UKRAINIAN CATHOLIC UNIVERSITY

BACHELOR THESIS

Performance comparison for different types of databases

Author:
Sofiia-Valeriia KHOLOD

Supervisor:
Andrii YASYNYSHYN

A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science

in the

Department of Computer Sciences
Faculty of Applied Sciences

Lviv 2021

Declaration of Authorship

I, Sofiia-Valeriia KHOLOD, declare that this thesis titled, "Performance comparison for different types of databases" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences

Bachelor of Science

Performance comparison for different types of databases

by Sofiia-Valeriia KHOLOD

Abstract

This thesis is focused on database management systems. At first, we will go through relational and non-relational databases, their principles and use cases. After that, we made a research on which databases are currently in trend. Next, we constructed a testing application that dynamically visualizes results of conducting CRUD tests on MongoDB, PostgreSQL and MySQL. At the end, we provide the interpretation of results.

The final conclusion of this thesis is that MySQL is not longer an option for modern applications, while PostgreSQL and MongoDB are on an equal lever, each is good for its type.

Acknowledgements

First of all, I would like to thank my supervisor Andrii Yasynyshyn for the research idea and support. Special thanks also goes to Serhii Puronen for consulting me about the DevOps part of this thesis.

Next, I thank my parents for the ability to study at Ukrainian Catholic University. Without them, I would not become who I am now and this thesis would not exist (neither would I).

I also thank Ukrainian Catholic University and especially Applied Science Faculty for the Computer Science program. I thank all lecturers who taught me and every person who is involved in this program and directly my education. At this university, I met people who changed my life forever. I am happy that back then, in 2017, I chose this university and this program (or did it choose me?).

I sincerely thank Nazar Kholod, who took my hand 4 years ago and showed me what UCU is. I thank him for all the times he guided, supported and appreciated me.

Last but not least, I thank my friends, family and everyone who was always there for me (even when I am writing this text). Life would be much harder without you, thank you for being a part of it.

Contents

List of Figures

List of Abbreviations

DBMS	**D**ata**b**ase **M**anagement **S**ystem
RDBMS	**R**elational **D**ata**b**ase **M**anagement **S**ystems
ACID	**A**tomicity, **C**onsistency, **I**solation, **D**urability
BASE	**B**asically **A**vailable, **S**oft state,**E**ventual consistency
CRUD	**C**reate **R**ead **U**pdate **D**elete
GUI	**G**raphical **U**ser **I**nterface
CLI	**C**ommand-**l**ine **I**nterface
CPU	**C**entral **P**rocessing **U**nit
JDBC	**J**ava **D**ata**b**ase **C**onnectivity
IO	**I**nput/**O**utput

Chapter 1

Introduction

Nowadays, it is hard to imagine a modern application that is able to function without some kind of data storage. From social networks to self-driving cars, from research stations in Antarctica to bank transactions, from Youtube videos to nuclear attack codes, the data in that way or another surrounds us everywhere we go. The more digitalized our life becomes, the more data we use. More data brings more challenges. Twenty-first-century organizations have to go through at least three of them: the quality of data has to be good, specifically trained engineers that can properly work with it, and, as an endpoint - some storage to save it and efficiently retrieve for future usage. But, back then, these were totally different.

The modern history of Big Data started in the 70s in paper by Codd, 1970. He introduced such concepts as relation, constraints, normal forms, and query language, which are now widely used in database development. Since then, many Relational Database Management Systems were created, but mostly, even now, we use 50-year-old concepts. These outdated concepts bring such problems as: the data location has to be on one server, complex SQL queries because of the need to satisfy normal forms, no support for complex data structures such as arrays, maps, etc, and no dynamic data schema change. They led to colossal performance issues as well as to difficulties in development. However, these were not too critical until the number of Internet users grew significantly in the 00s. Applications are now required to handle large loads of data, and the need for innovations was growing rapidly.

This way, in 1998 term NoSQL was used for the first time by Carlo Strozzi to name his "relational" database that did not use SQL. NoSQL was designed to satisfy the imperfections of RDBMS, with the ability to handle loads of unstructured data efficiently. In the following ten years, such projects were introduced as Big Table by Google, Hadoop, DynamoDB by Amazon, and 10Gen with its MongoDB.

The progress is inevitable and fast. The more it goes, the more solutions are there to choose from. Both RDBMS and NoSQL are designed with entirely different ideas in mind, first one - to meet ACID principles, and the second one - BASE semantics [see Chapter 3]. However, nowadays, developers often get stuck when the question about the database management system arises. Which one is the best option in that particular case? Will it be able to handle the application load? Will its throughput be enough? This thesis aims to answer these questions by performing a benchmark on a few popular databases, both relational and not relational, and comparing them. We will go through designing an application that is able to perform CRUD performance tests on MySQL, PostgreSQL, and MongoDB with dynamic visualization of results. At the end, we will analyze the results and compare them to other benchmarks.

Chapter 2

Related work

In this chapter, we are going to review few existing database performance comparison papers.

A study by V. Abramova and Furtado, 2015 compares five NoSQL databases: Cassandra, HBase, MongoDD, OrientDB and Redis. The database choice for the benchmark is sparse; however, the research lacks the description of each selected database. Also, there is no explanation of which data was used and how it was selected, same as arguments for performance comparison methods.

A master thesis by Kolonko, 2018 has sufficient description of data, its selection, the justification of databases choice, and results. The drawback is there are a lot of not that important details. Such details are provided, for instance, about browsing the most popular databases on the Internet and mentioning how the search was done. This work compared OracleDB and MongoDB; the choice is justified, again, with lots of details, but the good side is that the text is clear and easy to follow, but about 30% of the research can be cut off, and that would not make much difference.

The paper by Nepaliya and Gupta, 2015 compares two NoSQL databases of document type: RavenDB and CouchDB. It is not stated why such a selection was made. Also, not stated the data used and the type of conducted tests. Overall, the paper leaves many questions behind. The good point the authors made is the description of RavenDB and CouchDB. They described their pros and cons, as well as their standing out features.

Another master thesis by Hadjigeorgiou, 2013 compares MySQL and MongoDB. The thesis is well structured. It also has all the needed descriptions, such as the schema of data used, the queries performed, test configurations, and measurement options. The database choice is also justified. Two minor drawbacks are that the selected databases were not described in detail, and their versions used for the test were not mentioned, but, other than that, the benchmark was done well.

Research made by Li and Manoharan, 2013 compares seven databases: MongoDB, RavenDB, CouchDB, Cassandra, Hypertable, Couchbase, and Microsoft SQL-Express. Despite such vast databases choice is very approving, it is not justified, and the paper lacks the description of each of the selected databases. The schema used for the benchmark is also neglected.

Overall, there are loads of papers written on performance comparison themes. Many of them lack the core needed descriptions, but almost all of them have a good explanation of the results. Each of the studies reviewed here compares different databases and has a good visualization of results. A vast majority of studies compare at most two databases; there are some exceptions, however. Our solution, however, compares three different databases of two different types and has a real-time dynamic visualization of results. But, on the other hand, my tests are simpler than the ones made in the reviewed articles.

Chapter 3

Background

The database is simply a storage unit for somehow related information that is operated by a database management system. To conduct a benchmark on different types of DBMS, they first need to be appropriately categorized. The basic division is made into two categories: relational and non-relational. In this chapter, we will shortly go through each of the types and their core ideas, concepts, and use cases.

3.1 Relational databases

A relational database is a set of structured tables where data is stored using rows and columns. Its tables are based on a fixed schema, a sample of which is given on the Figure 3.1.

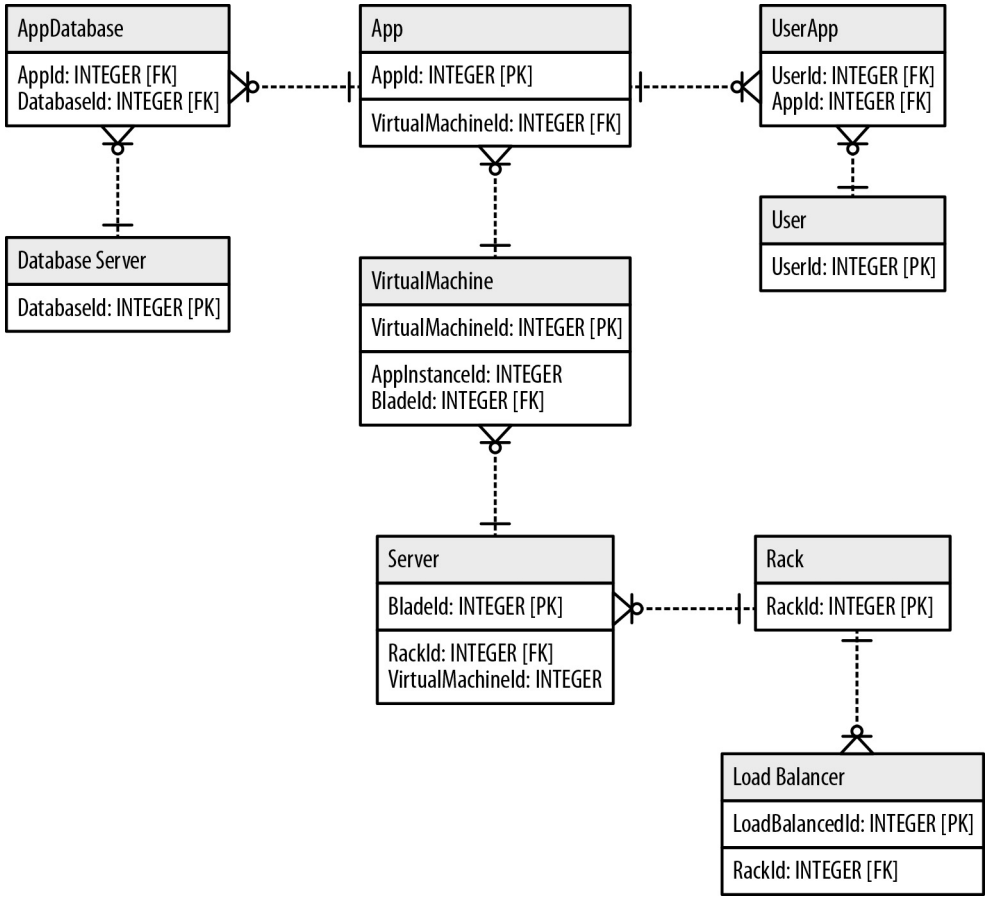

FIGURE 3.1: An example of data center management domain, made in relational database style, adopted from an article by Sasaki, 2018

Each table in relational database has to have a unique row identifier (**primary key**), that later forms an index to the table. Another important notion of RDBMS is **foreign key**, which is the reference to a specific row in another table.

3.1.1 Normal forms

The process of normalization (fitting tables to satisfy normal forms rules) is another core concept of relational databases, which was also presented by Edgar Codd in 70s. At that time, he suggested three types of normal forms. Later, the Boyce–Codd normal form was introduced in paper [Codd and Boyce, 1974]. Overall, there are seven normal forms in a hierarchical structure.

As normal forms are not that relevant for this research, we will not dig into details for this topic. The rule of thumb for the normal form is that your schema is not normalized, if there is any duplicated data.

3.1.2 ACID

A transaction is any change in a RDBMS database. To satisfy the reliability of such transactions, ACID properties were designed, and now every relational system should be compliant with them. ACID principles are:

- **Atomicity**: each transaction has to be identified as a single unit, which can either fail or success
- **Consistency**: data has to be kept in a valid state
- **Isolation**: in case transactions are executed concurrently, the order has always to be the same as it would be in a sequential manner
- **Durability**: if the system fails, transaction that was made before the crash, remains in the same state

ACID principles allow a RDBMS to be used for sensitive and important information storage, such as bank transactions.

3.1.3 SQL

SQL is a query language for manipulations with databases. The result of a query (statement in SQL) is a table, or, in other words, result set. Its main advantages are the ability to accessing many records in one statement and that it can access data with or without an index, unlike its ancestors, that could only work with indexes.

In order for the database to be called relational, it has to be SQL-compliant. It is not required for it to adhere to all rules and specifications of SQL, but to be fully compliant, it has to.

3.1.4 Conclusions

RDBMS were created for the purpose of having a consistent and reliable data storage on one server. Due to this and its ACID compliance, such systems are perfect when one has to store sensitive data, for example, healthcare data or bids on auctions, where the order of transactions is crucial. Their main drawback, however, is that they are not easily partitioned and replicated to clusters, and because of that, they become really slow when the amount of data becomes big.

3.2 Non-relational databases

NoSQL was designed on the contrary to RDBMS. It violates its principles, and creates new ones. Data is no longer required to be formed into a table, instead, it could be anything from document to graph. ACID principles are omitted and new ones were created.

On Figure 3.2 a schema for the same data as on Figure 3.1 is displayed, but it is much simpler and preserves data in its initial form.

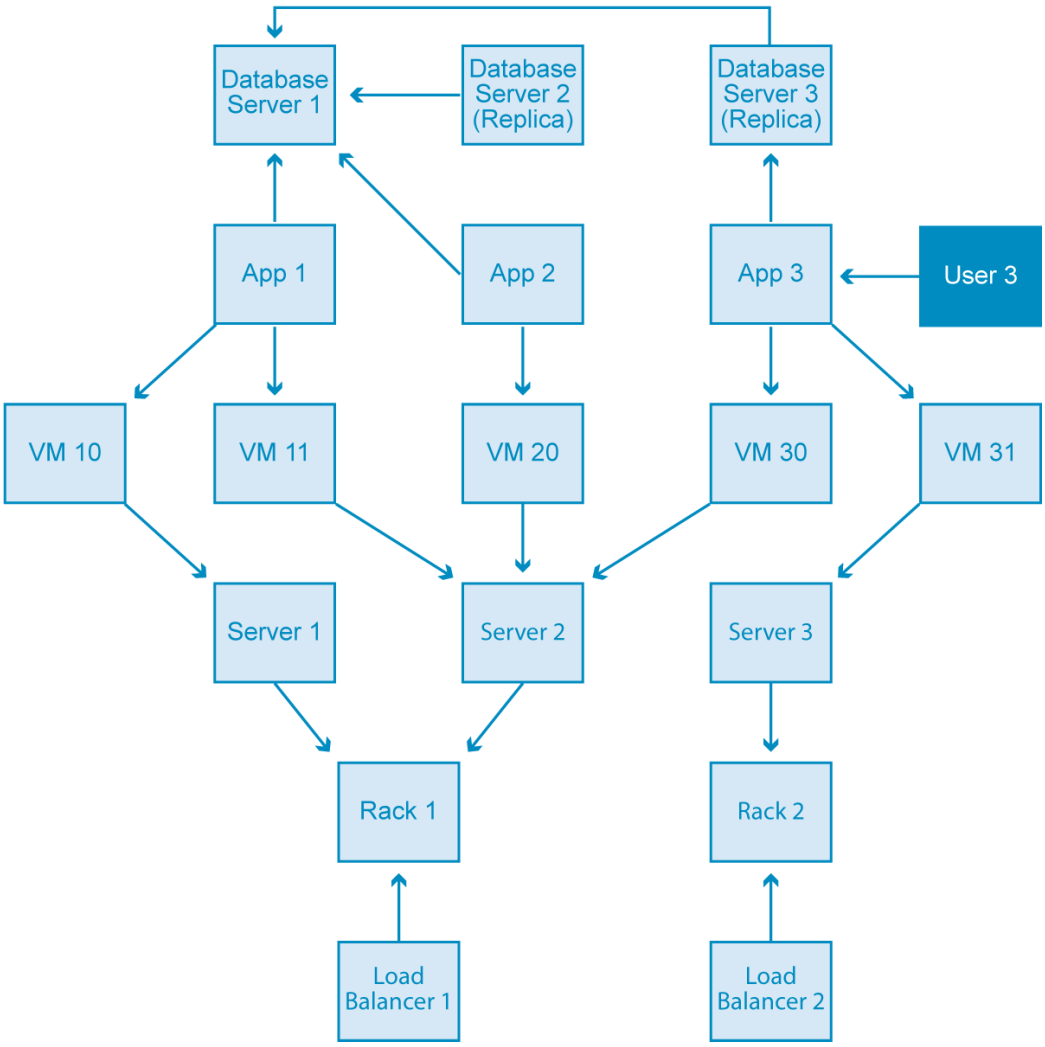

FIGURE 3.2: An example of data center management domain in its natural from using graphs, taken from the article by Sasaki, 2018

3.2.1 BASE

BASE is a softer version of ACID principles, that takes into account the flexibility of NoSQL. BASE consists of:

- **Basic Availability**: data is spread over the clusters with replication
- **Soft State**: consistency of data is not handled by the database
- **Eventual Consistency**: data does not have to be always consistent, instead, ensure it would converge at some point in the future

These principles mean that NoSQL databases do not guarantee that the records will be written to memory in the same order that the write request was made. Also, another problem in NoSQL are concurrent writes [see Chapter 5 of Kleppmann, 2017]. There are many ways to control such cases, and each database has its own one. For example, Cassandra has the simplest concept of Last Write Wins (each request has a timestamp of the time it was made).

3.2.2 CAP

Another core goal of NoSQL regards the CAP theorem. CAP means:

- **Consistency**: all users have to see the same data at the same time
- **Availability**: any request made to the databases, gets a response, no matter if any of the nodes are down
- **Partition tolerance**: the system has to continue to work even when there are any communications breakdowns

CAP theorem states that a system always has to strive to the three of its concepts, but it always never the case. But, since partitioning may happen whether it is expected it or no (any network fault) and it's a kind of fault that you cannot control, this theorem has to be correctly interpreted as: "...either Consistent or Available when Partitioned", as stated a in book by Kleppmann, 2017.

3.2.3 Types

There are four main types of NoSQL databases. However, many of today's top non-relational DBMS are a combination of several types listed below.

Document type

This type saves data mostly in JSON, XML or BSON documents. It is very flexible because the schema of documents does not have to be the same, and it can be easily adapted to the change in application or its requirements. Because of this, they are easy to work with, and this explains the popularity of databases of this type among the developers.

Another reason for their popularity is an intuitive data model - the DBMS can save the data in JSON format, as it is used all over the Web in HTTP protocol to transfer data. The most famous representative is MongoDB, which recently gained much popularity.

Key-value type

This type is the simplest of all, as it does exactly what its name suggests: saves values in a map, where the key is the index and the value is, in fact, a row value. It resembles a relational data storage with only two columns - name and value. Because of their simplicity, they are highly optimized and are leaders when it comes to operations execution time.

Such databases are a good storage to save data that does not have to be structured, such as shopping carts, or, more commonly, cache. The most popular database of such type is Redis, which positions as world's fastest database.

Column-oriented type

Column-oriented data storages are a complete difference to relational ones. While the RDBMS save data in rows and read row by row, column-oriented, as the name suggests, save it in columns. They are a common solution when it comes to analytic data, when there is a low number of long columns.

Columnar databases are very fast on column reads and their aggregation. Their drawback is that operations on few rows are not optimized, as well as they are hard to achieve strong consistency on writes, because one write request may require several writes to disk.

Graph type

This type is an ideal solution when the initial data is highly coupled, for instance, social network friends. Each element is a node in graph, and connections between nodes are called relationships. Such DBMS are optimized for related data, but performed worse on loose data.

Most widely used database of this type is Neo4j, which is used by Nasa and eBay [*Who uses Neo4j?*].

Chapter 4

Methods

4.1 Performance tests types selection

We decided to measure such metrics as response time and throughput. The type of tests was chosen based on their relevance to the real-life problems. Because of that, we have decided to conduct tests that resemble social network posts or news management - when there are many single reads and little deletes.

4.2 Databases

The databases choice for the benchmark is an integral part since poor and unjustified choice leads to poor results. Multiple parameters were taken into consideration when conducting this part of a thesis: popularity among businesses, developers, and their benchmark relevance, so that it would be a fair fight.

4.2.1 DB-engines ranking

The first thing that the web search has led to was the website DB-engines and its ranking [DB-engines, 2021], which monthly gathers statistics about the database usage. Their method is basically to count mentions on websites, using search engines such as Google and Bing. They also use Google Trends to see which databases gain popularity and which lose it. Interestingly, they also count the mentions of database names in tweets (the suggestion is to count the, let us say, informal popularity). As for May 2021, the top 10 consisted of Oracle, MySQL, Microsoft SQL Server, PostgreSQL, MongoDB, IBM Db2, Redis, Elasticsearch, SQLite, and Microsoft Access.

371 systems in ranking, May 2021

Rank			DBMS	Database Model	Score		
May 2021	Apr 2021	May 2020			May 2021	Apr 2021	May 2020
1.	1.	1.	Oracle	Relational, Multi-model	1269.94	-4.98	-75.50
2.	2.	2.	MySQL	Relational, Multi-model	1236.38	+15.69	-46.26
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	992.66	-15.30	-85.64
4.	4.	4.	PostgreSQL	Relational, Multi-model	559.25	+5.73	+44.45
5.	5.	5.	MongoDB	Document, Multi-model	481.01	+11.04	+42.02
6.	6.	6.	IBM Db2	Relational, Multi-model	166.66	+8.88	+4.02
7.	7.	↑ 8.	Redis	Key-value, Multi-model	162.17	+6.28	+18.69
8.	8.	↓ 7.	Elasticsearch	Search engine, Multi-model	155.35	+3.18	+6.23
9.	9.	9.	SQLite	Relational	126.69	+1.64	+3.66
10.	10.	10.	Microsoft Access	Relational	115.40	-1.33	-4.50

FIGURE 4.1: An example of data center management domain in its natural from using graphs, taken from the article by Stack Overflow, 2020

The main drawback of this ranking is that it shows the popularity of databases among businesses, including old and outdated systems. Many organizations stay with old-time solutions, such as Oracle, because it is more expensive and time-consuming for them to migrate to newer and modern solutions. For this reason, we decided to take another ranking, that is described in the next section.

4.2.2 Stack Overflow developers survey

This survey [Stack Overflow, 2020] is conducted on a yearly basis, and it shows the current trend in technologies, which developers currently use, and which technologies are most wanted between them. There are many parts, such as personal information (gender, age, etc) and, what makes the most interest - overview of technologies.

On [Figure 4.2] that shows the most popular databases the results (by counting the percentage of respondents who work with it) are more intuitive and indicative than in the previous ranking: top spots are preserved by databases that are popular and widely used.

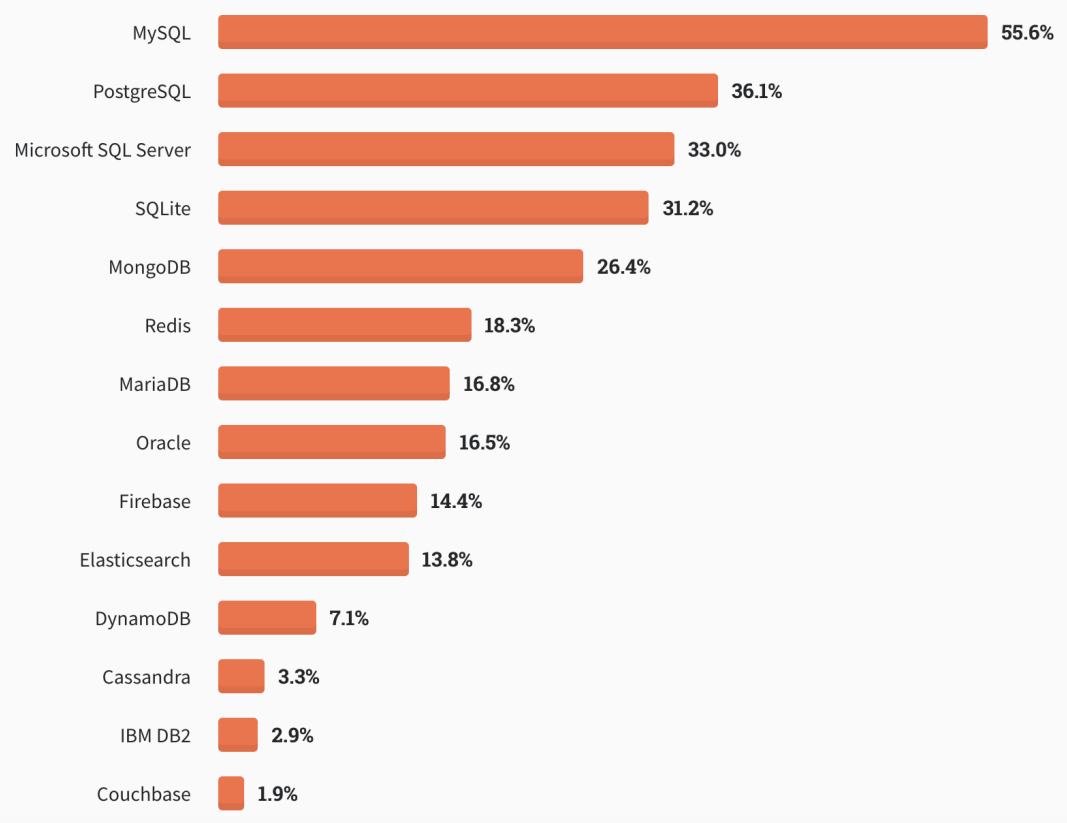

FIGURE 4.2: Top most used DBMS used by all respondents from Stack Overflow, 2020, based on 49,537 responses.

Apart from the metric on the previous chart, Stack Overflow also asks its respondents which technologies are most loved (developers work with them and want to continue), dreaded (would not like to continue) and wanted (do not develop but would love to). This metric is also important, since it shows a trend on technologies, whether it goes up or down.

As can be seen on Figure 4.3, a few of top position from ranking 4.1, are most unwanted for developers to continue working with. A surprise was that MySQL is

in a list of most dreaded databases, since it seems to be relatively popular and one of the most used ones. Top places by Oracle and IBM DB2 are not surprising since, as was mentioned before, these databases are an example of cases when they are left for backward compatibility and are expensive to be migrated into newer solutions.

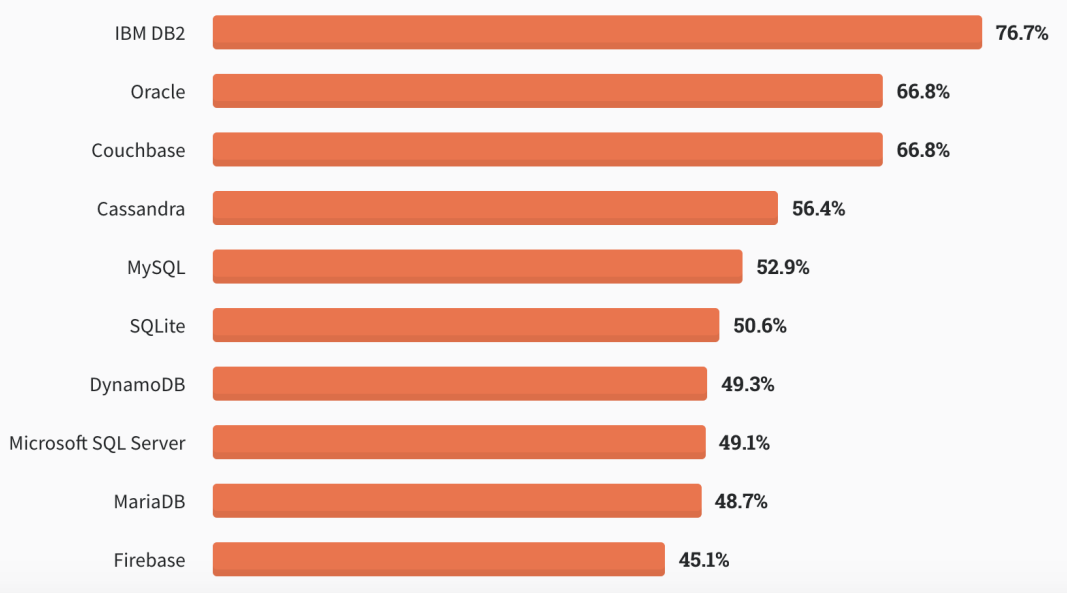

FIGURE 4.3: Top 10 most dreaded DBMS from Stack Overflow, 2020

4.2.3 Conclusion

Based on databases investigation, we have selected three of them for the research: MySQL, PostgreSQL and MongoDB. MySQL was selected since there is a negative trend for it in terms of popularity, and it is interesting to check why so. PostgreSQL is currently on hype, and, what is more, it is being compared with MySQL a lot. MongoDB is also very popular, and it is of one of the most reasonable types of NoSQL databases that can be compared with relational ones.

For the reason of making a correct comparison to relational databases, graph type, wide column and key-value were selected out. Graph type works better when the data is very connected, wide column - on analytic data. Key-value did not make much sense to test as it would take the first spot in performance measures since it is one of the simplest types, hence, its speed is the best.

4.3 Selected databases

In this section we will go through a short overview of the selected databases to identify their main pros, cons and use cases.

4.3.1 MySQL

MySQL is a representative of a relational database management system. It was build to be fast and easy to use. Moreover, MySQL can be build using different engines, and based on that, its features may be differ. For instance, MySQL is only fully SQL-compliant when it is build on InnoDB engine.

Pros and cons

Advantages of MySQL include: easy to use, stable performance, able to handle large-sized databases and being open source. It also very secured, especially when using InnoDB.

Its major drawback is that after Oracle purchased it in 2009, it became not fully open sourced, and since then there are different editions of MySQL, paid ones and limited free [AWS, 2021a]. Although, after that, there were many improvements and innovations in the development of this RDBMS.

Use cases

MySQL is mostly used for eCommerce platforms. Its engine is actually one of the most popular when it comes to transactional ones. It is also used by such companies as Facebook, Nokia and Nasa.

4.3.2 PostgreSQL

PostgreSQL is an example of an open source object-relational database system. Its official website states that it is "The world's most advanced open source relational database" [PostgreSQL, 2021].

It fully satisfies ACID principles, as well as is almost SQL-compliant. PostgreSQL, on the contrary to MySQL, was build to be feature-rich. It both supports relational (SQl), and non-relational (JSON) querying. Also, since it is object-relational, it supports tables inheritance.

Pros and cons

Major PostgreSQL advantage is that it is open source, and literally everybody can commit to the development of this DBMS, as well as propose their own solutions to bugs. It is also enhanced with features, and has a high fault tolerance database. [AWS, 2021b] One of its drawbacks is that because it has more features than MySQL, it is slower in terms of performance.

Use cases

PostgreSQl, because of its fully ACID-compliance, is suitable for financial structures. It has gained a lot of popularity recently, and it is now top two in most popular databases [Figure 4.2]. Such giants as Twitch, Spotify, Reddit and IMDb use it.

4.3.3 MongoDB

MongoDB is a NoSQL database of document type that saves data in JSON format. Its official website states that it is "The most popular database for modern apps" [MongoDB, 2021]. Since MongoDB is a NoSQL database, it has all principles of them. Instead of tables in relational databases, MongoDB has so-called documents, where any piece of JSON data can be written. It saves data in a binary JSON - BSON.

Pros and cons

MongoDB pros include: scalability, replication, fast performance, distributed and non-blocking IO. One of cons is that MongoDB does not support transactions, so it cannot be used for any financial structure.

Use cases

As a NoSQL database, MongoDB is used in case when the amount of data is big, and where you do not need atomicity as for transactions. It is used by such companies as Forbes and Toyota.

Chapter 5

Solution overview

In this chapter we will present the solution which was developed in this research.

5.1 Table

As we decided to do simple CRUD tests, next step was to decide on what data the databases will be tested. The requirement was that the data contained all the main types: an integer, float, date, bool, short text and long text. Based on this, we have selected the model of data that most people use on a daily basis - a post in social network, or something that resembles it. We found the Amazon database of reviews for their products and took it as a base for our table schema. Then, the data was cleaned and shortened to fit the requirements. The result schema can be seen on Figure 5.1.

reviews		
FK	id	char(50)
	product_title	char(255)
	star_rating	float
	helpful_votes	int
	total_votes	int
	verified_purchase	boolean
	headline	char(255)
	body	text
	date	date

FIGURE 5.1: The schema of table that will be tested.

It is worth mentioning, that this schema was used only for MySQL and PostgreSQL, since MongodDB does not have a fixed schema, as well as limited char fields.

5.2 Tools

Apart from databases, we used multiple tools for testing and visualisation.

5.2.1 JMeter

Apache JMeter is open source application written in Java that was designed for load and performance testing. It is mostly used for testing load on servers, but can also

be used for databases test via JDBC (Java Database Connectivity). PostgreSQL and MySQL can be connected to JMeter generically, and the tests can be executed using queries. For MongoDB we have written Java code to have the same functionality as for PostgreSQL and MySQL (see Appendix A). Not only can JMeter display results in its GUI interface, but it can also automatically save data to whatever you tell it to using listeners.

5.2.2 Grafana

Grafana is a popular open source framework for dynamically displaying and running analytics, specifically time series data. It is mostly used for Data Science. Grafana allows to query, aggregate and display data dynamically over a certain period of time.

5.2.3 InfluxDB

InfluxDB is an open source NoSQL database, written in GO, optimized for fast storage and retrieval of time series data. It is basically a bridge that connects JMeter and Grafana. We have set up JMeter to write results to it and Grafana to retrieve and display them on its dashboard.

5.3 Tests

We have decided to execute simple CRUD tests that are as follows:

- **CREATE** - count how many records were created in a fixed time (or average requests per second), initial setup: truncate table
- **READ** - count how many records were read in a fixed time, initial setup: truncate table and create one record
- **UPDATE** - count how many records were updated in a fixed time (boolean field was being updated for simplicity purposes), initial setup: truncate table and create one record
- **DELETE** - count how many records were deleted in a fixed time, initial setup: truncate table and create enough records

Note: all databases, taking part in this research, have batch create/delete, etc, but only single operations were executed on them.

5.4 System setup

The whole system schema is displayed on Figure 5.2. Everything is deployed on Microsoft Azure in one private subnet (only visualization server has public IP for it be accessible from the Web Browser) on such instances:

- **Databases server**: Standard D2s v3 (2 vcpus, 8 GiB memory)
  - **MySQL** version: 8.0.24
  - **PostgreSQL** version: 13.3
  - **MongoDB** version: 4.4.6

- **Testing environment**: Standard B2s (2 vcpus, 4 GiB memory)
  - **JMeter** version: 5.4.1
- **Visualisation server**: Standard B2ms (2 vcpus, 8 GiB memory)
  - **Grafana** version: 7.2.0
  - **InfluxDB** version: 1.8.2

Databases were installed using defaults in separate Docker containers and without any configurations made for improving their performance. They were installed with tag:latest to get the latest stable release version. This was done not to give any of the test subjects advantages.

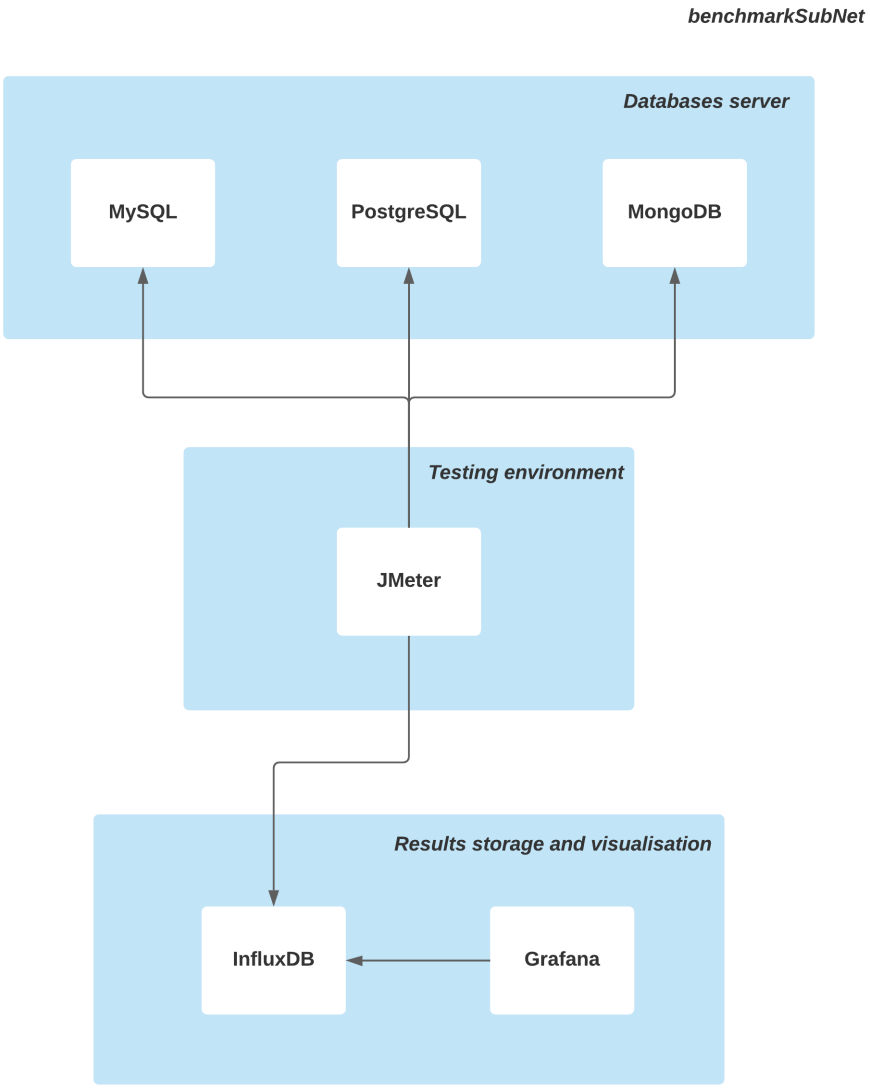

FIGURE 5.2: Benchmark application, deployed on Microsoft Azure. Each blue box represents different virtual machine.

The whole testing flow is:

1. The test is started using JMeter CLI.
2. JMeter connects to the database instance and to specific database.
3. JMeter performs initial setup and then testing.
4. During the test execution, JMeter connects to the Grafana server and writes results to InfluxDB.
5. Grafana constantly reads InfluxDB (read time is configurable) and displays charts.

Chapter 6

Results

In this chapter we will present a sample of results that we got, to take a look at all charts and launch demo - see Appendix A.

6.1 Read

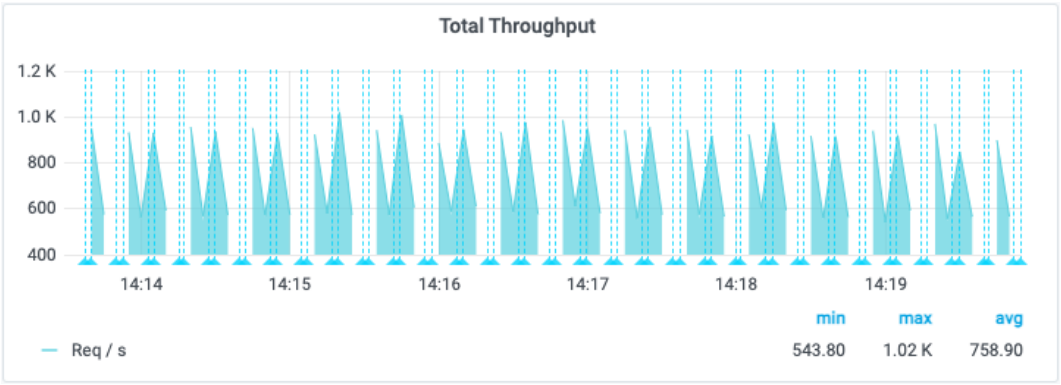

FIGURE 6.1: PostgreSQL READ test, throughput metric

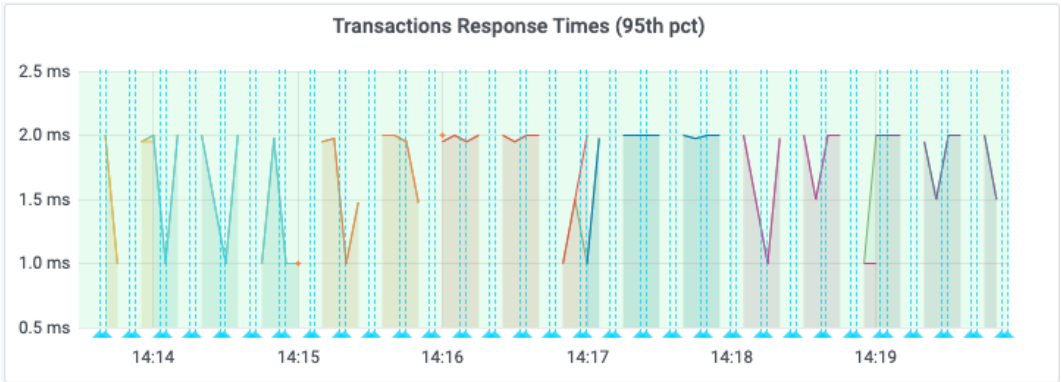

FIGURE 6.2: PostgreSQL READ test, throughtput metric

PostgreSQL showed good and stable performance. Its response time is pretty much the same, it does not vary a lot. The average of this metric is 1.74ms, min and max - 1.00ms and 2.00ms respectively. On Figure 6.1 we can see that PostgreSQL has a higher throughput at the beginning of the test, and looses some performance in the middle of it, and then gains it back.

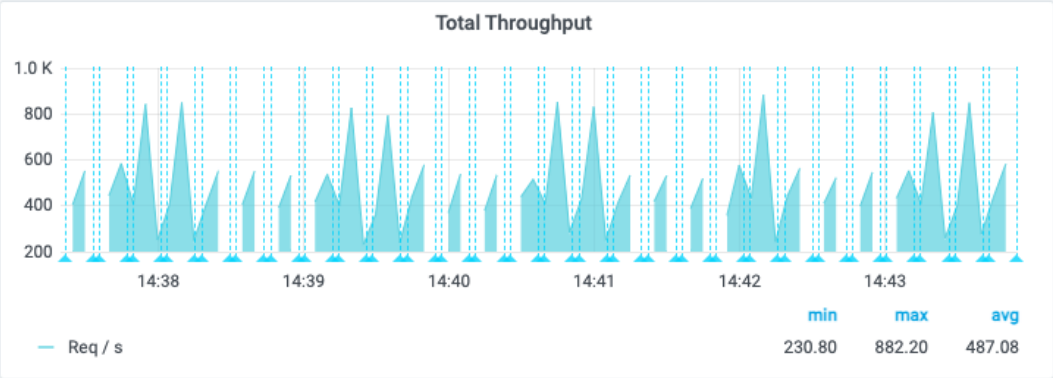

FIGURE 6.3: MongoDB READ test, throughput metric

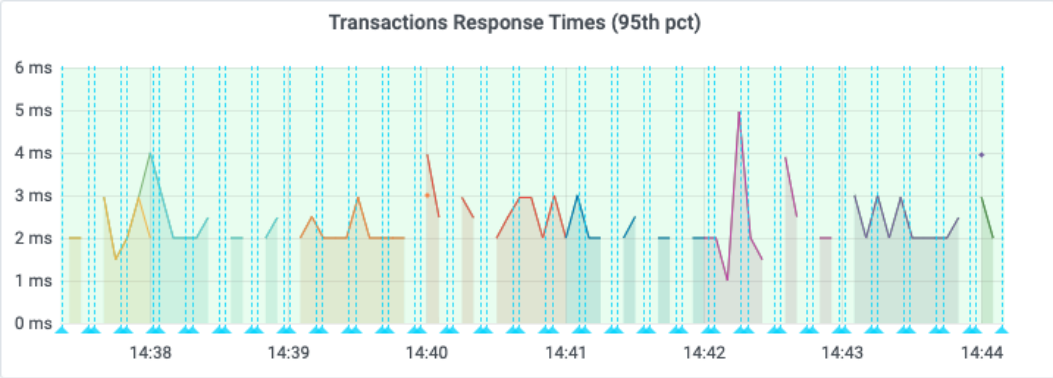

FIGURE 6.4: MongoDB READ test, response time metric

MongoDB also showed promising results, although they are a bit vise versa from what we saw for PostgreSQL: it showed the best throughput in the middle of the tests. The surprising fact is that it showed a little worse performance than PostgreSQL. Its response time is also worse: min is 1ms, average - 2.36ms, and max - 4.95ms.

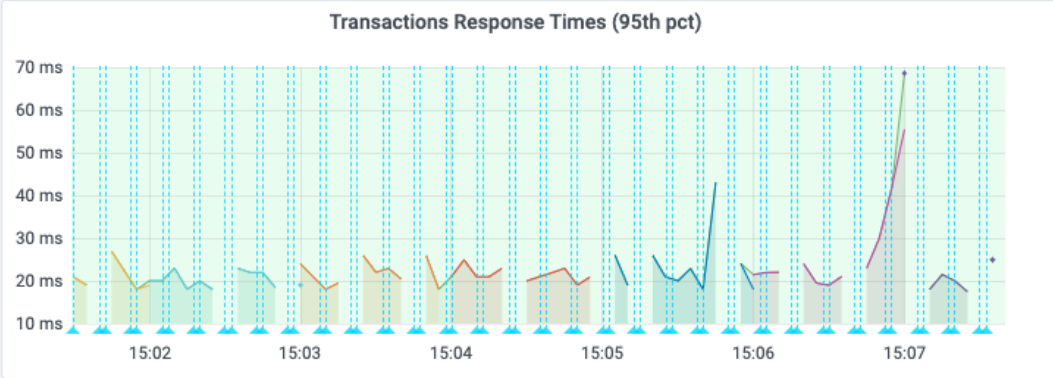

FIGURE 6.5: MySQL READ test, response time metric

MySQL test has a bit shocking results. Its performance is awful, the average response time is 68.60ms, which is 29 times worse than in MongoDB and 39 than PostgreSQL.

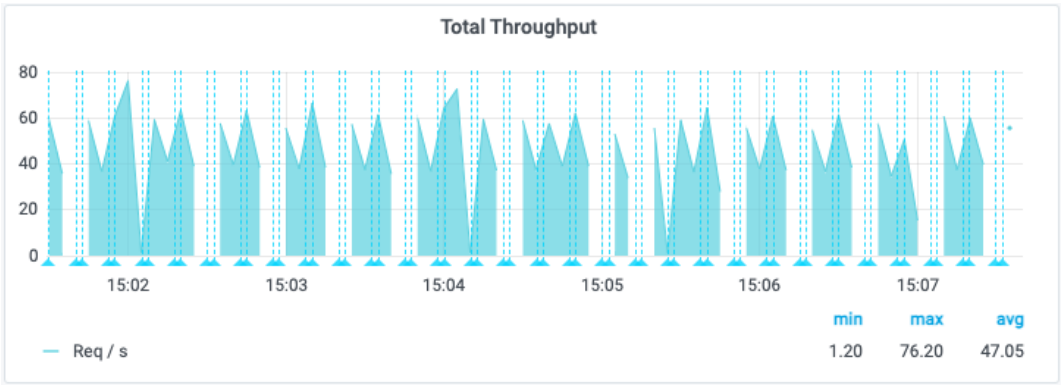

FIGURE 6.6: MySQL READ test, throughput metric

	min ^	max	avg
all	17.47 ms	68.60 ms	22.96 ms
request 12:07 16-05	17.47 ms	68.60 ms	27.49 ms
request 12:01 16-05	18.00 ms	26.90 ms	21.04 ms
request 12:02 16-05	18.00 ms	22.95 ms	20.30 ms
request 12:03 16-05	18.00 ms	25.90 ms	21.76 ms
request 12:05 16-05	18.00 ms	42.95 ms	23.77 ms
request 12:06 16-05	18.95 ms	55.40 ms	27.19 ms
request 12:04 16-05	19.00 ms	24.90 ms	21.49 ms

	min ^	max	avg
all	1.00 ms	2.00 ms	1.74 ms
request 11:13 16-05	1.00 ms	2.00 ms	1.72 ms
request 11:14 16-05	1.00 ms	2.00 ms	1.50 ms
request 11:15 16-05	1.00 ms	2.00 ms	1.68 ms
request 11:16 16-05	1.00 ms	2.00 ms	1.85 ms
request 11:17 16-05	1.00 ms	2.00 ms	1.89 ms
request 11:18 16-05	1.00 ms	2.00 ms	1.60 ms
request 11:19 16-05	1.50 ms	2.00 ms	1.88 ms

FIGURE 6.7: MySQL and PostgreSQL READ test results side by side

6.2 Create

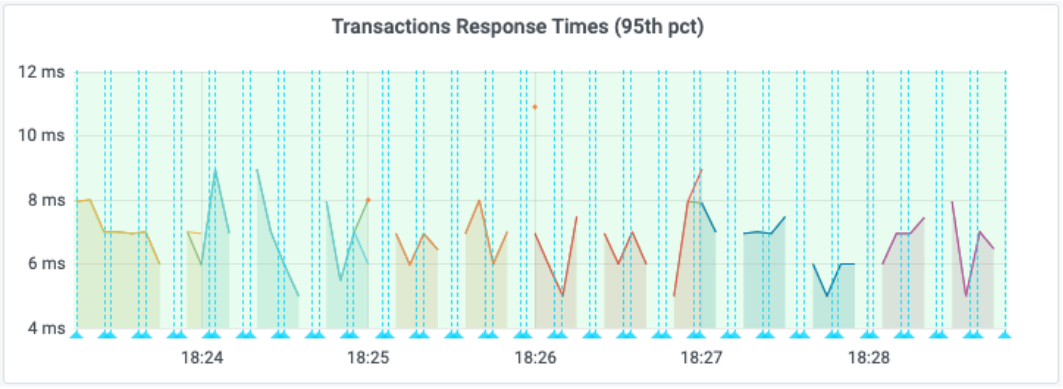

FIGURE 6.8: PostgreSQL CREATE test, response time metric

PostgreSQL has much worse performance on creating records than on reading them. In 5 minutes test, 70397 records were created (while on read this number was almost 5 times bigger). The lowest number of records created in a second is 3.8.

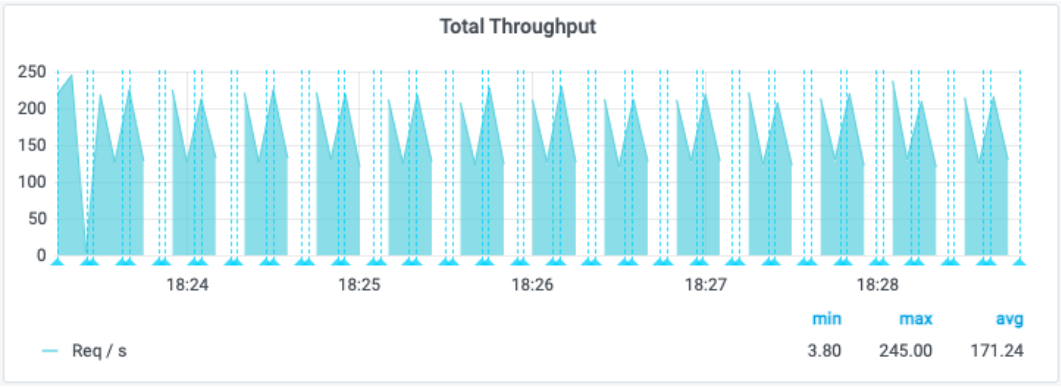

FIGURE 6.9: PostgreSQL CREATE test, throughput metric

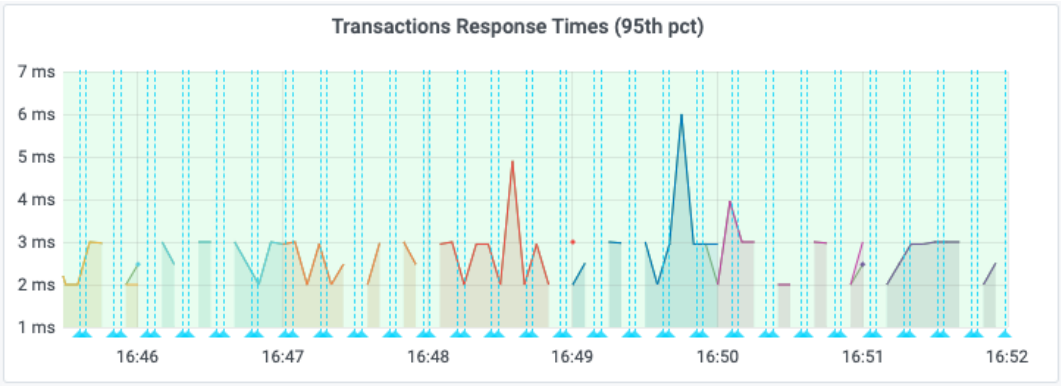

FIGURE 6.10: MongoDB CREATE test, response time metric

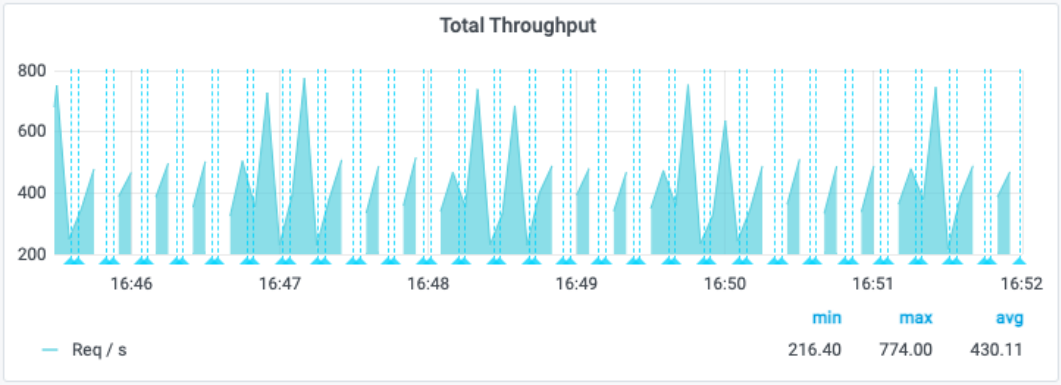

FIGURE 6.11: MongoDB CREATE test, response throughput metric

MongoDB here has much better results than PostgreSQL, with minimal write throughput almost the same as the maximum speed that PostgreSQL showed (min: 216.4 on one side and max:245.0 on the other).

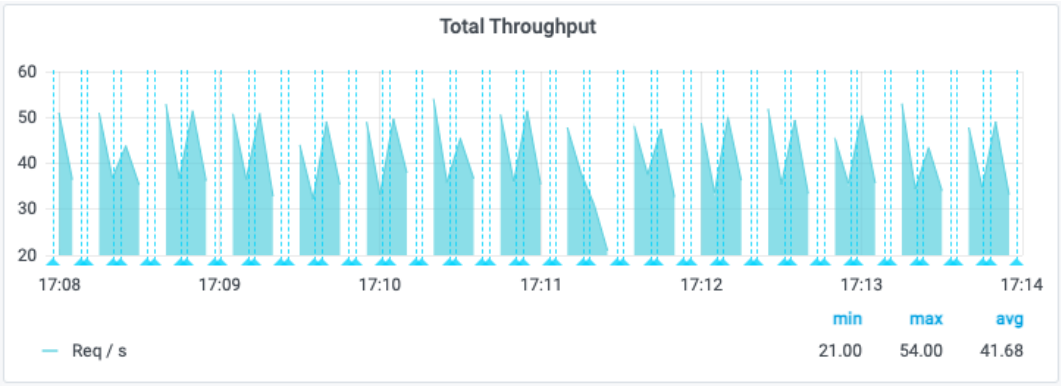

FIGURE 6.12: MySQL CREATE test, throughput metric

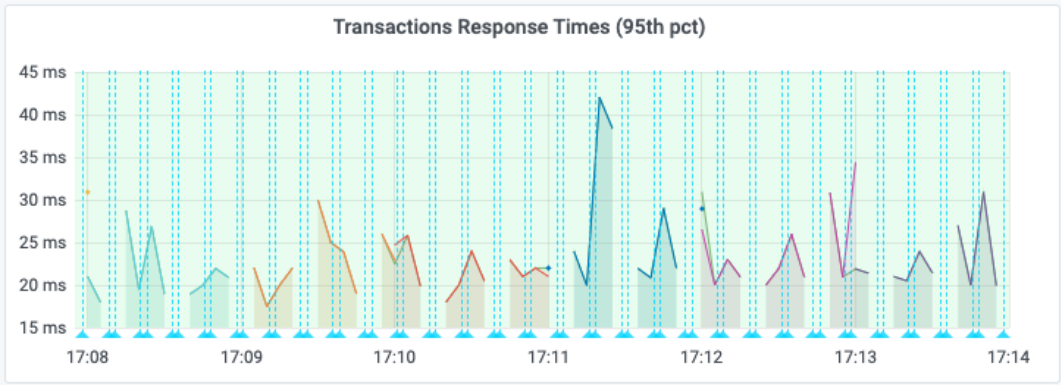

FIGURE 6.13: MySQL CREATE test, response time metric

Again, MySQL has the worst results here. Its average response time is three times slower than in PostgreSQL. Although, the only good side is that MySQL showed roughly stable performance, the response time varies around 20ms, excluding one outlier between 17:11 and 17:12 of 42ms.

	min ^	max	avg
all	17.50 ms	42.00 ms	23.15 ms
request 14:09 16-05	17.50 ms	29.95 ms	22.82 ms
request 14:10 16-05	18.00 ms	25.85 ms	21.81 ms
request 14:08 16-05	18.95 ms	21.95 ms	20.44 ms
request 14:11 16-05	19.95 ms	42.00 ms	26.90 ms
request 14:13 16-05	19.95 ms	30.90 ms	22.80 ms
request 14:12 16-05	20.00 ms	34.40 ms	24.14 ms
request 14:14 16-05	20.97 ms	22.00 ms	21.49 ms

	min ^	max	avg
all	5.00 ms	8.95 ms	6.85 ms
request 15:24 16-05	5.00 ms	8.95 ms	6.84 ms
request 15:26 16-05	5.00 ms	8.95 ms	6.66 ms
request 15:27 16-05	5.00 ms	7.90 ms	6.63 ms
request 15:28 16-05	5.00 ms	9.00 ms	7.07 ms
request 15:25 16-05	5.97 ms	10.90 ms	7.32 ms
request 15:23 16-05	6.00 ms	8.00 ms	7.09 ms
request 15:29 16-05	6.00 ms	8.00 ms	7.38 ms

FIGURE 6.14: MySQL and PostgreSQL CREATE test results side by side

Chapter 7

Conclusions

This thesis has studied different types of databases and conducted a review of current trends in DBMS. We have build and deployed a testing application that can interactively and dynamically display the results of tests. We researched the selected databases in terms of performance during simple CRUD tests. In the end, we did some conclusions and made assumptions based on results.

7.1 Discussion

Results in the previous chapter proved the trend, researched in [Chapter 4]. MySQL is losing its positions rapidly and inevitably. MongoDB showed good performance, but one has to remember that NoSQL databases are most effective for Big Data and when distributed on clusters. For this reason, it is not a surprise that PostgreSQL is faster in some cases.

We have not found a reason for such a poor performance of MySQL. The logic tells that PostgreSQL should be slower since it is more enhanced with features, and in MySQL, some of them are omitted in favor of performance.

We have also reviewed some existing benchmarks and saw that our results are similar. A benchmark from the creators of ArangoDB [ArangoDB, 2018] compared PostgreSQL and MongoDB (and some others, but they do not make interest here). We checked their tests since this benchmark is open-source; there were no faked results there. Their results show that MongoDB is slower than PostgreSQL on single read and write [see ArangoDB, 2018, Fig.Overall results].

Another benchmark from an independent developer [Petr Jahoda, 2021] compared MySQL, PostgreSQL, and some other RDBMS. His results show that PostgreSQL is much faster than all of them, not only from MySQL. He concludes that the "Postgres engine family is about twice as fast as MySQL engine family, exceptMariaDB".

There also is a benchmark [Anastasia Raspopina and Sveta Smirnova, 2017] in which MySQL performed better than PostgreSQL. Nevertheless, the tests performed there differ too much from ours. They did a multi-threaded test with millions of queries, and they also tuned both databases, so it does not make sense to compare our benchmarks. However, it may mean that MySQL is better on heavy load, but we can not state it.

To wrap it all up, MySQL is an outdated solution and should be replaced by new, better solutions, such as PostgreSQL. MongoDB showed good performance, as was expected. PostgreSQL surprised with its excellent performance and that it could even be an equal adversary to a NoSQL database.

And, the final note is that one has to remember that this performance test was made without any tuning. MySQL can indeed be configured to match PostgreSQL

and MongoDB results, but the two winners of this benchmark could also be tuned to show better results based on an application's needs.

7.2 Deviations from initial plan

The first deviation was encountered during the setup of virtual machines. The initial plan was to use AWS for deployment, but we refused to it since there was no student subscription available, so we decided to take Microsoft Azure. In the Microsoft Azure Student Subscription, as it turned out, there are available only six CPUs. Initially, we planned to have each database deployed on a separate machine, but we kept all three databases on one instance after this restriction. Also, we had to choose not that powerful virtual machines, that for sure worsened our results.

Another restriction was on databases that took part in the research. In the beginning of the project, we thought to take one database of each type. However, later, we dropped graph and key-value type because that would not be relevant to the type of test data. We also had to drop Cassandra, that was selected as the most famous representative of the wide-column type. It was done because the Java driver for it was supplied only for the older version of Cassandra and JMeter. What is more, it had very limited functionality, which would not be enough for our tests.

7.3 Future work

There is a lot of room for future improvement. The best improvement would be the measurement of other metrics, such as stability, reliability, scalability, and resource usage of databases.

It is also a point of interest how the system would behave on better/worse virtual machines, would the throughput be bigger and the response time shorter or no.

The third improvement is that it would be great if there was a method to run JMeter tests directly from Grafana, rather than running JMeter CLI. Unfortunately, according to Grafana documentation, it is not possible yet.

Appendix A

Implementation

A.1 Code

The code of the implementation of this benchmark can be found here: `https://github.com/lazyTurtle21/databases-benchmark`.

A.2 Demo

Visualisation of results is available on `http://13.82.195.63:3003/`. Use this credentials to login: username: user, password: password. Then, select dashboard named **Apache JMeter Dashboard using Core InfluxdbBackendListenerClient**. Once in dashboard, select time range when the experiments were conducted (the date is 2021-05-16, time can be found on screens in chapter 6).

Bibliography

Anastasia Raspopina and Sveta Smirnova (2017). *PostgreSQL and MySQL: Millions of Queries per Second*. URL: https://www.percona.com/blog/2017/01/06/millions-queries-per-second-postgresql-and-mysql-peaceful-battle-at-modern-demanding-workloads/ (visited on 05/16/2021).

ArangoDB (2018). *NoSQL Performance Benchmark 2018 – MongoDB, PostgreSQL, OrientDB, Neo4j and ArangoDB*. URL: https://www.arangodb.com/2018/02/nosql-performance-benchmark-2018-mongodb-postgresql-orientdb-neo4j-arangodb/ (visited on 05/16/2021).

AWS (2021a). *What is MySQL?* URL: https://aws.amazon.com/ru/rds/mysql/what-is-mysq (visited on 05/12/2021).

— (2021b). *What is PostgreSQL?* URL: https://aws.amazon.com/ru/rds/postgresql/what-is-postgresql/ (visited on 05/12/2021).

Codd, Edgar (1970). "A Relational Model of Data for Large Shared Data Banks ". In: *Communications of the ACM* 13.6, pp. 377–387. URL: https://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf.

Codd, Edgar and Raymond F. Boyce (1974). "Recent Investigations into Relational Data Base". In:

DB-engines (2021). *DB-engines Ranking*. URL: https://db-engines.com/en/ranking (visited on 05/10/2021).

Hadjigeorgiou, Christoforos (2013). "RDBMS vs NoSQL: Performance and Scaling Comparison". In: DOI: https://static.epcc.ed.ac.uk/dissertations/hpc-msc/2012-2013/RDBMS\%20vs\%20NoSQL\%20-\%20Performance\%20and\%20Scaling\%20Comparison.pdf.

Kleppmann, Martin (2017). *Designing Data-Intensive Applications*. first. O'REILLY.

Kolonko, Kamil (2018). "Performance comparison of the most popular relational and non-relational database management systems". In: *Open Journal of Databases (OJDB)*. DOI: https://www.diva-portal.org/smash/get/diva2:1199667/FULLTEXT02.pdf.

Li, Yishan and Sathiamoorthy Manoharan (2013). "A performance comparison of SQL and NoSQL databases". In: *EEE Pacific RIM Conference on Communications, Computers, and Signal Processing - Proceedings*, pp. 15–19. DOI: 10.1109/PACRIM.2013.6625441.

MongoDB (2021). *MongoDB: The most popular database for modern apps*. URL: https://www.mongodb.com/ (visited on 05/16/2021).

Neo4j. *Who uses Neo4j?* URL: https://neo4j.com/who-uses-neo4j/ (visited on 05/10/2021).

Nepaliya, Prateek and Prateek Gupta (2015). "Performance Analysis of NoSQL Databases". In: *International Journal of Computer Applications* 127.12. DOI: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.735.9010&rep=rep1&type=pdf.

Petr Jahoda (2021). *Benchmark databases in Docker: MySQL, PostgreSQL, SQL Server*. URL: https://itnext.io/benchmark-databases-in-docker-mysql-postgresql-sql-server-7b129368eed7/ (visited on 05/16/2021).

PostgreSQL (2021). *PostgreSQL: The World's Most Advanced Open Source Relational Database*. URL: https://www.postgresql.org/ (visited on 05/12/2021).

Sasaki, Bryce Merkl (2018). *Graph Databases for Beginners: The Basics of Data Modeling*. URL: https://neo4j.com/blog/data-modeling-basics/ (visited on 05/10/2021).

Stack Overflow (2020). *2020 Developer Survey*. URL: https://insights.stackoverflow.com/survey/2020 (visited on 05/11/2021).

V. Abramova, J. Bernardino and P. Furtado (2015). "Which NoSQL Database? A Performance Overview". In: *Open Journal of Databases (OJDB)* 1.2. DOI: https://dnb.info/1132360862/34.