitable to adapt to new technologies in order to store data in natural formats. Relational systems mostly store structured data. Structured data is data that is organized or that is contained in fixed fields. This data has to be mapped in pre-designed field names or what is normally referred to as schemas [1][2]. Unstructured data has no predefined model or schema. This data can be found in weblogs, social media data, multimedia content, sales automation, customer interactions, and even emails. Companies and organizations are continuously producing more and more of this data from their activities. This data is pushing relational systems to their limits creating the need for a replacement on legacy infrastructure. Relational systems follow an inflexible structure to store and organize their data [3]. It is due to this structure, that relational systems cannot scale effectively. This often inhibits the agility of businesses.

While doing my internship at Portal 47, a company with a heavy reliance on relational infrastructure, I experienced several performance issues that highlight the limits of relational database systems when working with unstructured data. Some of the main problems experienced were as a result of executing queries with aggregation functions on large data sets. While working with such queries, operations became slow to impractical speeds once the dataset went beyond trivial, for example, a few hundred thousand rows, which meant having to create extract, transform and load (ETL) processes to aggregate long-running calculations. This was costly. Moreover, scaling was slow as the hardware could only scale to a certain limit. To optimize query performance, indexes were used but were still quite slow due to the size of the data sets. Application integration was delayed because in some instances, schema changes had to be made. This, in general, delayed the development process. Due to some of these problems, the company is looking at non-relational systems in order to handle some of their multi-structured data types and also be able to scale beyond the capacity of their existing systems.

There has also been a need to adopt agile development as new applications which continue producing unstructured data are being developed differently with different agile methods [4]. Development is also happening in agile sprints as opposed to the traditional waterfall methods [5]. This means that in place systems need to have the ability to meet the current development needs and also be able to handle the output of massive data.

Since their inception in 2009, NoSQL systems have become favoured due to their flexibility and ability to handle large unstructured data [6]. They have different features to relational systems but could still be used for similar applications. However, this does not mean that one is supposed to be a substitute for the other. This study will benchmark performance between relational and NoSQL systems. Comparing these two will enable us to draw conclusions with regard to how fast they process data and how they handle large amounts of it. These conclusions can help us access the necessity of migrating from a relational to a NoSQL system. It would be fundamentally important to know which database works best for a particular application as making a bad choice could have devastating effects.

Scope and Limitations

The internship conducted at Portal 47 focussed on equipping the trainee with the fundamentals of data analysis and engineering. To that end, I interacted with database systems, different programming languages and had a first-hand experience of working in an agile environment.

The scope area of this study is benchmarking performance in relational and NoSQL database systems. The databases used were MySQL and MongoDB. The metrics used are core features of both database systems. The study investigated index performance, union of tables and collections, application integration and scalability of both databases.

Although this study was executed successfully, some limitations were inevitable. First of all, the research was conducted with a dataset of up to 2 million rows. To mimic a real production environment, we would need a huge dataset spanning more than 100 million rows.
In addition to that, the infrastructure was inadequate. Due to a lack of hardware resources, the databases could not be tested extensively as you can only perform operations to a certain limit depending on CPU and memory limits. Better hardware would have allowed for more extensive and comprehensive tests.

Goals

One of the main goals of my internship was to fulfil my academic requirement, which would eventually culminate in the research work. Other goals included training and equipping me with the fundamentals of data analysis, learning the best practices of database optimization, agile development and solving real-world problems with data science.

The main goal of the research work was to benchmark the performance of relational and NoSQL systems on unstructured data, in particular, MongoDB and MySQL. This would enable us to understand whether migration from a relational system would be worth it. The specific objectives were to understand how indexes affect read operations, how both systems differ when it comes to joining different tables or collections in the same database, how fast both systems are able to perform read and write operations through third-party drivers and last but not least how both systems perform when replicating data.

Main Contributions

Past research done on the comparison of database systems has concluded that NoSQL systems are the go-to systems when it comes to big data and high throughput web applications. This research work aims to contribute to that narrative by including new performance metrics. Thus, this study avails a unique perspective on the comparison of

relational and NoSQL systems by furthering our knowledge of the core features of these database systems.

Structure of the Thesis

This thesis is organized in 2 parts which make up 8 chapters. The first part talks about my internship at Portal 47 Ltd. Chapter 1 gives a basic introduction to the company and details about the position I was offered. Chapter 2 covers the tools that I worked with during this period while chapter 3 gives describes the work plan, tasks that I performed and a breakdown of what was done for each month. Chapter 4 gives an assessment of the internship with a final conclusion.

The second part expounds on the research work conducted. Chapter 5 discusses the methodology used for the research work while chapter 6 talks about relational and NoSQL systems and their models and gives a comparison of both systems. Chapter 7 covers the metrics that were used in the research work while chapter 8 illustrates the case study and results obtained. Chapter 9 concludes with a conclusion of the research work and discusses possible future work.

Chapter 1

Internship

The following section describes my internship at Portal 47 Ltd. My traineeship period was from the 27th of June 2017 until the 22nd of December 2017. The discussion centres around the company, work plan, tasks given, detailed internship progress and skills acquired after the traineeship.

1.1 Host Organisation

Portal 47 is a real estate portal company that has been in existence since 2003 and is located in Bath, United Kingdom. The company runs an online real estate platform called Kyero[7], with the main focus being the selling of property in Spain and now recently in Portugal. The company also acts as an advisor to property developers, agents, buyers, and sellers. Portal 47 has close to 30 employees spread across the UK, Spain and Portugal. The company is divided into 4 departments: Data, Marketing, Design and Engineering and Administration.

The data team is in charge of the technical and data operations. This includes architecting systems and data stores, building data pipelines and systems programming. The marketing team is in charge of marketing the product, studying competition, creating content and innovating amongst others. Design and Engineering team is comprised of User Interface (UX) designers and software engineers responsible for a seamless user experience on the website, launching of tools and making sure that the whole product is functioning as expected as far as the technical bit is concerned.

1.2 Internship Position

My role in this internship was a Data Analysis Engineer. I was integrated into the Data department where I had one supervisor, Mr. Richard Spiegel, the head of research. The Data team had 5 members including myself. The company takes a flexible approach to the tools they use, as long as it solves the particular task at hand efficiently and effectively. Despite this, they have a default technological stack of tools that holds everything together. I discuss this in the next subsection.

Chapter 2

Development Environment

This chapter details the development environment that I worked with during the period of my internship.

Flexibility in the tools to be used in a company depends on their availability and cost. To minimize cost and save time, the company relies heavily on open source software. Open source software is software with source code that can be modified, enhanced and distributed to anyone and for any purpose [8]. Part of the technology stack included the following:

- Backend - Python 3, PHP
- Databases - MySQL, PostgreSQL
- Frontend - JavaScript, Bootstrap, CSS
- Search - Apache Solr
- Data Visualisation and analytics - Chartio, Tableau, Carto, Google Analytics
- Software development methodology - Scrum, Kanban
- Documentation - Sphinx, Read the Docs, Google Docs

Most of these tools are used by the Design and Engineering department since this department hosts the company's software and web developers. The section below describes the tools that I worked with in detail.

2.1 Python

Python is an open source high level, object-oriented programming language written in the C language [9]. Python has mostly been used for scripting and for rapid web application development and in recent times, data science. The company uses Python for database access, scientific computing, using the Pandas library for data analysis and interacting with the Google Analytics [10], Campaign Monitor [11] and Kissmetrics [12] application programming interfaces (APIs)

2.2 MySQL

MySQL is a fully-featured open-source relational database management system (RDBMS) that is developed, distributed and supported by Oracle Corporation [13]. Like other relational systems, MySQL stores data in relations and uses the structured query language (SQL) for database access. The company uses MySQL as its primary data store.

2.3 PostgreSQL

PostgreSQL is an open source, object-relational database management system (ORDBMS). It is developed by the PostgreSQL Global Development Group. It is free and licensed under the PostgreSQL License. Similar to other relational systems, it uses the structured query language (SQL) for making queries [14]. PostgreSQL acts as a secondary storage for the *properties* and tables in the company's main database.

2.4 Apache Solr

Solr is a highly reliable, scalable fault-tolerant search engine providing distributed indexing, replication and load-balanced querying, automated failover, and recovery [15]. Solr is used to provide search optimization in high traffic websites with large text-centric data [16]. It allows for near real-time indexing features and allows for full-text search capabilities [15]. At the company Solr is used for common search engine use cases like basic keyword search and for ranked retrieval of documents.

2.5 Chartio

Chartio [17] is a cloud-based analytics software solution. It comes with many features that allow for great data analysis through the creation of charts and dashboards from data. Due to its built-in support for SQL database systems, the company integrated it with its databases and enabled us to build performance reports, present web analytics and build data pipelines.

2.6 Sphinx

Sphinx makes it easy to generate documentation for your projects [18]. It allows for the documentation of software projects in different languages. The company uses Sphinx to document information about data and database access for each department, internal projects, programming best practices, and all solved tasks.

2.7 Scrum

Scrum is an agile software development methodology. A scrum setting enables a team to develop new features or a new capability of a software within 2 to 4 weeks. We used scrum

to build some of the projects and also to ship some updates on the real estate platform in weekly sprints held every week.

Chapter 3

Plan and Mission

This section details the work plan and mission of the internship.

3.1 Work Plan

The work plan for the internship acted as a blueprint of what was to be learned at the end of it. The work plan was defined as follows:

- Analysis of the company data - this was fundamental at it involved analysing and understanding the data in use, how it is collected, the data types, it's structure and how it is stored.
- Writing database queries on installations of MySQL, PostgreSQL, and Google Bigquery - running cron jobs, writing of relevant queries for important data output.
- Data wrangling, scraping, machine learning - collecting, cleaning, parsing and training of data.
- Producing visual data reports in Chartio, Carto, Tableau, Mapbox, and other open source data visualization tools.
- Writing Documentation - providing and updating documentation for every completed task for interdepartmental use.

3.2 Monitoring Plan

- Familiarisation with the data, understanding the internals of the Data department and the installation of the necessary tools.
- Execution of available tasks given by the team
- Departmental meetings were held every Thursday where sprints were conducted and where an individual performance of tasks was discussed.

3.3 Objectives

The following were the objectives of this internship:

- Enable the trainee to understand data engineering and analysis fundamentals.
- Enable the trainee to be able to perform data analysis to generate business insights.
- Enable the trainee to acquire big data and database terms, optimization and best practices for data analysis.
- Expose the trainee to exploring data with concepts like machine learning and data mining.
- Expose the trainee to agile software development environments.

3.4 Tasks

This section lists the main tasks I was given. Due to the autonomy of the Data and Engineering Departments, it was really flexible to work on a variety of projects as they were open to all. Preference to a project, in this case, meant one that a person felt comfortable in. Nevertheless, all tasks were supervised. These included:

- Producing performance reports based on MySQL and PostgreSQL queries on Chartio - The company has huge data sets and as such it was necessary to understand the performance in order to avoid bottlenecks. These reports would go a long way in helping us understand what and how to query and also making necessary changes to the schema in the database.
- Predicting house rental prices and house preferences using machine learning algorithms - This task entailed using machine learning algorithms, in this case, the random forest to predict house rental prices. The task also involved, among others, cross-checking with other real estate websites and also data from property letting websites like Airbnb.
- Analysing if the language is an important factor when buyers are looking for property on the website - The property platform was served in eight languages and more often than not, the number of enquiries on a property was dictated by the language in which the user accessed the site. This task involved analysing all the data from the *enquiries* table in the database and aggregating where the users came from and from where they sent the request. This also involved getting data from Google Analytics

- Analysing buyer trends during the different seasons of the year using all the buyer data - This task involved gathering all the data collected on the platform from 2004 until 2016 and analysing the buyer trends for every season. Having this information would be fundamental as it would lead to the development and execution of different marketing strategies for the 2017/2018 financial year.
- Analysing user data from Campaign Monitor, Google Analytics and Kissmetrics application programming interfaces (APIs). An Application Programming Interface (API) is a set of clearly defined routines and methods that allows communication between software components [19]. This task was involved with fetching data from these tools to try and understand user behaviour, time spent on the website, browser activity, website response times, user enquiries and newsletter performance.
- Identifying and reporting on the relationship between house prices and the location of a property in relation to distance from the sea - This task involved analysing if the location of a house affects its price. Of primary focus was a house's distance from the sea.
- Documenting new and updating existing documentation - Every task or project had to be documented for use by other departments and for reference purposes.

3.5 Detailed Internship Progress

This section breaks down the tasks into their different subtasks.

June 2017 (27th - 30th)
This was the initial week of my internship and mainly involved understanding the work culture and how things run at the company. This mostly involved:

- Induction - introduction to the company, the different departments and what they do and all staff members.
- Induction II - Familiarisation with the company's working environment, the technological stack in use, the data and other tools.
- Setting up of my environment - installation of necessary applications and workstations.

July 2017 (3rd - 28th)
Task: Analyse buyer trends during the different seasons of the year (2004 - 2016) using all buyer data & Google Analytics. Publish reports in Chartio, discuss and document

observations. This task involved looking at the company's data from 2004 in order to analyse buyer trends. This task mainly involved the following activities:

- Write SQL queries to get data on properties bought using each season (Winter, Autumn, Summer, Spring).
- Document and report on the different patterns for each year and each season in regard to page visits and properties bought.
- Integrate MySQL with Google Analytics to get season data through the Chartio aggregate pipelines.
- Analyse the different trends for buyers with regard to the different seasons.

August 2017 (1st - 29th)
Task: Produce performance-based reports based on MySQL and PostgreSQL queries on Chartio.

- Writing optimized queries using the *archived_properties*, *properties*, *user* and *enquiries* tables using *enquiry_id* as an index.
- Comparing the performance of the queries using secondary indexes and increased memory cache.
- Building Chartio aggregation pipelines to integrate the database tables with Google Analytics to obtain page visitor data
- Publishing the performance reports on Chartio to be used by the engineering department.

September 2017 (5th - 28th)
Task: Use machine learning algorithms to predict house prices and preferences using Scikit-Learn and Python 3

- Exploring the data and identifying the features and the target variables
- Calculating statistics on the main features to be used. Removing outliers.
- Utilising Scikit-Learn[20] to shuffle and split data.
- Identifying common factors affecting the house price
- Comparing models used in other companies using big data in real estate commerce on a large scale like Airbnb.

October 2017 (3rd - 27th)
Task: Identify and report on the relationship between house prices and the location of a property in relation to distance from the sea.

- Identify how proximity to the sea affects house prices.
- Identify properties within a considerable distance from the sea but with equal or similar prices to those in the sea.
- Obtain and compare data results with the “Instituto Nacional Estadistica” of Spain.
- Identify models used by Airbnb and other rental or real estate or housing portals.

November 2017 (3rd - 24th)
Task: Analyse if the language is an important factor when buyers are looking for property on the website.

- Writing SQL queries gathering all enquiries from *Enquiries* table in MySQL and PostgreSQL databases noting the different languages used in each enquiry.
- Aggregating enquiry origin and enquiry destination.
- Identifying enquiry patterns in regard to the language in which the page was served in.
- Comparing the results on a Sankey diagram.

December 2017 (4th - 22nd)
Task: Analyse user data from Campaign Monitor, Google Analytics and Kissmetrics APIs. This task involved the following activities:

- Using Python 3, to collect data from the Kissmetrics and Campaign Monitor APIs.
- Under the Campaign Monitor API, reporting on Campaigns, Lists, and Subscribers.
- Under the Kissmetrics API, reporting on Events, Reports, and Properties.
- Under Google Analytics API, using the Reporting API provided to identify user behaviour on the different pages of the website, browser loading times and time spent on the website.
- Documenting average duration spent on the different pages.

Chapter 4

Internship Assessment

Most of the tasks that were done by the data department mostly helped the other departments like marketing and engineering departments to make informed decisions. They were also used for the internal operations of the Data department. The routine Thursday meetings were used to review and evaluate tasks done by each team member.

4.1 Technical Skills

During my 6 months at the Internship I was able to acquire the following technical skills:

- Data wrangling and scraping using Python and the R programming language - I learned how to pull and parse data from the web, documents, and Extensible Markup Language (XML) markups.
- Database query optimization and performance analysis - I learned the best practices of writing database queries especially involving huge datasets and when faced with infrastructural challenges. Moreover, I was able to learn the role of database engines both in the storage and querying of data.
- Machine learning algorithms - this was beneficial as it added to the knowledge I had already acquired in my lessons by helping me apply what I learned to solve real-world problems.
- Programming with Representational State Transfer(REST) APIs with Python - APIs are very vital in the development of web services. I was able to obtain data from Campaign Monitor, Kissmetrics, Google Analytics, tools heavily used by the company to obtain the necessary user information to be used in further analysis.
- Writing documentation - documentation is a key principle an in any software process. I learned how to generate data from documentation tools like Sphinx and also how to document the building of a project from start to finish

- Agile development practices - I learned how to work with agile methodologies in software development, more specifically, scrum.
- Data visualization using Chartio, Tableau, Plotly, Carto, and Mapbox - Visualisation is key when explaining statistics and facts to non-tech consumers of data. I was able to visualize complex data in an easy, intuitive way using these tools.

4.2 Professional Skills

In addition to the technical skills, I was able to develop in my personal and professional capacity. Some of these skills include:

- Problem-solving - I was able to own my role which helped me to resolve the challenges that occurred when executing a project. Moreover, I am now able to provide effective solutions to problems, analyse the root cause of a problem and also assess performance metrics.
- Integrity - working at Portal 47 enabled me to acknowledge my strengths and weaknesses. I have also learned how to take charge of my own mistakes and adhere to standards set.
- Communication skills - it goes without saying that communication skills are very vital in the workforce. I have grown tremendously by being able to communicate new ideas and suggestions in a clear and confident way.
- Agility - I have learned how to adapt to different circumstances and different environments. I was also able to react and adapt quickly to different situations and to be flexible when working with new tools.
- Eagerness to learn new skills - I took it upon myself to learn something different from each department. This involved learning a new tool and strategies.

4.3 Conclusion

Working at Portal 47 Ltd gave me a first-hand opportunity of practicing what I learned in theory in the class by enabling me to tackle real-world problems. This opportunity has definitely had a strong impact on my growth as an informatics engineer. I can confidently say that I have grown in regards to my technical, professional and personal skills. Moreover, this internship enabled me to form the basis of my research work which I discuss in the next sections.

Part 2 - Research Work

Chapter 5

Methodology

According to Nicholas in [21], research methods are techniques used for doing research. He goes on to state that these tools represent the tools of the trade, and provide you with ways to collect, sort and analyse information so that you can come to some conclusions. Thus, depending on the research or study to be conducted, the choice of the method to be used depends on the researcher.

This chapter will introduce and explain the motivation for the research method chosen for this research work. The chapter also details on the data collection, the cases to be analysed, an analysis on how the data was processed and finally concludes with a summary.

5.1 Method

The aim of this research is to benchmark and compare the performance of relational systems and NoSQL systems on unstructured data. Relational systems have been at the helm of database technology. However, with recent changes in database technology, emerging tech difficulties and a lot of unstructured data, these systems have come to hit a snag. Unstructured data requires systems that are flexible and easy to scale. Hence our research work will entail an experimental method for doing the benchmarking.

The method chosen for this study is the case study method. According to [22], since there explanation of what a case study is, it is not easy to describe it. A case study can be described as a thorough, methodical investigation of a single individual, group or some other unit in which the researcher examines in-depth data relating to several variables [23].

Research in a case study calls for choosing a few examples of a theme or subject matter to be studied and then exhaustively investigating the attributes or features of those examples. Closely studying these examples or cases, and then comparing and contrasting them enables a researcher to learn about notable features of the subject matter and how it deviates under different situations. According to Yin in [24], a case study research is especially well suited to investigating processes.

A case study approach was chosen because it provides a unique method for investigating a single case which can then be replicated for other similar cases. According to [25], a case study has its strength by obtaining relevant data argues that intensive study methods have their strength in obtaining detailed and relevant data. This implies that this data or information cannot be taken out of context, thus increasing the value of the study.

Another advantage of this method is that it allows for the discovery of more information not planned prior to the start of the study. Due to this, case study becomes a suitable method for creating hypotheses [25] which go a long way in helping to define future research. This also implies that case studies play a role in advancing the knowledge base of a particular field [26].

5.1 Data Collection

According to [76], case study research is not limited to a single source of data as in the use of questionnaires or interviews for carrying out a survey. The following is a list of sources you can use, according to [24][76]:

- Interviews - this involves having open conversations with the main participants or survey-type questions.
- Direct observations - this involves observing the subject in a natural setting.
- Archival records - this could be any records for example census record and survey records.
- Artifacts - tools or objects often observed during a direct observation of the object
- Participant observation - this involves the researcher serving as a participant in an event and observing the actions
- These sources may be used in any combination as well as related sources depending on what is available and relevant for your study.

Data from this research work came from my internship company and was the primary source. This dataset was a replica of the main database used at the company, which involves three tables with over 4 million records. This is a higher number of rows compared to a previous study in [27] that compared the two systems with 25,000 rows and concluded that MongoDB performed better.

5.2 Cases

Since most of the features in database benchmarking have already been used in previous research studies, 4 core features least investigated when comparing the two database

systems were used in this study. Within the four cases selected, the goal was to understand the performance of each system and how it differs with a change in different parameters. The four cases revolved around Indexes, JOINS and aggregation functions, third-party driver integration and scalability of a system.

5.3 Data Analysis

In [24], Yin encourages researchers to produce a high quality analysis of their work. To this end, he came up with three principles. These were:

- Display that the analysis relied on all the data collected
- Mention the main significance of the study and last but not least,
- The researcher should use his or her prior experience or knowledge in the field to further his or her analysis.

In addition to the above, data in this study was analysed using the observe, think, test and revise (OTTR) method as described by [75]. According to [75], analysis should be a frequent process whereby the initial observations made are reflected upon and shape future data collection. Following this method, data was observed and preliminary hypotheses were formulated. After that, consideration was made as to whether additional data was needed, in order to confirm or refute the initial hypotheses. More tests under each case were then conducted to gather more information. Finally, a review of all the observations and results obtained was done.

Summary

This section detailed the research methodology used for this study, the justifications for the method, and the processes taken to compare the two database systems. The next sections discuss relational and NoSQL systems, their models and a comparison of the terminologies used in each system, break down in detail the performance metrics that were used and the case study itself which then culminate to the main comparison between the two systems.

Chapter 6

Related Work

Traditional relational systems have been at the helm of database technology since their inception. However, with growing demands of data, these systems have experienced numerous challenges. New systems have been developed to counter some of these challenges and offer better solutions at a lower cost than the latter.

This chapter will discuss both relational and non-relational or NoSQL systems, their data models and features. To conclude, a comparison between MySQL and MongoDB terminology is provided.

This chapter is organized into 5 sections. The first section describes databases systems and defines relational and NoSQL systems. The second section discusses about the data model used by relational systems while the third section details about the data model used by NoSQL systems. Section four gives an introduction of the Consistency, Availability, Partition Tolerance (CAP) theorem while section 5 compares and gives examples of the terminology and functions used by both systems.

6.1 Database Systems

A database can be defined as a collection of any data that is organized for easy access and management. This data can be stored in tables, columns, rows or documents and can be indexed for faster access [32].
A database management system (DBMS) is a set of programs which provide the user with tools to add, delete, access and analyse the data stored in one location. Data can be accessed using queries, reporting tools or using application programs written specifically for the purpose of accessing the information. The DBMS also provides mechanisms for data integrity, management of security and for user's access to information.

The following subsections discuss relational and NoSQL systems giving examples of each database system.

6.1.1 Relational Databases

A relational database is a collection of structured data stored in tables consisting of rows and columns. The relational model was proposed by E.F. Codd in 1970 [33]. In the relational model, data has to fit a specified schema, considering the various data types before it is

stored. A schema can be defined as the structure of a database that defines the objects in the database. Figure 6a shows a generic representation of a relational model.

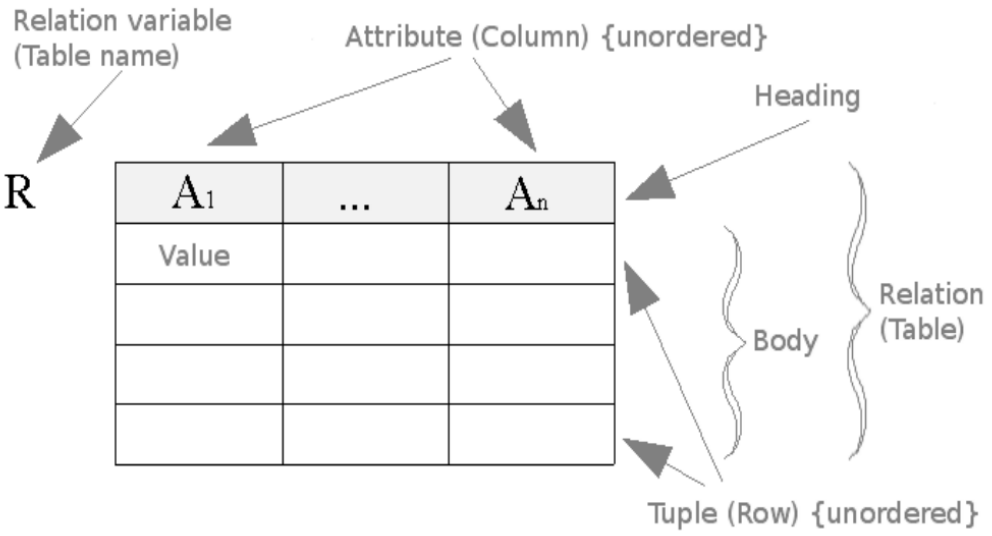

Fig. 6a: Representation of a relational database model [34].

Rows in tables can be related or can form relationships. Primary keys can be used to identify unique rows within a table. Likewise, rows spanning multiple tables can form relationships by using foreign keys. This structural feature makes relational databases vertical scalable, which implies that best performance takes place in a single machine. Vertical scalability means expanding the resources of that single machine to address performance or storage constraints.

These databases mostly use the structured query language (SQL) [35], which is a standard language for managing and accessing databases. SQL consists of many types of statements [30] that provides the scope for data querying, data manipulation (insert, update and delete), data definition (schema creation and modification), and data access control.

Relational databases have been dominant since their inception and provide high-end features that support a majority of today's business systems. Examples of relational databases are MySQL, PostgreSQL & Microsoft SQL SERVER.

6.1.2 Non-Relational Databases

As the name suggests, non-relational databases don’t follow the relational model to store their data. This means that data doesn’t need to fit a specific pattern for it to be stored. These databases are distributed, open-source and offer horizontal scalability which is a problem for relational databases [36]. They are sometimes called “NoSQL (Not only SQL)” as they don’t use SQL as the main query language, or to emphasize that they may support similar languages to SQL [37].

The term “NoSQL” surfaced around 1998 when Carlos Strozzi used it to name his lightweight database management system, which although did not offer the SQL interface, was still relational [38]. This was however distinct from the other databases developed when the term was reintroduced around the year 2009 by Johan Oskarsson at an event discussing “open source, distributed, non-relational databases” [39].

NoSQL databases were built in response to the demands presented by modern applications. They are increasingly being used in big data and real-time web applications [40]. Simplicity in horizontal scaling [41], lucidity of design and first-rate control over availability are some of the motivation for this approach. Some operations are faster in these systems due to the fact that they use different data structures compared to those used in relational systems.

6.1.2.1 Types of NoSQL Systems

There have been different ways to classify the different NoSQL systems, each with different categories and subcategories. An extensive list of the different NoSQL stores can be found in [42].
The following is a presentation of the different NoSQL systems.

6.1.2.1.1 Key-Value Stores

Key value stores use Data stored in databases with this model is stored and looked up using keys [43] that uniquely identifies a record. Every item in the database is stored as a key with its corresponding value (figure 6b). However, the value is entirely hidden to the system as the data can only be queried by the key. Riak [44] and Redis [45] are some examples of Key-Value stores.

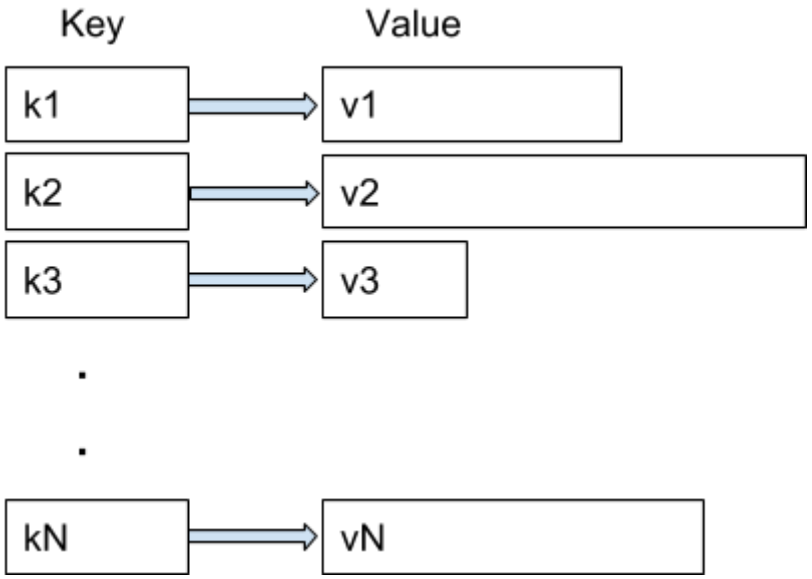

Fig. 6b: Implementation of a key-value store.

6.1.2.1.2 Document Data Stores

Document databases store data in documents (figure 6c). These documents normally are objects in JavaScript Object Notation (JSON) or Extensible Markup Language (XML) format. Each document is inherently an object, which contains one or more fields. They have dynamic schemas meaning each document can contain different fields. Documents can be grouped to form collections which are similar to a table in a relational database. This is particularly helpful when it comes to modelling unstructured data. The most common examples of document data stores are MongoDB [46] and CouchDB [47].

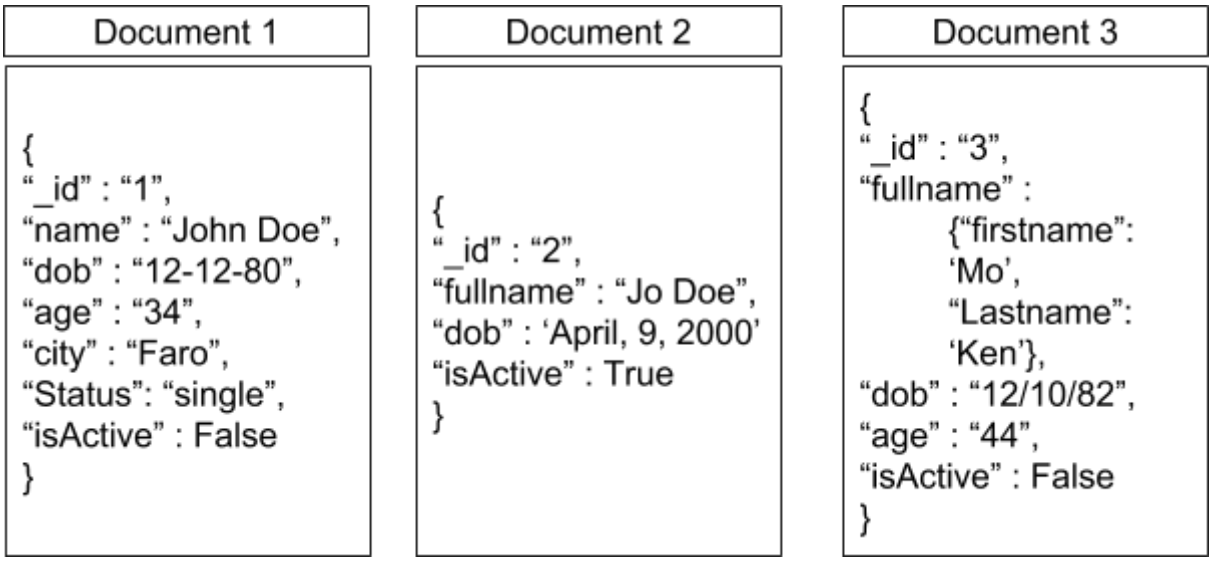

Fig. 6c: Implementation of a document data store.

6.1.2.1.3 Wide Column Data Stores

Also known as column family stores, wide column data stores use a sparse, distributed multidimensional sorted map to store data. Each record can vary in the number of columns that are stored and data is retrieved by primary key per column family. Known examples of wide column data stores are Cassandra [48] and HBase [49].

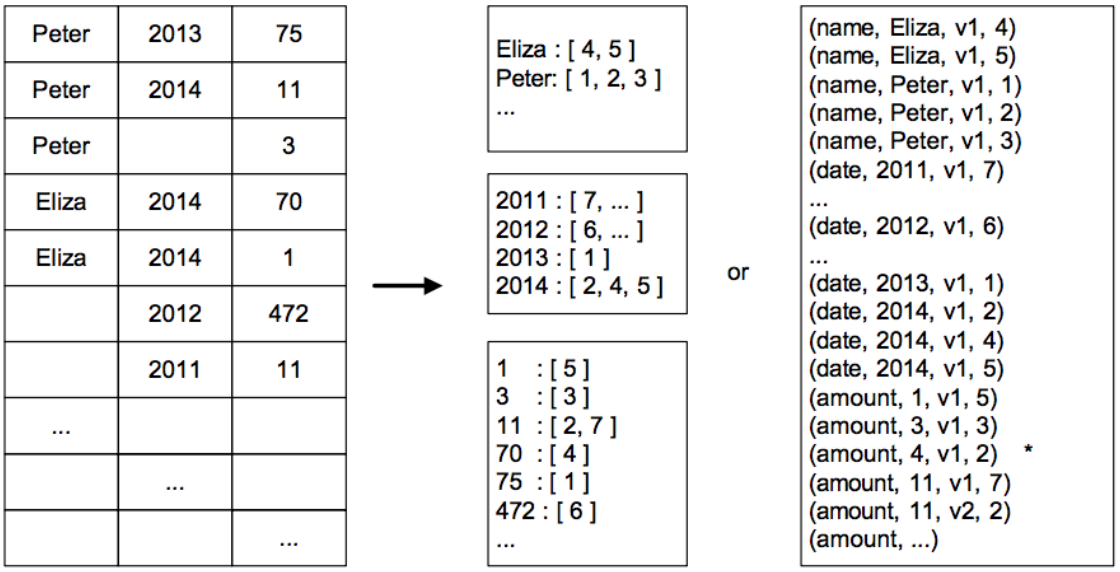

Fig. 6d: Wide column database implementation.

6.1.2.1.4 Graph Data Stores

Graphs are a fundamental aspect of computer science. A graph is a set of items connected by edges [50]. Thus, graph databases normally use graph structures with nodes, edges, and properties to represent data. In this case, data is modelled in the form of a network of relationships between specific elements. Common examples of this data store are Neo4j [51] and Giraph [52].

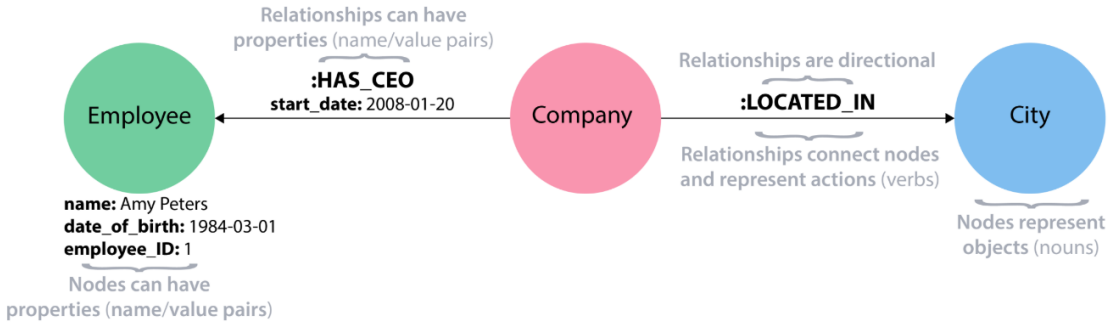

Fig. 6e: Illustration of a graph database implementation [53].

6.2 ACID Model

Atomicity, Consistency, Isolation, Durability (ACID) are properties of database transactions that guarantee validity in the event of failure. The acronym ACID was coined by Theo Harder and Andreas Reuter in 1983 [54], building on earlier work [55] by Jim Gray. They are a key feature of relational databases. A database transaction is a sequence of operations

performed in a single unit of work [56] For a unit of work to be considered a transaction, it must satisfy these ACID properties. These properties are defined as follows:

- Atomicity - states that a transaction should be atomic meaning that either all of its modifications are performed or none is performed.
- Consistency - once a transaction is complete, data must be left in a consistent state. To maintain data integrity, all rules must be applied to the transaction's modifications.
- Isolation - a transaction ensures that concurrent executions are isolated and would result in a system state that would be obtained as though the executions were executed sequentially.
- Durability - after a transaction, any modifications of data should be persisted even in the event of a failure. Changes made are permanently stored in the system.

ACID properties have provided transaction processing with a solid and stable foundation from which to build for decades. Nevertheless, due to the dawn of the internet, the growth of distributed data stores, the unprecedented increase in data variability and volume, the need to document and store unstructured data, and the subsequent need for more flexibility in terms of scaling, cost, processing, disaster recovery and design, this building foundation has shifted [57]. ACID properties are still essential in transaction processing, but high throughput web applications, non-relational data stores, big data, and data centre distribution called for the invention of new alternatives.

6.3 BASE Model

One factor that enables good performance and horizontal scalability in NoSQL databases is the fact that they have sacrificed the use of ACID. NoSQL databases use BASE [58]. BASE stands for “Basically Available, Soft state, Eventual consistency”. Each property is defined below.

- Basically Available - this principle aims to focus on the availability of data even in the presence of multiple failures. This is mostly achieved using sharding and replication. This means that non-relational systems will spread the data across many storage systems instead of maintaining a single machine. This ensures that in the unlikely event of a failure which would disrupt access to a segment of the data, it will not necessarily result in a complete database outage.
- Soft state - this principle basically means that the state of a system could change over time. This also applies in times where there has been no input since as per the principle of “eventual consistency”, the system will still be receiving updates. As such, the state of the system will always be “soft” [59].

- Eventual Consistency - this principles states that the system will ensure that the data will eventually assume a consistent state. This is in contrary to relational systems where the consistency requirement of ACID will prevent a transaction from executing until the prior transaction is complete and the database is in a consistent state.

6.4 CAP Theorem

As we have seen in the sections above, both relational and NoSQL systems offer a different type of consistency. Relational databases provide strong consistency. Strong consistency, in this case, means ACID-compliant consistency, that is, the properties Atomicity, Consistency, Isolation, and Durability which guarantee that data will be consistent at all times. NoSQL systems, however, offer eventual consistency, which basically means that reads will reflect the latest updates at some point in the future and not at the specific time when they happened. Since some systems e.g., bank applications depend on consistency, eventual consistency becomes a weakness. Thus, when choosing to adopt a NoSQL database, some trade-offs have to be considered.

The trade-offs are explained by the CAP theorem. The CAP theorem, which stands for Consistency, Availability and Partition Tolerance, also known as Brewer's Theorem, was introduced by Eric Brewer on his keynote, at the Symposium on Principles of Distributed Computing in the year 2000 [60]. It is a concept that states that a distributed database system can only have two of the three features at any given time [61][62][63]. A distributed system can only guarantee two of the following:

- Consistency - states that the data should remain in a consistent state after each and every operation.
- Availability - states that the system provides availability with no downtime. Every request sent must also receive a response.
- Partition Tolerance - states that the system should continue to function even in the event of a network partition.

Figure 6f illustrates the CAP theorem.

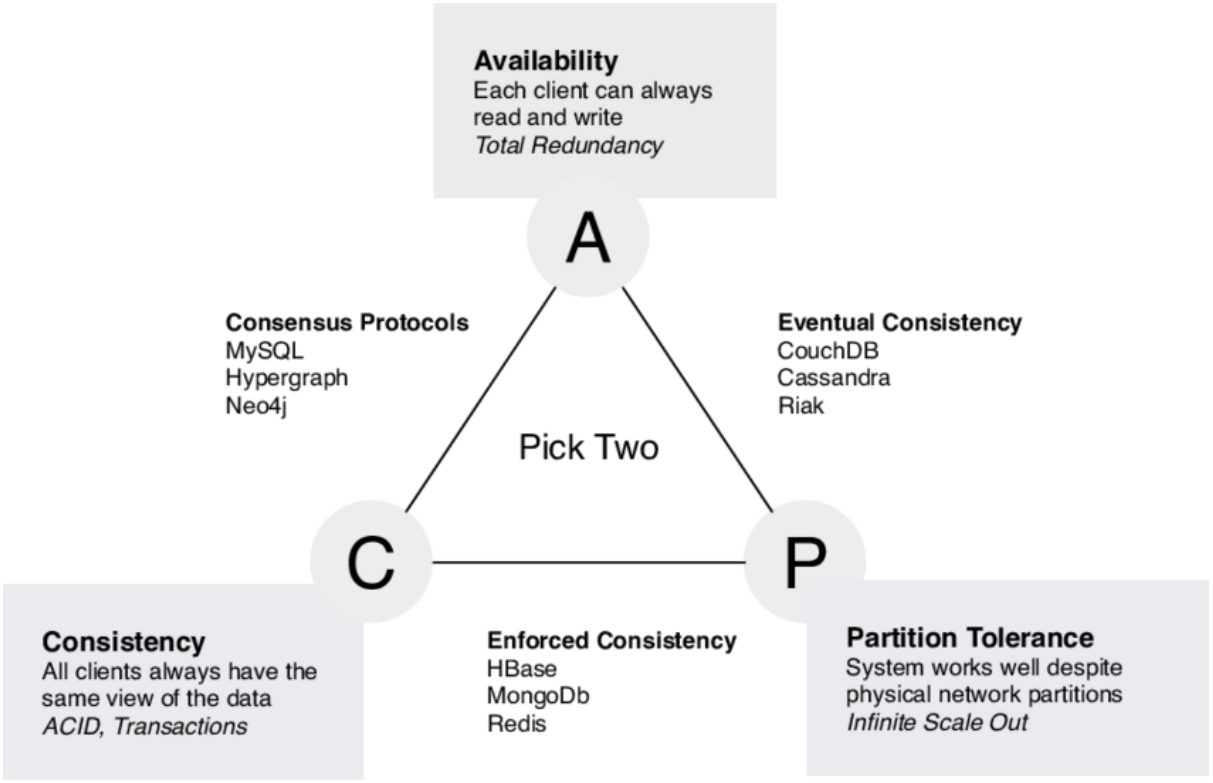

Fig. 6f: The Cap Theorem [64].

According to the theorem, a network prone to partitioning cannot guarantee both consistency and availability. Thus, Partition tolerance becomes a must select since an assumption that a that a network will not fail cannot be made. This means that two options are available to choose from, Availability and Consistency, which brings the possibilities to Consistency and partition tolerance(CP) and availability and partition tolerance(AP).

Table 6a shows some of the differences between the ACID and BASE properties.

ACID (RDBMS)	BASE (NoSQL)
Strong Consistency	Weak Consistency
Isolation	Last write Counts
Transaction	Program managed
Robust database	Simple database
Simple code (SQL)	Complex code
Availability and Consistency	Availability and Partition Tolerance
Scale-up (limited)	Scale-out (unlimited)
Shared resources (Memory / disk / CPU)	Parallelizable

Table 6a. ACID and BASE features

6.5 Comparison of Relational and NoSQL Systems

NoSQL systems have been seen as a better solution for big data applications due to their scalability, cost, and flexibility. They differ from relational systems in four main areas:

- Data model - Relational systems need a defined schema before adding any data to it. NoSQL systems are flexible and don't need a predefined schema which allows for easier updates to data.
- Structure - NoSQL systems are able to handle unstructured data, which currently makes most of the data available as compared to Relational databases which handle structured data due to the fact that they were developed when there was a need to service such data.
- Scaling - Scaling of relational databases is a costly affair as it involves buying bigger servers, increasing processor power or adding more memory. It is much easier and cheaper to scale NoSQL systems as you would only require commodity servers.
- Development model - With the exception of MySQL and PostgreSQL, a majority of the relational systems are closed source which means expanding the infrastructure could turn out to be a costly affair due to licensing fees. On the contrary, NoSQL databases are free and open source with a wide community of users.

For our research, we are going to compare MySQL with MongoDB.

6.5.1 MySQL

MySQL is a relational database which is currently owned by Oracle. MySQL has mostly been used to store data in web applications. It provides support for all major operating systems such as Linux, OS X, and Windows.
MySQL is a fully-featured RDBMS. Some of the features it offers include stored procedures, triggers, views, indexes, cursors, query caching. among other features. MySQL offers support for storage engines. These include InnoDB, MyISAM, CSV, Blackhole, Archive and Merge.

6.5.2 MongoDB

MongoDB is a free, open-source document-oriented database. It is one of the largest used NoSQL database systems. It stores data in flexible JSON (JavaScript Object Notation) like documents, which means fields can vary from document to document and data structure can be changed from time to time. MongoDB supports range queries, field, and regular expression searches [65]. It also provides high availability with replica sets. Being a distributed database at its core, it qualifies to be used as a file system thus providing load balancing [66]. This is made possible through the grid file system [67] function. MongoDB offers support for different programming languages such as Python, Java, C++, C#, and JavaScript.

Table 6b shows a brief comparison of the standard features of the two database systems.

Feature	**MySQL**	**MongoDB**
Written in	C++, C	C++
Type	RDBMS	Document-store
Licence	GPL V2	GNU AGPL V3.0
Schema	Strict	Dynamic
Scaling	Vertical	Horizontal

Table 6b - MySQL, MongoDB feature terminology

6.5.3 Terms and Concepts

This section describes the different terminology used and shows how querying differs in both database systems.

Table 6c presents the MySQL terminology and concepts together with the corresponding MongoDB terminology and concepts on the standard features.

MySQL terms	MongoDB terms
Database	Database
Table	Collection
Index	Index
Row	Document
Column	Field
Join	Embedded Document, $lookup
Primary key set explicitly	Primary key set implicitly
Aggregation (e.g. group by)	Aggregation framework

Table 6c. MySQL, MongoDB terminology

6.5.3.1 Executables

Table 6d shows the SQL executables and the corresponding MongoDB executables.

	MongoDB	MySQL
Database server	mongod	mysqld
Database Client	mongo	mysql

Table 6d: showing MongoDB and MySQL executables.

6.5.3.2 Create and Alter

Table 6e displays functions for creating and altering objects in both database systems. In these examples, we assume we are creating a table named `people` .

MySQL Schema Statements	MongoDB Schema Statements

`CREATE TABLE people (`
`id MEDIUMINT NOT NULL`
`AUTO_INCREMENT,`
`user_id Varchar(30),`
`age Number, status char(1),`
`PRIMARY KEY (id))`
	`db.people.insertOne( {`
	`user_id: "abc123", age: 55,`
	`status: "A"} )`

`ALTER TABLE people` `ADD join_date DATETIME`	`db.people.updateMany({ },` `{ $set: { join_date: new Date() } })`
`CREATE INDEX idx_user_id_asc` `ON people(user_id)`	`db.people.createIndex( { user_id: 1, age: -1 } )`
`DROP TABLE people`	`db.people.drop()`

Table 6e. Create and Alter statements

6.5.3.3 Insert

Table 6f represents SQL statements used when inserting records into a table, and the corresponding MongoDB statements.

MySQL Insert Statement	MongoDB insert schema
`INSERT INTO people(user_id, age,` `status)`	`db.people.insertOne(` `{ user_id: "bcd001", age: 45,`

VALUES ("bcd001", 45, "A")	status: "A" })

Table 6f. Insert Statements

6.5.3.4 Select

Table 6g presents examples of SQL statements related to reading records from tables and the corresponding MongoDB statements.

MySQL Select statements	MongoDB find statements
SELECT * **FROM** people	db.people.find()
SELECT id, user_id, status **FROM** people	db.people.find({ }, { user_id: 1, status: 1 })

Table 6g. Select statements

6.5.3.5 Update

Table 6h presents examples of SQL statements related to updating existing records in tables and the corresponding MongoDB statements.

MySQL update statements	MongoDB update statements
UPDATE people **SET** status = "C" **WHERE** age > 25	db.people.updateMany({ age: { $gt: 25 } }, { $set: { status: "C" } })
UPDATE people **SET** age = age + 3 **WHERE** status = "A"	db.people.updateMany({ status: "A" } , { $inc: { age: 3 } })

Table 6h. Update statements

6.5.3.6 Delete

Table 6i presents examples of SQL statements related to updating existing records in tables and the corresponding MongoDB statements.

MySQL Delete Statements	MongoDB delete statements
DELETE FROM people **WHERE** status = "D"	db.people.deleteMany({ status: "D" })

`DELETE FROM people`	`db.people.deleteMany({})`

Table 6i. Delete statements

6.5.3.7 Aggregation

Table 6j shows an overview of SQL aggregation terms and functions and the corresponding MongoDB aggregation operators.

MySQL aggregation terms & functions	MongoDB aggregation operators
`WHERE`	`$match`
`GROUP BY`	`$group`
`HAVING`	`$match`
`SELECT`	`$project`
`ORDER BY`	`$sort`
`LIMIT`	`$limit`
`SUM`	`$sum`
`COUNT`	`$sum` `$sortByCount`
`JOIN`	`$lookup`

Table 6j. Aggregation Terminology

Table 6k shows an example of a simple aggregation query using the `COUNT()` function in MySQL and the equivalent `$sum` in MongoDB.

SQL Example	MongoDB Example	Description
`SELECT COUNT(*) AS count FROM orders`	`db.orders.aggregate( [ { $group: { _id: null, count: { $sum: 1 } } } ] )`	Count all records from the `orders` table

Table 6k. Aggregation example

Summary

This chapter gave an overview of database technologies. We introduced the concepts of database systems and discussed relational and NoSQL systems. On relational systems, we discussed their table structure and how they store data. We also discussed the ACID

properties, a key feature in these systems. We further discussed NoSQL systems, how they store data and how they disrupted the database ecosystem. We also saw how contrary to relational systems, they substitute consistency by offering eventual consistency using the BASE properties. We detailed, using the CAP theorem, where database systems fall in terms of consistency, availability and partition tolerance.

Last but not least, we introduced the databases to be used for our comparison, MySQL and MongoDB. We went further and compared the terminology used by both systems. Of particular interest were the query statements used on create, read, update and delete operations. From the examples, we can see how the query statements differ. MySQL uses SQL to query database objects while MongoDB uses JavaScript. The next section discusses the performance metrics that were used for this study.

Chapter 7

Performance Evaluation Metrics

This study provides more details about the goal of this research work. It includes a comparative study of relational and NoSQL database systems using the different metrics to be investigated.

7.1 Index Performance

Indexing is a way to optimize database performance by minimizing disk access operations required when a query is processed. A database index is a data structure that speeds up fetching of data from a query. Indexing makes it easier to search through a table or a collection. Indexing increases read performance but decreases write performance.

Both MySQL and MongoDB support indexes through the B-tree data structure. The storage engines in use will be `InnoDB` for MySQL and `wiredTiger` for MongoDB. Both of these databases will automatically create indexes for primary and foreign keys in the database. These indexes will not be sufficient for our benchmarking and thus we would need to create our own.

7.2 JOINS and The Aggregation Framework

A join combines rows from two or more tables. Joins are a key feature in relational databases. It is this feature that makes it difficult for them to scale out. MySQL supports CROSS JOIN, INNER JOIN, and RIGHT JOIN. In MySQL, JOIN, CROSS JOIN, and INNER JOIN are syntactic equivalents, meaning they can replace each other [68]. However, in standard SQL, they are not equivalent. The joins can be described below:

- CROSS JOIN - matches each row from one database table to all rows of another.
- INNER JOIN - used to return rows from both tables that satisfy the given condition.
- RIGHT and LEFT JOIN - a LEFT JOIN returns all the rows from the table on the left even if no matching rows have been found in the table on the right while a RIGHT JOIN returns all the records from the table on the right even if no matching rows have

been found in the table on the left. Where no matches have been found in either table, NULL is returned.

Being a NoSQL system, MongoDB does not support JOINS. As an alternative, MongoDB provides the aggregation framework. The aggregation framework provides aggregation operations that process data records and return computed results. Aggregation operations group values from multiple documents together and can perform a variety of operations on the grouped data to return a single result [69]. MongoDB provides three ways to perform aggregation:

- the aggregation pipeline - A framework for data aggregation modelled on the concept of data processing pipelines. In an aggregation pipeline, documents go through a pipeline with multiple stages that transforms the documents, resulting in aggregated results [69].
- the map-reduce function - a map-reduce is a data processing pattern for condensing large volumes of data into useful aggregated results. Map-reduce operations use custom JavaScript functions to associate, values to a key. If a key has multiple values associated or mapped to it, the operation reduces the values for the key to a single object [70].
- single purpose aggregation methods - MongoDB provides two methods, `db.collection.count()` and `db.collection.distinct()`,which aggregate documents from a single function [69].

The `$lookup` function allows MongoDB to perform a left outer join to a collection to filter in documents from the *joined* collection. Generally, the `$lookup` function adds a new array field whose elements will be the matching documents from the *joined* collection [69].

7.3 Application Integration

Both MySQL and MongoDB provide ways of interacting with the database on the application layer. They both supported a variety of high-level programming languages like PHP, Python, and Ruby. For this metric, we used Python version 3.7.0. Under application integration, we tried to understand the read and write performance of the database system, through the drivers provided by the language.

This metric is vital as it allows us to tell read and write performance of a high throughput application. In light of current software development practices, application integration is crucial in determining how fast a company can make rapid changes to an application, how easy it is to make those changes and best ways of preventing downtime as the query runs. This metric will apply two balanced workloads with a similar number of reads and writes.

7.4 Replication

Database replication can be described as the process of copying and maintaining database objects in multiple databases to maintain consistency. Replication provides redundancy and

also increases data availability. Replication allows for fault tolerance against the loss of a single server since data is in multiple copies in different servers [71]. In some cases, replication can provide increased read capacity [71] as clients can send read operations to different servers. Furthermore, maintaining additional data copies can be used for dedicated purposes such as data recovery, backup, operational reporting or analytics purposes and also to enable distributed processing of data.

In MySQL, replication is asynchronous by default, which means that the slaves (one or more MySQL database servers) don’t have to be connected to the master (one MySQL database server). MySQL also supports semi-synchronous replication which means that a commit done on the master will have to get acknowledgment from a slave before returning to the session that executed the transaction. After receiving the acknowledgment, the master can return to the session and can then proceed to execute other transactions.

MongoDB supports asynchronous replication through replica sets. A replica set basically consists of a primary and secondaries. All write operations are received by the primary. The secondaries copy operations from the primary to maintain identical datasets.

Chapter 8

Case Study

This section describes the case in detail. It details the application of the case study, the environment set up, the metrics and finally concludes with the results.

8.1 Environment Set-up

This section details the environment set up.

8.1.1 Implementation Details

All the tests were done on a local machine. The details are as follows:

- Operating System - Mac OS High Sierra
- Memory - 8GB DDR3
- Hard disk drive - 512GB
- Processor - Intel i5, 2.7 GHz

8.1.2 Database Configurations

The following subsection provides a detailed description of the configuration of each database.

8.1.2.1 MySQL

MySQL version 5.7.12 was used for this benchmarking. MySQL is readily available online on the official MySQL website and can be downloaded for all operating systems. It is necessary which engine is used as far as the performance of the database is concerned. For all the experiments conducted, `InnoDB` storage engine was used. This storage was chosen because of its support for ACID transactions, its stability when performing write-intensive operations, its support for row-level locking and the ability to do full-text search. `Pymysql` version `0.7.2` was used as the main Python driver for the application integration tests.

8.1.2.2 MongoDB

MongoDB 3.6.2 was used for this study. For MongoDB, `wiredTiger` is the default storage engine. A major benefit of this storage engine includes document level concurrency which ensures that multiple clients can write to different documents of a collection at a go. It also supports the compression of collections and indexes which minimizes storage use. Both of these benefits were important for this study. `Pymongo` version 3.7.1 was used as the main Python driver.

8.1.3 Benchmarking Tool

For the benchmarking, we used the Yahoo! Cloud Serving Benchmark (YCSB) [28]. YCSB is an open source benchmarking framework designed by Yahoo to compare performance of database systems. At its inception, it only had support for NoSQL systems but currently supports relational systems too. YCSB provides six inbuilt workloads which provides different scenarios for testing read-write operations and table scans. The workloads used are as follows:

- Workload A - This workload consists of a 50/50 reads and writes mix respectively. Thus, it is an update-heavy workload. The read update ratio is 50/50.
- Workload B - This workload consists of a 95/5 reads and writes mix respectively. Thus, it is a read-mostly workload. The read update ratio is 95/5.
- Workload C - This is a read-only workload where the read-update ratio is 100/0 respectively.
- Workload D - This is a read latest operation. This implies that new records are inserted (5%) and the recently inserted records which are the ones that are read (95%). The read-update-insert ratio is 95/0/5.
- Workload E - This involves querying in short ranges of records, instead of individual records. This workload has a scan-insert ratio of 95/5
- Workload F - This workload is a read-modify-write. What happens in this workload is that a client reads a record, modifies it and writes back the changes.

8.2 Metrics

For the benchmarking process, this research work consisted of four main metrics as defined in chapter 7. The metrics were set up as follows:

- Index performance - this metric investigated the performance of the systems where we had indexes to speed up query lookup time. Two workloads were applied from the benchmarking tool with a scenario of 1 and two threads. These workloads were

mostly read operations. Read-intensive queries were executed to understand which system performs a search faster.

- Aggregation Framework & JOINS - this metric investigated the aggregation functions, and the performance of JOINS compared to the aggregation framework. MongoDB does not support relations as it is not a relational database but nevertheless offers a function that allows for joining collections of different documents, similar in function to JOINS offered in MySQL. Two workloads, focussed on reads, were applied to this metric and four equivalent functions were compared with each other.

- Application Integration - Both of the database systems allow for third-party driver integration. Different drivers for the different languages are all supported. This metric investigated how both MySQL and MongoDB integrate with their respective Python 3 drivers. For MongoDB, the `pymongo` [30] driver was used while `pymysql`[31] was used for MySQL. Two balanced workloads with an equal number of reads and writes were applied to this metric. It was necessary to include writes in this metric to measure how long it takes to insert or update a record.

- Replication - both MySQL and MongoDB support asynchronous replication. A master-slave setup for MySQL was set up as well as the corresponding primary-secondary setup for MongoDB. This metric involved 3 different workloads; a balanced read-write operation, a read-mostly workload and an asynchronous replication throughput test.

8.3 Methodology

This section details the application of the methodology used for this research work.

8.3.1 Data

The data used was data from the company. This dataset consisted of more than 4 million rows. The data was stored in three database tables. The main tables in this dataset were:

- Enquiries - This table had 3.3 million records and contained enquiries sent to property agents via the platform up to December 2017. This table contained five fields
- Properties - this table had 438,000 records and contained property listing on the platform. This table had 9 fields..

- Archived_properties - This table consisted of archived property listings on the platform and contained more 4.7 million records. This table had nine fields.

The *enquiries* table mapped to the properties table via the *property_id* field. A property could have many enquiries an enquiry can have only one property. The company used two property tables for performance reasons. When properties became inactive, they were moved from the *properties* table to the *archived_properties* table.

This data was imported into MongoDB using the `mongoimport` function and populated on to the tables. For MySQL, the data was imported into the tables using the `LOAD DATA INFILE` syntax.

8.3.2 Cases

There were four main cases in this study. These were:

- Index performance - this case investigated how fast the database systems performed operations with different index parameters.
- Aggregation Framework & JOINS - this case investigated how fast each database system performed operations with the different aggregation functions
- Application Integration - this case investigated how each database system performed operations with a third party driver in the application layer.
- Scalability - This case investigated replication in both database systems and how fast each of the system performs under a certain load and different parameters.

8.3.3 Results and Analysis

This section presents the results obtained together with an analysis.

8.3.3.1 Index Performance

For index performance we used Workload C and D. Workload C is focussed on reads while Workload D is a read-update-insert operation workload.

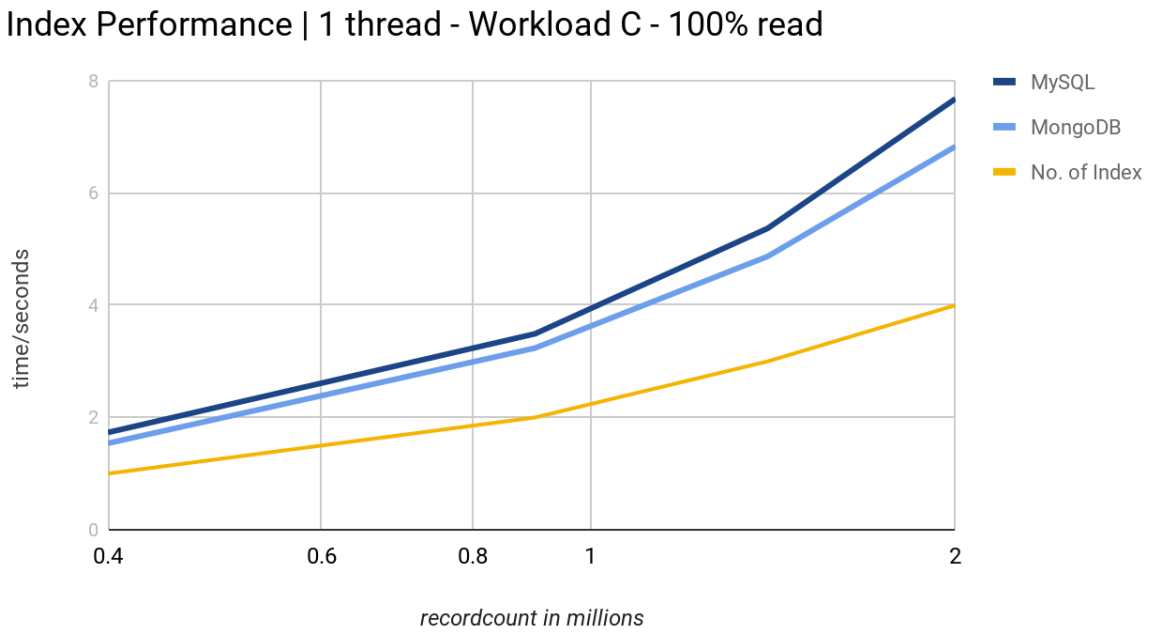

Fig 8.3.3.1(a): MySQL vs. MongoDB, workload C - 100 % read, 1 thread

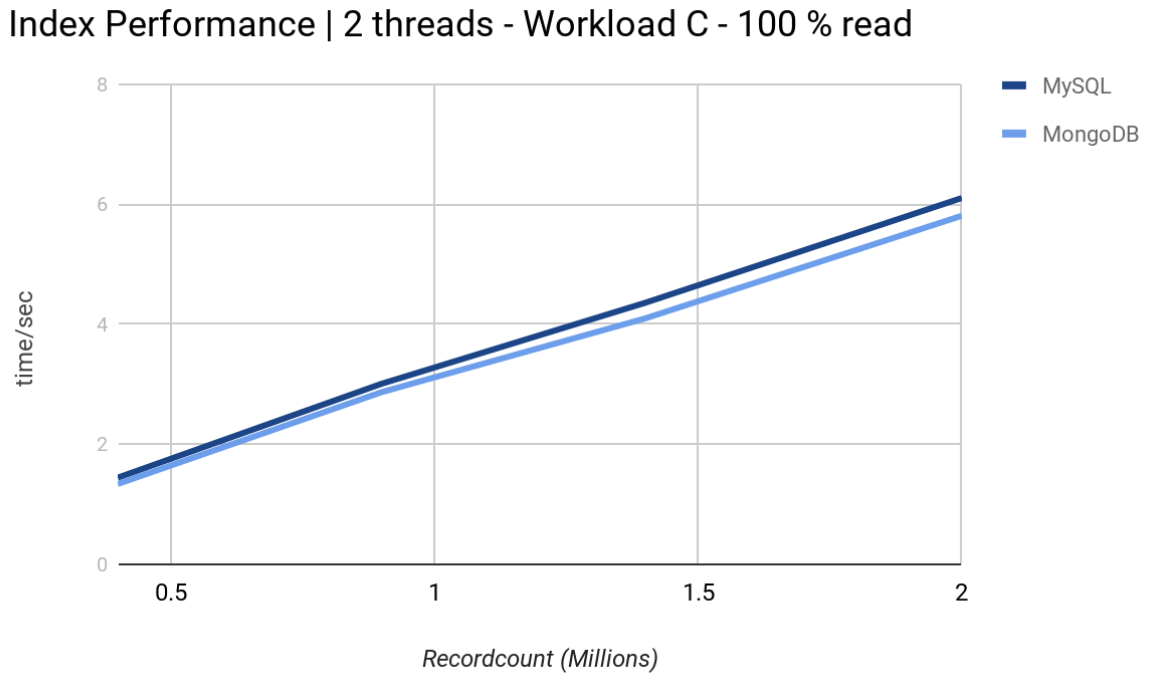

Fig 8.3.3.1(b): MySQL vs. MongoDB, Workload C - 100% reads, 2 threads

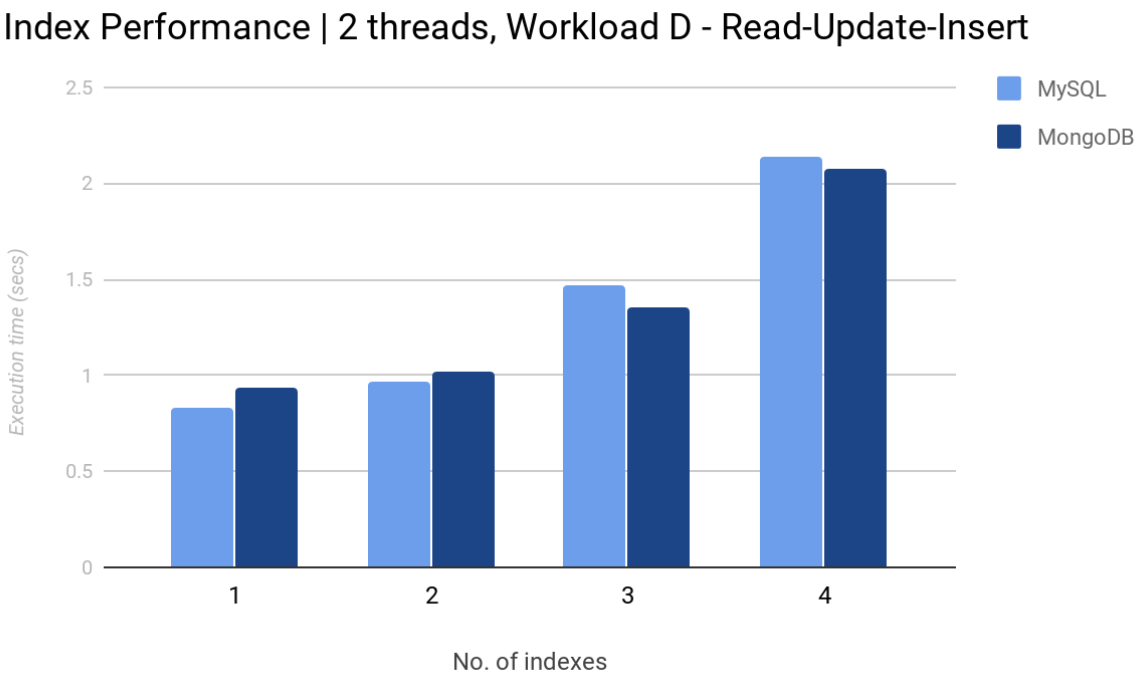

Fig. 8.3.3.1(c): Index Performance, Workload D - read-update-insert, 2 threads

The first test of this metric was focussed on obtaining the fetch times on a workload operation that was read-only. This was done on a single thread with the data being split in different record counts starting with 400,000 for each index. From figure 8.3.3.1(a) we can see that MongoDB performed better than MySQL in all instances. For 0.4 million records, using one index, MongoDB took 1.54 seconds compared to 1.73 for MySQL. Increasing the recordcount to 900,000 and adding another index increased the time to fetch a record by almost 1.5 seconds to 3.23 seconds for MongoDB and 3.49 for MySQL. Important to note is that the time to fetch a record increased as the `recordcount` increased. We also note that MongoDB improved its performance as the `recordcount` increased. MySQL fetched records slower as the `recordcount` increased. We deduce this from the difference in seconds between the record counts. Between the 1.6 and 2 million `recordcount`, MySQL executed the operation in 2.31 seconds compared with 1.96 for MongoDB.

Using the same workload but increasing the number of threads to two and maintaining the same `recordcount`, improves the performance of both databases as shown in figure 8.3.3.1(b). This chart indicates that both systems perform well as the number of threads increase. However, MongoDB still performed better than MySQL.

Figure 8.3.3.1(c) shows the performance of both systems with an increase in the number of indexes. For this test, we used Workload D which is a read-update-insert operation workload. This workload focuses on the latest inserts made on the database. From the chart we observe that for each database, the execution time increased with an increase in the number of indexes. MySQL performed better than MongoDB with 2 indexes but lost to

MongoDB as the indexes increased. The latter was 0.11 seconds faster with one index and 0.05 seconds faster with 2 indexes. Nevertheless, the difference in execution times was low. From this figure, we can conclude that MySQL performs better than MongoDB with a certain number of Indexes and that MongoDB performs better than MySQL as the number of indexes increase.

Nevertheless, in both systems, there was too much overhead as the size of the dataset increased as the number of indexes increased. A higher number of indexes on a table slows down the execution times.

8.3.3.2 Aggregation Framework & JOINS

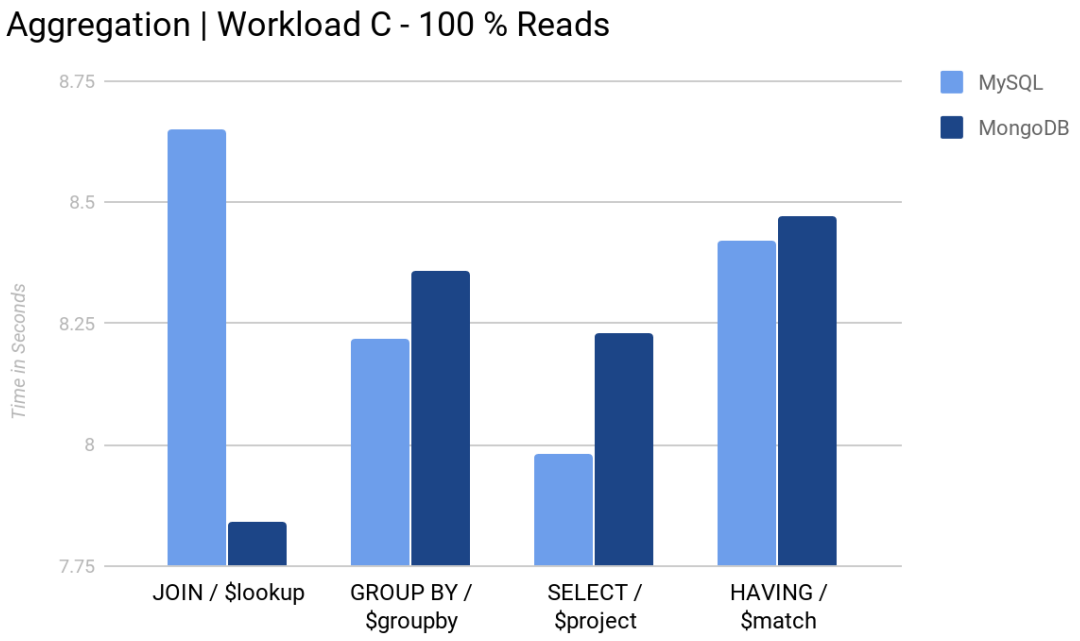

Figure 8.3.3.2(a): Aggregation queries, read-only workload

Figure 8.3.3.2(a) displays the results of the first test of the aggregation metric on both systems. This test was purely a read-only operation. From the results, MySQL outperformed MongoDB in all the aggregation queries except from the `JOIN / $lookup`, where the former took 0.61 seconds longer. MySQL was faster by 0.14, 0.25 and 0.05 seconds on the `GROUP BY / $groupby`, `SELECT / $project`, `HAVING / $match` queries respectively. One reason MongoDB performed faster on the `$lookup` function is because it does not need to do a collection scan due to the `$match` stage of the aggregation pipeline. The `$match` stage filters the results before the `$lookup` stage is actually performed. This also explains why MongoDB performed better on the HAVING / `$match` query where the difference was smallest compared to the rest. The `recordcount` used for this test was 1.5 million records.

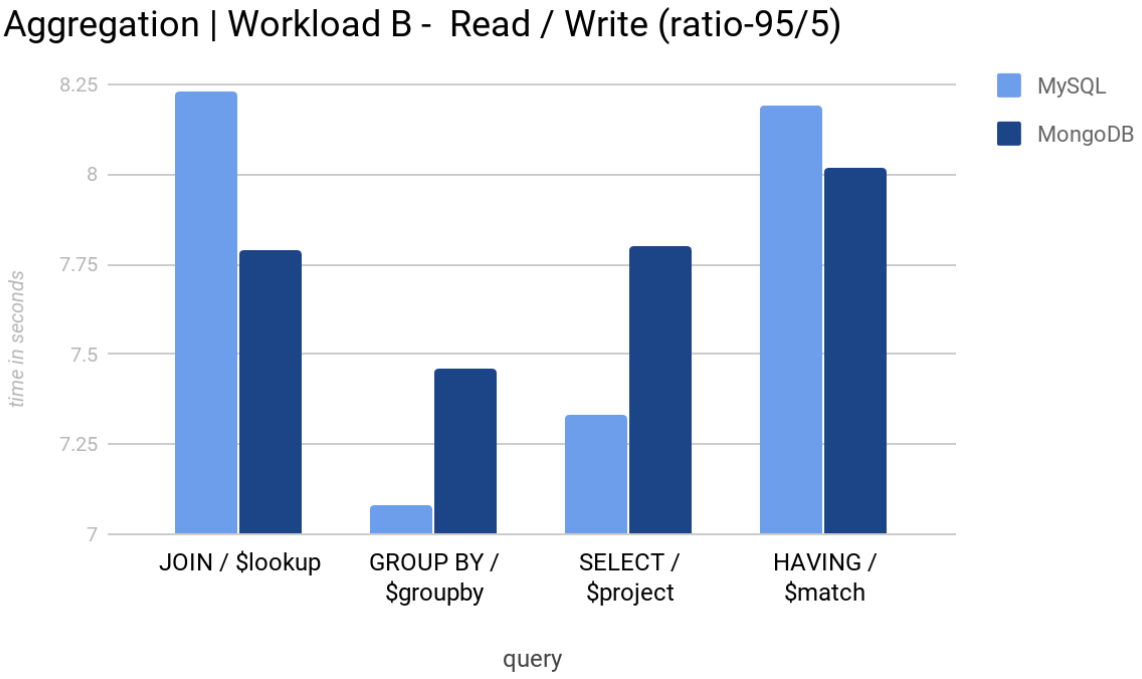

Figure 8.3.3.2(b): Aggregation queries, Workload B

Figure 8.3.3.2(b) illustrates similar results to what was observed in figure 8.3.3.2(a) where MySQL performed better in the GROUP BY / `$groupby` and SELECT / `$project` queries. Comparably, MongoDB was 0.44 seconds faster on the JOIN / `$lookup` and 0.17 seconds faster on the HAVING / `$match` query. Important to note is the variation in the time taken compared to the previous diagram. This is due to the workload used. This test used workload B, which is a read mostly operation which means that it included writes in the ratio of 95:5. Similar to the previous test, the `recordcount` used was 1.5 million records.

8.3.3.3 Application Integration

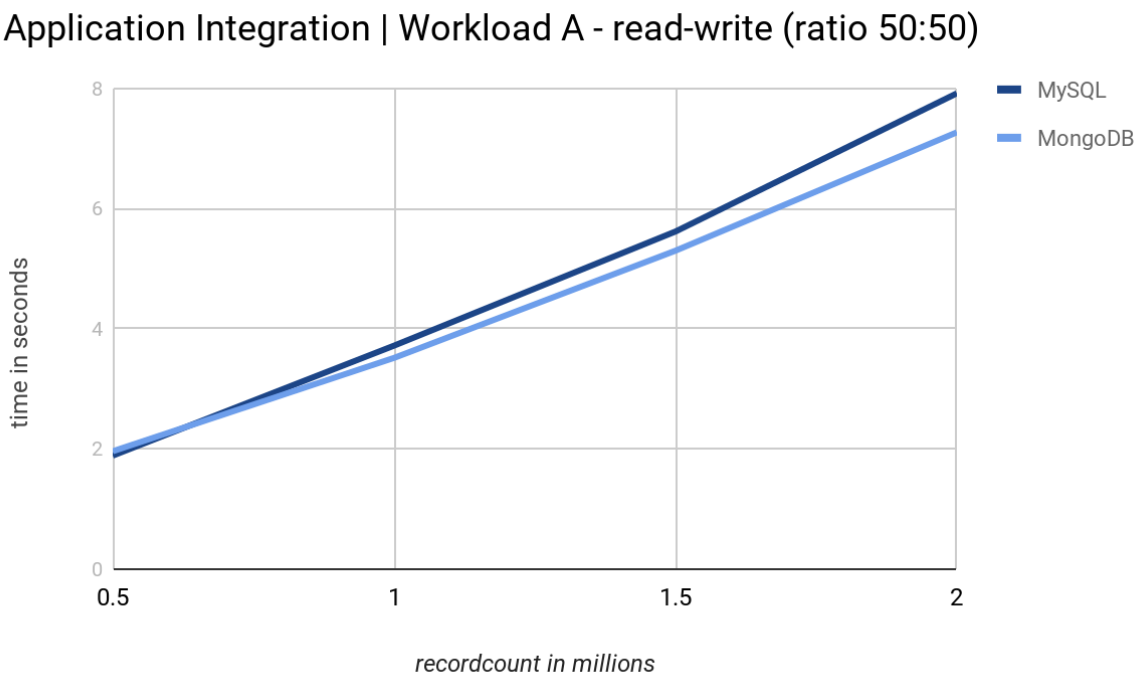

Figure 8.3.3.3(a): Application Integration | Workload A - read-write (50:50)

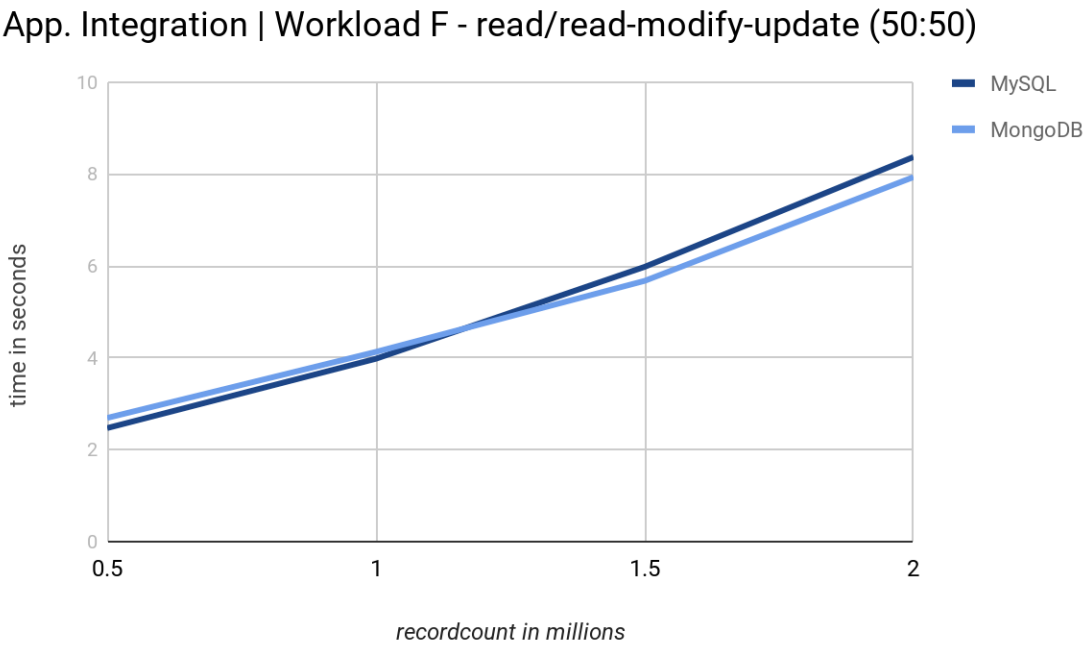

Figure 8.3.3.3(b): Application Integration | workload F - read/read-modify-update (50:50)

Figure 8.3.3.3(a) displays the results of both databases on a workload that has equal read-write ratio. We can observe that MySQL was 0.073 seconds faster on a `recordcount`

of 500000 documents than MongoDB. When compared to figure 8.3.3.1(b) where a read-only workload was applied, MongoDB was faster. This behaviour is due to the fact that MySQL performs better on writes. However, as the `recordcount` increases, MongoDB edges slightly faster than MySQL. This is evident as the recordcount hit the 1 million mark, where MySQL was 0.204 seconds slower. Another interesting fact is the small gap of performance between the two, compared to the workloads that were mostly read-only. For instance, MongoDB was only 0.649 seconds faster than MySQL on a balanced workload, compared to the read only workload in figure 8.3.3.1(b) where it was 0.858 seconds faster.

Figure 8.3.3.3(b) presents an almost similar trend. MySQL was fastest on the first two instances of the recordcount. The workload applied in this load was a read-modify-write operation. MySQL was 0.22 seconds faster with a `recordcount` of 500,000 and 0.149 seconds faster with a `recordcount` of 1,000,000. However, this performance degrades as the `recordcount` increases. Performance in MongoDB continues to improve as the data increases. MongoDB was 0.30 seconds faster at the 1.5 million mark and 0.44 seconds faster at the 2 million mark. It is also important to note the low gap of performance between the two on this particular workload. Although MySQL loses out eventually, it is able to reduce the performance gap due to its ability to process writes faster, thanks to the high-performance capabilities of its `InnoDB` storage engine.

8.3.3.4 Replication

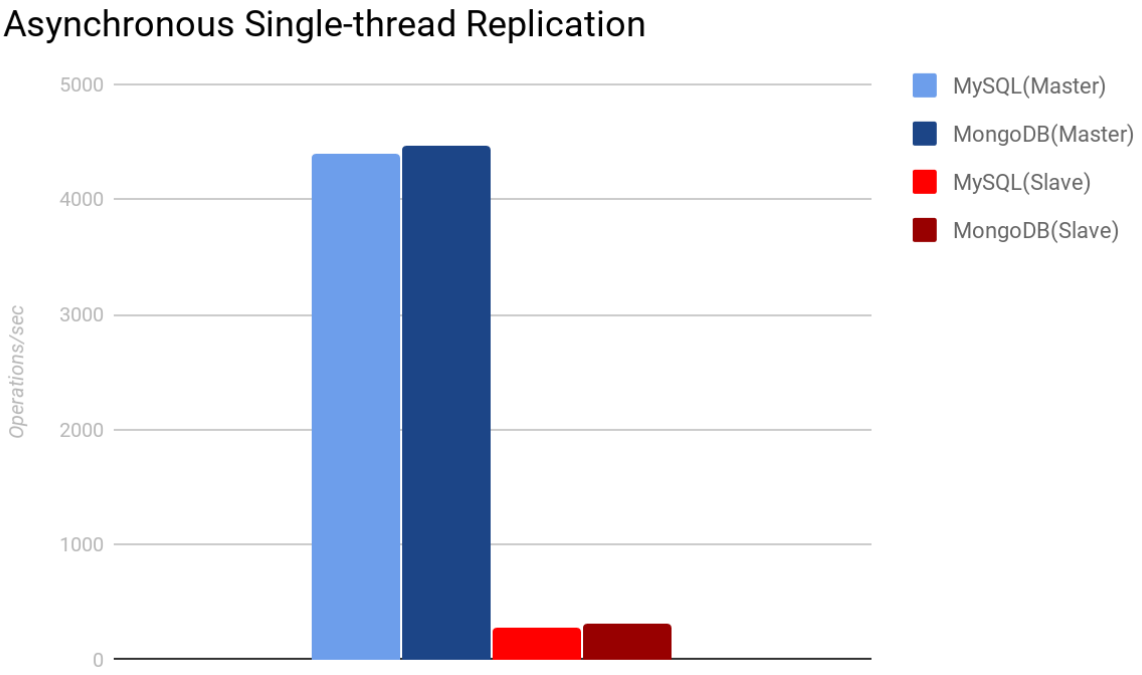

Figure 8.3.3.4(a): Asynchronous single-thread replication

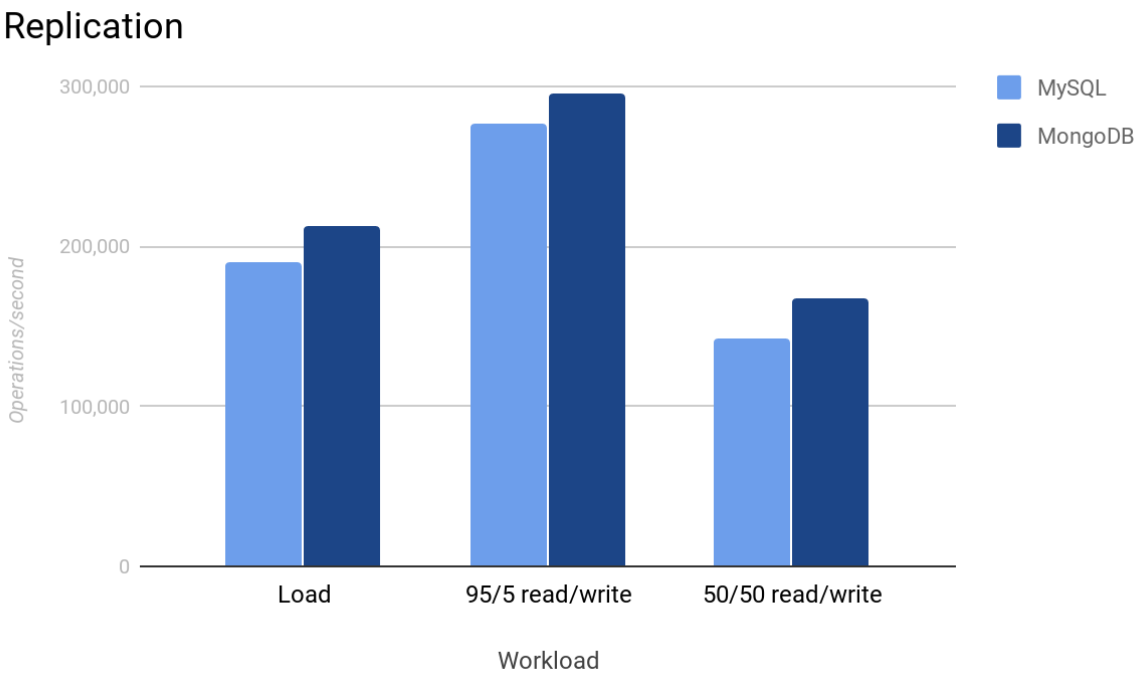

Figure 8.3.3.4(b): Replication with different workloads

Figure 8.3.3.4(a) displays the results of the asynchronous replication test of both systems, each of them running on a single thread. For this test we used Workload C which is a read-only operation. From the figure we can conclude that both of the Masters perform more operations than the slaves. The MySQL master was able to process 4388 operations in a second, while the MongoDB master processed 4473 operations, 85 more than the former. The slaves processed very few operations due to the size of the dataset thus causing a lag. The MongoDB slave was able to process 313 operations while the MySQL slave was able to process 218 operations which was 95 operations less than the former.

Figure 8.3.3.4(b) shows the second test which compares the two systems using workload A that is 95% reads and 5% updates. From the results we can see that MongoDB has a higher throughput than MySQL. It processed 18,757 more operations than MySQL which processed 276,126 operations. Moreover, this represents a slight decrease in improvement than was observed in the load test because the write operations only accounted for 5% of all the operations run. This result indicates that MongoDB, using its `wiredTiger` storage engine provides better concurrency control than `InnoDB` does for MySQL.

On the balanced workload, consisting of an equal number of reads and writes, MongoDB processed 24,985 more operations than MySQL which processed 189,800 operations. This test had the least number of operations processed on both systems due to an increase in the number of writes.

Chapter 9

Conclusion and Future Work

This chapter summarises the research work and details about possible future work.

9.1 Conclusion

This study benchmarked the performance of relational and NoSQL database systems. The goal was to understand how each database system performs using the different features that were tested. Our findings indicate that MongoDB performs better than MySQL on read operations and on a higher load than MySQL. On the other hand, MySQL performs much better on writes, but up to a certain load. As the load increases, write and read performance decreases. MongoDB outperformed MySQL on index performance, application integration and replication. Although MySQL performed better on the aggregation framework, it was only up to a certain recordcount. As the data increased, performance decreased.

From this we can conclude that the choice of a database depends on the needs of an application. It would be important to factor in the advantages and disadvantages of each system and how each system performs best under certain features that would affect performance of an application. From this work we can argue that applications that need high scalability and or have high performance throughput would be deployed on MongoDB while those that would need integrity and strong consistency of data, would best be deployed on MySQL. At any one point, a sacrifice would have to be made as we saw earlier on the CAP theorem. In all, optimization for any system selected would be key to the best performance of each system.

9.2 Future Work

New features are increasingly being added to database systems. As of the conducting of this study, the latest release of MongoDB, version 4.0, supports ACID transactions [66]. This feature would make for a good comparison between the two systems on how each processes the transactions.

For future tests, adding a new database system would help us increase the scope of this study. There has been a new wave of database systems known as NewSQL. According to

[67] and [68], NewSQL database systems are modern relational systems that aim to provide the scalability performance of NoSQL systems while still maintaining ACID transactions. It would be interesting to compare the performance of the three systems with similar or different metrics.

Storage engines play a key role in how data is stored and how queries are executed. A lot of the features that come with storage engines were not used in this study as they were far beyond the scope. As future work, a comparison can be done between the two storage engines, `InnoDB` and `wiredTiger`, investigating how both indexes implement hash indexes, and row level and document level locking for MySQL and MongoDB respectively.

References

[1] Ming-Li Emily, Hui N. Chua (2018). "*SQL & NoSQL Database Comparison: from Performance Perspective in Supporting Semi-Structured Data".* Future of Information and Communications Conference (FICC) .

[2] O. Runu, Lonela Halcu (2013). *"Converting unstructured and semi-structured data into knowledge"*. Roedunet International Conference.

[3] Atzeni, P., Bugiotti, F., Cabibbo, L., and Torlone, R. (2016). *"Data modelling in the NoSQL world"*. Computer Standards and Interfaces.

[4] MongoDB White Paper, June 2016. *"How a Database Can Make Your Organization Faster, Better, Leaner"*.

[5] MongoDB White Paper. *"MongoDB Architecture Guide". https://www.mongodb.com/mongodb-architecture*

[6] Leavitt, Neal (2010). *"Will NoSQL Databases Live Up to Their Promise?"*. IEEE Computer.

[7] Kyero website. *Real estate platform*. https://www.kyero.com/

[8] St. Laurent, Andrew M. (2008). "*Understanding Open Source and Free Software Licensing"*. O'Reilly Media. p. 4.

[9] The Python programming language.
https://www.python.org/

[10] Google Analytics APi.
https://developers.google.com/analytics/

[11] Campaign Monitor APi.
https://www.campaignmonitor.com/api/

[12] Kissmetrics APi
https://developers.kissmetrics.com/reference

[13] *"Sun Microsystems Announces Completion of MySQL Acquisition; Paves Way for Secure, Open Source Platform to, Power the Network Economy"*. Sun Microsystems. 26 February 2008. Retrieved 17 September 2012.

[14] PostgreSQL 9.3.0 Documentation. *"What is PostgreSQL?".* PostgreSQL Global Development Group. Retrieved 2013-09-20.
https://www.postgresql.org/docs/current/static/intro-whatis.html

[15] Apache Solr. http://lucene.apache.org/solr/

[16] Trey Grainger, Timothy Potter. *Solr in Action.* First Edition, 2014.

[17]Chartio Documentation. https://chartio.com/

[18] Sphinx Documentation. http://www.sphinx-doc.org/en/master/
[19] What is an API. https://en.wikipedia.org/wiki/Application_programming_interface
[20] Scikit-Learn. http://scikit-learn.org/stable/
[21] Nicholas W. *“Research Methods: The Basics: 2nd Edition*”. 2010
[22] Gustafsson J. “*Single case studies vs. multiple case studies: a comparative study” (Thesis)*. Halmstad, Sweden: Halmstad University, 2017
[23] Woods NF, Calanzaro M. *“Nursing research: theory and practice”*. St Louis: Mosby, 1980.
[24] Robert Yin, “*Case Study Research: Design and Methods”*, 2nd ed. (Thousand Oaks, Calif.: Sage Publications, 1994).
[25] Jacobsen, D. I. (2002). “*Vad hur och varför: Om metodval i företagsekonomi och andra samhällsvetenskapliga ämnen”*. Lund: Studentlitteratur.
[26] Merriam, S. B. (2009). “*Qualitative research: A guide to design and implementation”.* San
Francisco, California: Jossey-Bass.
[27] Damodaran D, et al. 2016. “*Performance Evaluation of Mysql And MongoDB”*. Databases International Journal on Cybernetics & Informatics (IJCI) Vol. 5, No. 2.
[28] Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., and Sears, R. *“Benchmarking Cloud Serving Systems with YCSB”*. In Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC 2010, Indianapolis, Indiana, USA, June 10-11, 2010 (2010), pp. 143– 154.
[29] DB Engines. https://db-engines.com/en/ranking
[30] Pymongo Documentation. https://api.mongodb.com/python/current/
[31] Pymysql Documentation. https://pymysql.readthedocs.io/en/latest/
[32] Database. https://searchsqlserver.techtarget.com/definition/database
[33] Haerder, T.; Reuter, A. (1983). "*Principles of transaction-oriented database recovery*". ACM Computing Surveys. 15 (4): 287. https://doi.org/10.1145%2F289.291
[34] Relational model. https://karthidb.wordpress.com/
[35] *"Structured Query Language (SQL)"*. Msdn.microsoft.com.
[36] Leavitt, Neal (2010). *"Will NoSQL Databases Live Up to Their Promise?"*(PDF). IEEE Computer.
[37] Fowler, Martin. "*Nosql Definition*". https://martinfowler.com/bliki/NosqlDefinition.html
[38] Lith, Adam; Mattson, Jakob (2010). "*Investigating storage solutions for large data: A comparison of well performing and scalable data storage solutions for real time extraction and batch insertion of data"* (PDF). Göteborg: Department of Computer Science and Engineering, Chalmers University of Technology. p. 70.

[39] *“NoSQL 2009"*. Blog.sym-link.com. 12 May 2009.
[40] *"RDBMS dominate the database market, but NoSQL systems are catching up"*. DB-Engines.com. 21 Nov 2013. http://db-engines.com/en/blog_post/23
[41] Leavitt, Neal (2010). *"Will NoSQL Databases Live Up to Their Promise?"*. IEEE Computer Society.
[42] S. Edlich. “*Nosql databases”*. http://nosql-database.org/, July 2012.
[43] Venkat Gudivada, Dhana Rao and Vijay Raghavan: "*NoSQL Systems for Big Data Management*", June 2014, Conference: 2014 IEEE World Congress on Services, At Anchorage, Alaska.
[44] Riak, a key-value database. URL: http://basho.com/products/#riak
[45] Redis, official website. URL: https://redis.io
[46] Official MongoDB website, URL: https://www.mongodb.com
[47] CouchDB, official website URL: http://couchdb.apache.org
[48] Apache Cassandra, URL: http://cassandra.apache.org
[49] Apache HBase, URL: https://hbase.apache.org
[50] What are graphs?: https://xlinux.nist.gov/dads/HTML/graph.html
[51] Official website of Neo4j, URL: https://neo4j.com
[52] Apache Giraph official website, URL: http://giraph.apache.org
[53] Neo4j.com. https://neo4j.com/developer/graph-database/
[54]Haerder, T.; Reuter, A. (1983). "*Principles of transaction-oriented database recovery*". ACM Computing Surveys. 15 (4): 287. https://doi.org/10.1145%2F289.291
[55] Gray, Jim (September 1981). "*The Transaction Concept: Virtues and Limitations*". Proceedings of the 7th International Conference on Very Large Databases.
[56] Microsoft Documentation. *“What is a transaction”*. https://docs.microsoft.com/en-us/sql/t-sql/language-elements/transactions-transact-sql?view=sql-server-2017
[57]”*The question of database transaction processing: an acid, base, nosql primer*”. Charles Roe. 2013, Dataversity.
[58] Pritchett, D. BASE: “*An ACID Alternative*”. ACM Queue 6, 3 (2008), 48–55.
[59] Acid vs. Base. http://www.dataversity.net/acid-vs-base-the-shifting-ph-of-database-transaction-processing/
[60] Brewer, E. A. “*Towards Robust Distributed Systems*” (abstract). In Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing, July 16-19, 2000, Portland, Oregon, USA. (2000), p. 7.

[61] Seth Gilbert and Nancy Lynch, "*Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services*", ACM SIGACT News, Volume 33 Issue 2 (2002), pg. 51–59
[62] "*Brewer's CAP Theorem*", julianbrowne.com, Retrieved 02-Mar-2010
[63] "*Brewers CAP theorem on distributed systems*", royans.net
[64] The CAP theorem. https://www.researchgate.net/publication/221462089_Social-data_storage-systems/figures?lo=1
[65] Davis Kerby. "*Why MongoDB is the way to go*". DZone. https://dzone.com/articles/why-mongodb-is-worth-choosing-find-reasons
[66] "*Turning MongoDB Replica Set to a Sharded Cluster*". Severalnines. https://severalnines.com/blog/turning-mongodb-replica-set-sharded-cluster
[67]"*GridFS & MongoDB: Pros & Cons*". Compose. https://www.compose.com/articles/gridfs-and-mongodb-pros-and-cons/
[68] MySQL JOINS. https://dev.mysql.com/doc/refman/8.0/en/join.html
[69] MongoDB Aggregation Pipeline. https://docs.mongodb.com/manual/aggregation/#aggregation-pipeline
[70] MongoDB Map-Reduce. https://docs.mongodb.com/manual/core/map-reduce/
[71] MongoDB Replication Documentation. https://docs.mongodb.com/manual/replication/
[72] MongoDB Transactions. https://www.mongodb.com/transactions
[73] Aslett, Matthew (2011). "*How Will The Database Incumbents Respond To NoSQL And NewSQL?*" (PDF). 451 Group (published 2011-04-04). Retrieved 2012-07-06
[74] Pavlo, Andrew; et al. (2016). "*What's Really New with NewSQL?*" (PDF). SIGMOD Record.
[75] Texas State Auditor's Office. *“Data Analysis, Methodology Manual, rev. 5/95”.* Texas State.
[76] Stake, R. (1995). *“The art of case research”*. Thousand Oaks, CA:Sage Publications

Annex

Benchmarking Results

Index Performance

Workload C - 1 thread - 100% Read

Recordcount	No. of Index	MySQL	MongoDB
400,000	**1**	1.733	1.54
900,000	**2**	3.491	3.237
1,400,000	**3**	5.37	4.87
2,000,000	**4**	7.68	6.83

Workload C - 2 threads - 100% Read

RecordCount	MySQL	MongoDB
400,000	1.445	1.34
900,000	3.01	2.87
1,400,000	4.36	4.1
2,000,000	6.106	5.81

Workload D - 2 threads - Read-Update-Insert

No. of Indexes	MySQL	MongoDB
1	0.83	0.94
2	0.97	1.02
3	1.47	1.35
4	2.14	2.07

Aggregation Framework

Workload C - 100 % reads

Aggregation Function	MySQL	MongoDB

JOIN / $lookup	8.65	7.84
GROUP BY / $groupby	8.22	8.36
SELECT / $project	7.98	8.23
HAVING / $match	8.42	8.47

Workload B - read / write (95:5)

Aggregation Function	**MySQL**	**MongoDB**
JOIN / $lookup	8.23	7.79
GROUP BY / $groupby	7.08	7.46
SELECT / $project	7.33	7.8
HAVING / $match	8.19	8.02

Application Integration

Workload A - read/update (50:50)

Recordcount	**MySQL**	**MongoDB**
500000	1.89	1.963
1000000	3.721	3.517
1500000	5.62	5.301
2000000	7.92	7.271

Workload F - read/read-modify-update (50:50)

Recordcount	**MySQL**	**MongoDB**
500000	2.47	2.69
1000000	3.981	4.13
1500000	5.98	5.677
2000000	8.37	7.930

Replication

Asynchronous single-threaded Replication (Figure 8.3.3.4(a))

MySQL(Master)	**MongoDB(Master)**	**MySQL(Slave)**	**MongoDB(Slave)**
4388	4473	283	313

Replication | Load, Workload A, Workload B (Figure 8.3.3.4(b))

Workload	**MySQL**	**MongoDB**
Load	189,800	212,345
95/5 read/write	276,126	294,883
50/50 read/write	142,347	167,332

Configuration

MongoDB.

To deploy MongoDB replication:

- Create the data directories for each replica set member using the following command:

  ```
  mkdir -p /PATH/TO/MONGODB/test-0  /PATH/TO/MONGODB/test-1
  /PATH/TO/MONGODB/test-2
  ```

- Start the `mongod` instances on different shell windows of a terminal window with the following command:

```
    mongod --replSet test --port 27017 --bind_ip localhost,
--dbpath /srv/mongodb/rs0-0 --smallfiles --oplogSize 128

    mongod --replSet test --port 27018 --bind_ip localhost,
--dbpath /srv/mongodb/rs0-0 --smallfiles --oplogSize 128

    mongod --replSet test --port 27019 --bind_ip localhost,
--dbpath /srv/mongodb/rs0-0 --smallfiles --oplogSize 128
```

These commands starts each instance of the replica set called `test`. The `--smallfiles` and `--oplogSize` settings are used to reduce the disk space consumed by each `mongod` instance.

- Connect to one of your instances through the `mongo` shell, specifying the port:

  ```
  mongo --port 27017
  ```

- Use `rs.initiate()` to initiate the replica set. This can be done on the mongo shell by creating a configuration object as follows:

```
    testConf = { _id: "rs0", members: [{ _id: 0,  host:
"localhost:27017" }, { _id: 1, host: "localhost:27018" },
          { _id: 2,  host: "localhost:27019" } ] }
```

```
rs.initiate(testConf)
```

- To display the configuration type:

```
rs.display()
```

MySQL

To set up MySQL replication:

- Edit the mysql configuration file (Mac OS X):

```
sudo nano /usr/local/mysql/my.cnf
Bind-address = 127.0.0.1
```

- Edit the file as follows:

```
bind-address  = 127.0.0.1
server-id     = 1
log_bin       = /usr/local/mysql/mysql-bin.log
binlog_do_db  = test
```

Save the file and exit.

- Restart MySQL and open up the MySQL shell (Mac OS X):

```
sudo /usr/local/mysql/support-files/mysql.server restart
mysql -u root -p
```

- Grant privileges to slave and lock database to prevent further changes:

```
GRANT REPLICATION SLAVE ON *.* TO 'slave_user'@'%' IDENTIFIED BY
'password';
FLUSH PRIVILEGES;
FLUSH TABLES WITH READ LOCK;
```

- Export the recently created database and unlock it:

```
mysqldump -u root -p --opt test > test.sql
UNLOCK TABLES; QUIT;
```

- Configure the slave.

```
CREATE DATABASE test;
EXIT;
mysql -u root -p test < /path/to/test.sql
```

- Edit the configuration file for the slave as the master but add the following:

```
sudo nano /usr/local/my.cnf
server-id   = 2
relay-log   = /usr/local/mysql/mysql-relay-bin.log
log_bin     = /usr/local/mysql/mysql-bin.log
Binlog_do_db = test
```

- Restart MySQL and activate the slave:

```
sudo /usr/local/mysql/support-files/mysql.server restart
mysql -u root -p
CHANGE MASTER TO MASTER_HOST='root', MASTER_USER='slave_user',
MASTER_PASSWORD='password', MASTER_LOG_FILE='mysql-bin.000001',
MASTER_LOG_POS=107;
START SLAVE;
```

YCSB

To configure YCSB on your machine you need to have Java installed.

To run a workload in MongoDB:

- Configure the database
  `java -cp PATH/TO/YCSB/mongodb/src/main/java/com/yahoo/ycsb/db/MongoDbClient.java`

- Configure the properties on a file with .dat extension
- Load and then run the workload with required properties

```
$ ./bin/ycsb load mongodb -s -P workloads/workloada -P large.dat -s > load.dat

$ ./bin/ycsb run mongodb -s -P workloads/workloada -P large.dat -s > load.dat
```

To run a workload in MySQL:

- Configure the database
  `java -cp PATH/TO/YCSB/jdbc-binding/lib/jdbc-binding-0.4.0.jar:mysql-connector-java-5.1.37-bin.jar com.yahoo.ycsb.db.JdbcDBCreateTable -P db.properties -n test`

- Configure the connection properties on a file. This will include the connection string, database username and password and all other properties you might want for the database.
- Add the JDBC driver to your systems path.
- Load and then run the workload

```
bin/ycsb load jdbc -P workloads/workloada -P
db.properties -cp mysql-connector-java.jar

bin/ycsb run jdbc -P workloads/workloada -P db.properties
-cp mysql-connector-java.jar
```

Pymysql Connection String

Must be initialized with shebang.

```
#!/usr/bin/env python3
import pymysql

#connect to db
conn = pymysql.connect(host='localhost', port=3306, user='root',
passwd='', db='test', cursorclass=pymysql.cursors.DictCursor)

try:
      with conn.cursor as cursor:
            #query
            Sql = “query”
            cursor.execute(sql)
      conn.commit()

finally:
      conn.close()
```

Pymongo Connection String

Must be initialized with shebang.

```
#!/usr/bin/env python3
import pymongo
from pymongo import MongoClient

client = MongoClient(‘mongodb://localhost:27017/’)

# connect to database
```

```
db = client.test

# connect to collection
collection = db[‘collection’]
```