Abstract

This paper delivers a comparative analysis of the performance of three of the NoSQL technologies in Web applications. These technologies are graph stores, key-value stores, and document stores. The study aims to assist developers and organizations in picking the suitable NoSQL solution for their application. For this purpose, three identical e-book applications were developed. Each of these is connected to a database from the selected technologies to examine how they perform compared to each other against various performance measures.

Keywords: Databases, NoSQL, web application, MongoDB, Neo4j, Redis, Node.js, E-books
Preface

I wish to thank my supervisors for their outstanding advice and help during this process. I would also like to thank all of those professors whom I would not have been able to reach the needed knowledge to achieve this study without whose assistance.

I would also like to thank my other fellow students, for their excellent information sharing. It has always been useful to exchange information with you about my research. You kept me motivated whenever I lost interest. My parents deserve special thanks: as always, your wise advice and your kind words have served me well.
## Contents

### List of Figures

### List of Tables

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Problem formulation</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Motivation</td>
<td>2</td>
</tr>
<tr>
<td>1.3 Objectives</td>
<td>2</td>
</tr>
<tr>
<td>1.4 Related work</td>
<td>2</td>
</tr>
<tr>
<td>1.5 Target group</td>
<td>3</td>
</tr>
<tr>
<td>1.6 Outline</td>
<td>3</td>
</tr>
<tr>
<td>2 Background</td>
<td>4</td>
</tr>
<tr>
<td>2.1 NoSQL</td>
<td>4</td>
</tr>
<tr>
<td>2.1.1 Key-value stores</td>
<td>6</td>
</tr>
<tr>
<td>2.1.2 Document stores</td>
<td>7</td>
</tr>
<tr>
<td>2.1.3 Graph stores</td>
<td>8</td>
</tr>
<tr>
<td>2.2 Test Application: E-Books</td>
<td>9</td>
</tr>
<tr>
<td>3 Method</td>
<td>11</td>
</tr>
<tr>
<td>3.1 Reliability and Validity</td>
<td>12</td>
</tr>
<tr>
<td>3.2 Method discussion</td>
<td>12</td>
</tr>
<tr>
<td>4 Implementation</td>
<td>13</td>
</tr>
<tr>
<td>4.1 Test Applications Implementation</td>
<td>13</td>
</tr>
<tr>
<td>4.2 Redis Implementation</td>
<td>16</td>
</tr>
<tr>
<td>4.3 Neo4j Implementation</td>
<td>17</td>
</tr>
<tr>
<td>4.4 MongoDB Implementation</td>
<td>18</td>
</tr>
<tr>
<td>4.5 Validating the implementation</td>
<td>19</td>
</tr>
<tr>
<td>5 Results</td>
<td>20</td>
</tr>
<tr>
<td>5.1 Insert operation results</td>
<td>20</td>
</tr>
<tr>
<td>5.2 Update operation results</td>
<td>21</td>
</tr>
<tr>
<td>5.3 Select operation results</td>
<td>22</td>
</tr>
<tr>
<td>5.4 Delete operation results</td>
<td>23</td>
</tr>
<tr>
<td>6 Analysis</td>
<td>25</td>
</tr>
<tr>
<td>6.1 INSERT Operation</td>
<td>25</td>
</tr>
<tr>
<td>6.2 UPDATE Operation</td>
<td>25</td>
</tr>
<tr>
<td>6.3 SELECT Operation</td>
<td>25</td>
</tr>
<tr>
<td>6.4 DELETE Operation</td>
<td>26</td>
</tr>
<tr>
<td>7 Discussion</td>
<td>27</td>
</tr>
</tbody>
</table>
List of Figures

2.1 Visualization of the CAP theorem ........................................ 4
2.2 Key-Value store ................................................................ 6
2.3 Document Store NoSQL Database ....................................... 7
2.4 Illustration of Graph store NoSQL database ....................... 8
3.5 Illustration of the experiment’s steps ................................. 12
4.6 Component Diagram for Node/Express Web applications ....... 14
4.7 Structure of Routes that handles APIs communications for all applications 14
4.8 Home Page for All Applications ......................................... 15
4.9 Implementation of the insert method in Redis .................... 16
4.10 Illustration of book document stored in MongoDB ............... 18
4.11 Inserting one book of type EPub in all databases ................ 19
5.12 Response time for inserting 10 books ............................... 20
5.13 Response time for inserting 500 books ............................. 21
5.14 Response time for updating 10 books ............................... 21
5.15 Response time for updating 500 books ............................. 22
5.16 Response time for selecting 10 books ............................... 23
5.17 Response time for selecting 500 books ............................. 23
5.18 Response time for deleting 10 books ............................... 24
5.19 Response time for deleting 500 books ............................. 24

List of Tables

4.1 List of the hardware components used in this experiment ....... 13
4.2 List of the software used in the implementation .................... 13

Listings

1 Method used to measure time for operation ......................... 15
2 Retrieving a number of HTML books from the Neo4j database .. 17
3 Retrieving data in MongoDB ............................................. 18
1 Introduction

For a long time, relational databases have been dominating the market when it comes to data storage and management. However, the growth of the amount of data that is needed to be stored and managed in these databases increased the need for scalability and performance.

Therefore, NoSQL or "Not Only SQL" databases have been introduced and emerged as a new way to deal with data storage. It provided more efficient solutions for numerous use cases. Big Data is the term used for a significantly large amount of structured or unstructured data that cannot, in some cases, be handled with traditional databases [1].

The term NoSQL was used for the first time in 1988 by Carlo Strozzi when he presented his relational database that does not implement the SQL interface [2]. Then, the term started referring to databases that do not use "RDBMS" meaning they do not use tables or fixed schema for storing and managing the data. That was the most significant shift from the relational databases that used to define the schema at the beginning of the process which made it harder to maintain when data needed to evolve. Moreover, Web applications had a big impact on motivating the argument of using the non-relational databases due to their need to support millions of users simultaneously and handle large amounts of data traffic especially after the requirement of Web 2.0 application [3] [4].

Relational databases and their proprieties caused some limitations in handling vast amounts of data. These limitations arose after some big companies like Facebook and Google demanded solutions for their services that support a massive number of users. These limitations have resulted in faster moving toward non-relational databases[4].

1.1 Problem formulation

NoSQL is classified into four main technologies, and each of them is designed to meet some use cases scenarios and offer solutions for them [3]. There are hundreds of databases that belong to these categories depending on their data model storage.

Henceforth, several pieces of research have been conducted to compare these technologies with relational databases, bearing in mind the performance in different situations. Despite the importance of these researches, a comparative performance study of these categories in terms of the Web application is required to help programmers and organizations in choosing the right solution for their application.

Consequently, to deliver high-quality service, the communications between the databases and the application are always a crucial matter to consider. These communications are mainly done by four operations which are INSERT, UPDATE, SELECT, and DELETE. Applications generally unevenly use these operations, whereas an application may need to ensure the high performance of one or two of them over the other. Therefore, this experiment evaluates the performance of the chosen databases by calculating the time each of these operations takes to complete the task.
1.2 Motivation
Web and mobile applications provide services for millions of users simultaneously. Several applications tend to move to non-relational databases to handle this enormous amount of data.

There are nowadays lots of databases that use NoSQL criteria which were created for different purposes. Therefore, the NoSQL can be broadly categorized into four groups depending on their data model. These categories are Key-value stores, Document stores, Graph stores, and Column-wide stores.

These four ways of storing data can be used in different situations according to the application they are used for and the various criteria. Consequently, finding the appropriate, most suitable, and most efficient NoSQL technology among these four available ones, is always a challenge for developers and architects.

Finally, choosing the right database for the application is important not only for the performance requirements but it helps in other quality attributes as well, such as the software architecture, where it may assist developers and architects in making the most beneficial decisions.

1.3 Objectives
The main goal of this work is to compare the performance of different technologies of NoSQL in order to observe their behaviour in different situations and conditions. Additionally, the developed applications in the experiment must be reusable to facilitate future work on top of this one.

- To find which NoSQL database should be chosen for a certain type of Web-based application.
- To determine how the data type influences the performance of the selected databases.
- To find how the amount of data influences the performance of the selected databases.
- To find how the various NoSQL databases perform compared to each other against various operations.

1.4 Related work
Konrad Fraczek and Malgorzata Wojcik (2017) performed a comparative analysis of non-relational and relational databases in the context of performance in web-applications [5]. They implemented a social media web application that supports one relational database and two non-relational databases. They used PostgreSQL as relational and MongoDB and Cassandra as non-relational. They compared the performance of these databases on data reading and writing. However, they conclude that MongoDB was fastest when reading data while PostgreSQL is faster when it comes to writing data.

In addition, C. Gyorod, R. Gyorod and R. Sotoc (2015) conducted another study to compare the relational and non-relation database model for Web-based applications [4]. In their experiment, MSSQL and MongoDB were chosen. They executed several operations on both databases and compared the experiment’s results. However, they concluded
that MongoDB performs better in terms of execution time when it comes to INSERT, UPDATE and DELETE while MSSQL gave better results in SELECT operation. Additionally, their results showed that the difference gets noticed when the number of executions is bigger.

Correspondingly, Enqing Tang and Yushan Fan (2016) did a study to compare five different NoSQL technologies and evaluate their performance by using YCSB\(^1\) [6]. YCSB is a framework that applies different workloads on different databases to evaluate their performance. They first loaded 100,000 records in the database and then executed different workloads to compare the performance by checking the time of loading and executing the data. They concluded that Redis as a database is the best solution for the experiment, while the second best solution was document databases and then comes column-family databases. Their study showed that using master-master architecture is better than master-slave architecture in case of rapidly growing data that need to be stored.

This study, unlike the above-mentioned studies, investigates the performance of the NoSQL technologies and the how the amount of data and its type can affect performance in different situations.

1.5 Target group

The target group of this project is software development bachelor students, who have some interest in developing Web applications and the interaction with Databases. In addition, developers who want to gain a better understanding of NoSQL technologies. Basic understanding of Node.Js and database queries is required to repeat the experiment and understand the concept of the paper.

1.6 Outline

The following section in this report presents the background knowledge for NoSQL databases and the technologies studied in this paper, furthermore, a brief description of e-books which will is used in the test applications. Sections 3 is concerned with the methodology applied in this research. It describes in detail the experimental steps to conduct results that answer the problem followed by a discussion of the method and reliability and validity of the method. Section 4 concerns about the implementation of the application used in performing the experiment. In Section 5 all results conducted from the experiment are illustrated and described. Further details and analysis of the results are provided in section 6. Section 7 an evaluation of the analysis is provided in a broader scope of the field. Finally, in section 8 a conclusion of the research is provided with mentioning the results of the paper and reflecting on the progress of the project.

---

\(^1\) Yahoo Cloud Serving Benchmark
2 Background

This section gives the necessary information for the reader about the project’s most important aspects in a theoretical way.

2.1 NoSQL

NoSQL (Not Only SQL) is a data storage solution that is different from the relational solution that uses SQL. That is why it is called non-relational databases. Eventually, NoSQL is not to replace the relational database, but to provide another way of solving the problem when it comes to data storage.

NoSQL DBMS\(^2\) store large amounts of data. They are scalable, accessible, fast and tackle many problems with different types of data. Unlike SQL, NoSQL has no definite criteria for how the database should be structured. However, there are four main categories or ways of structuring data that are discussed later in this paper. NoSQL databases do not completely support ACID\(^3\) transactions, which are supported in SQL, although, they are determined by the CAP theorem to justify the trade-off for not supporting ACID. CAP stands for consistency, availability and partition tolerance which are the guarantees that no distributed system can achieve simultaneously [7][8].

- Consistency: data is available at any time for all parts of the system.
- Availability: if part of a system fall, the remaining will not be affected.
- Partition tolerance: the system continues to work even if it is split into a disconnected sub-system.

![Figure 2.1: Visualization of the CAP theorem](image)

Relational databases guarantee consistency and availability in the relational databases by supporting ACID, especially when the data is stored on the same server. Whereas,

---

\(^2\) Database management system

\(^3\) Atomicity Consistency Isolation Durability
when the data is distributed on multiple servers or increased consistently, the ACID properties are not the best solution. NoSQL is built on clusters which makes the databases available on multiple servers. That means when data is modified, all machines that use these data gets notified, and by that consistency is guaranteed. To achieve that, NoSQL follows BASE principles: [7]

- Basically Available: indicates that the system does not necessarily guarantee availability as part of it might go down if a node fails.

- Soft State: indicates that the system does not promise consistency as the state of the system may change because of the Eventual Consistency property.

- Eventual Consistency: indicates that the system ensures consistency at some point even if the data is not consistent

Finally, and as mentioned above, NoSQL does not replace RDBMS, and the comparison between them is not straightforward due to the different implementation and use cases that they cover. One of the main problems that are always faced while using SQL is a schema where different attributes can be added. That is in many cases, a requirement that SQL had a hard time to solve. Therefore, comparing them depends always on the use case or the type of data and the way of storage that fits best the programs. NoSQL main categories are key-value store, document store, column-based store and graph store.
2.1.1 Key-value stores

A key-value store is like word references where a key presents data. This key is what we use to retrieve our stored data, and that is why this key should be unique and different from the other [9]. That is considered as the simplest data structure where no schemas needed and saving new values do not cause any conflicts. The values can be grouped into collections to provide the desired data model.

This technology could be valuable for straightforward tasks, for instance, search systems where data can is stored in nodes [10]. Moreover, the Key-Value database is considered the most appropriate for software that does not need a lot of data [11].

Three primary operations can be applied to the key: delete, change, and insert, where the delete operation deletes the key and its value, the change operation modifies the entry associated with that key, and, finally, the insert operation adds a new key with a value. Consequently, the data is only accessible if the key is known.

Noteworthy, the key can be formed merely as hash or file-name, or it can be system-generated or defined. However, the data that is associated with the key is represented by the value and encoded in a byte array.

![Figure 2.2: Key-Value store](image)

- Redis

For the implementation and the experiment in this paper, Redis is going to be used, and the reason behind this decision is that Redis is one of the most popular key-value databases. Redis is an open-source, in-memory database used mostly as a cache and a message broker [12][13].

It runs in the RAM storage, which makes this database consistent and eventually provides more availability for the system [3]. To that extent, Redis is commonly used for storing cookies and real-time applications. Redis supports various data structure, namely strings, hashes, lists, sets, and sorted sets, hence, retrieving the data differ depending on these data structure and each of them requires different commands [13].

Finally, due to the lack of security and the high probability of losing data, Redis is not recommended to be used for transaction application as it relies on the network connection. If a connection error occurs, a loss of data most likely to happen.
2.1.2 Document stores

As to what is stated in its name, Documents are the fundamental part of document databases where all data is stored in different types of documents such as JSON, BSON\(^4\), and XML\(^5\). Multiple documents can be saved and grouped in collections and similarly to Key-Value, each document has a key to be traced with, and all queries are done by using this key.

However, in Document store, all data in a document can be modified, unlike Key-Value. Moreover, document stores provide more complex lookups comparing to key-value where multiple attributes can be used at once for the queries. The properties mentioned above make this way of data store one of the best solutions for web application and as they provide embedded and self-describing documents, which means that no fixed schema is followed for the stored data.

There are many document databases, but the most prominent ones are CouchDB, RavenDB, OrientDB, and MongoDB. This paper investigates and discusses MongoDB as it is the most frequently used database among the ones mentioned above [14].

![Figure 2.3: Document Store NoSQL Database](image)

- **MongoDB**

MongoDB is one of the most popular document-oriented databases. It is an open-source written in C++ and Devel. Data is stored as JSON and schema-less, where the only needed information is the ID [15]. It was described by Han, H, Le, and Du as a database between relational and non-relational, which implies that it combines features from both [16].

The reasons behind its popularity are the features that it provides, such as, a powerful query language where almost all functions in relational databases are allowed. Moreover, the high-speed access to data which is 10 times faster than MySQL. Lastly, MongoDB supports more complex data types such as BSON which is not supported in many other document databases [16].

\(^4\)Binary JSON  
\(^5\)Extensible Mark-up Language
MongoDB’s document size does not exceed 16 MB, and equivalent to a raw in relational databases. Documents can be grouped into one collection even if they do not share the same data structure or the content [16].

MongoDB provides different frameworks such as Aggregation and MapReduce framework, which makes interacting with documents and modifying its content easier and simpler. Moreover, for faster queries, indexes are defined and could be used in multiple fields at the same time [15].

### 2.1.3 Graph stores

A graph database is designed to store data where the relationship is the main thing to consider. Data is stored in interconnected nodes with attributes to describe each node. These nodes form a graph data model that illustrates the database [17]. Each node in most cases is tagged with a label to represent its role. Nodes are connected and related to each other by relationships (edges), which can have properties that indicate the type of relationship among nodes [17].

Accordingly, graph databases are considered suitable for applications that rely more on the relation between the data instead of the data itself. Due to the multiple ways provided for handling the interpretation of the data as well, these applications are mostly related to traversing social networks.

Graph databases support replication, which can be useful in case of having data distributed on multiple servers, and these data represent multiple connected nodes that are included in an operation.

There are many accessible graph databases, such as Neo4j, OrientDB, FlockDB, and InfiniteGraph. A study in 2015 [18] has shown that the use of them has increased by 500% since 2013, and they will be used by 25% of the venture in the future.

![Illustration of Graph store NoSQL database](image)

Figure 2.4: Illustration of Graph store NoSQL database
• Neo4J

Neo4j, as described above, is one of the most used graph store database application, which was launched in 2007 by Neo technology. It is introduced in almost all programming languages with native APIs.

As in almost all graph databases, Neo4j store data into nodes as properties, and these nodes are interconnected by edges that indicate the relationship. Relationships between two nodes can be in both ways, which shows that there is a different type of relationship depending on the start node. That helps in traversing the tree and ensures that the nodes that are getting traversed are the ones in the right direction.

Its query language is called Cypher Query, which is similar to the relational databases query, especially the keywords used in the commands. In Cypher, complicated statements can be formed easily, and it focuses more on what data to retrieve instead of how to retrieve it, and thus, it is considered as declarative language. Like almost every query language, Cypher query uses clauses and queries start by retrieving many nodes and then giving back a small collection of nodes [19] [20].

Moreover, ACID is transaction compliance used in this database, which means that either all operations fail or all succeed. Neo4j is a schema-optional database where schemas are not necessarily used but recommended in some cases, to improve the performance of indexing [19].

2.2 Test Application: E-Books

A technology revolution is surrounding us in many parts of our lives nowadays. Education, as one of the essential parts, is being affected by this revolution and the methods are changing rapidly to fit these new technologies. To that extent, electronic devices, i.e., tablets and computers, are being used increasingly and, in some countries, they are now considered as a fundamental element and not an accessory anymore. These devices are somehow replacing books or traditional teaching materials such as papers and written exams. Teachers are interacting with their students through these devices, and even exams and quizzes are mostly being done online.

Teaching materials, mainly books, are in physical format except for the new ones that were created in both physical and digital. Therefore, digital format is required of all these books so students, researches, and everyone interested in the book can reach it online. Project Gutenberg is one of the first projects in converting books that their copyright has been expired into digital format [18]. The project is non-profitable, and its only goal is to make it easier for users to access as many books as possible. The website now provides access to 58000 free e-books through the browser [18].

E-books like the regular paper books provide all standard features such as page bookmarks, notes and text highlights. However, on top of that, some other features like built-in dictionaries, adjusting the font size, search by words and sentences and most importantly the ability to read in the dark with the help of the device light. These mentioned features have led to making it easier for an e-book to replace the regular ones.

Since e-books are accessible by different devices, they come in different formats with considering that these devices do not necessarily support all formats. Among the most common formats are Epub, Mobi, Pdf, and Azw.
• Epub: (E-Publication) is the most adopted e-book format, which was developed by the International Digital Publishing Forum. Almost all devices widely support epub. It supports images, graphics, and videos.

• Mobi: This format was originally created to serve mobile devices. Unlike Epub, it is not publicly accessible and does not support audio and video.

• AZW: Amazon e-book format is based on Mobi but only directed to Amazon’s Kindle devices, which are e-readers devices designed by Amazon to read and browse books. Azw is not as popular as Epub and Mobi.

• PDF: (Portable Document Format) is the most common format as it is widely adopted in the Web. The PDF format is created by Adobe, and it is designed for easier sharing of the documents between devices. PDF is more limited than the other formats, and most importantly, it does not support re-flowing.

Some other formats are less used than the ones discussed above, i.e., LIT, RTF, DJVU and LRS, LRF.

In order to access these e-books, a way of storage is needed. Databases as a way to store these e-books can be a suitable solution. Especially the ones that are good at handling large amounts of data in a structured way.

Beyond choosing the storage way, the tasks that would be more often used in the application should be considered. For instance, tasks, like saving e-books and retrieving information about them, can be done manually or by using different scripts. That means the response time these tasks take to be completed should be as low as possible to provide a better service for the users. That is an essential aspect of having a user-friendly website.

It is always challenging to decide which database should be chosen as they differ a lot in the way they handle the various operation and the storing of the data. Generally, these databases are either relational or non-relational. Wei Ping, Ming Xin, and Huan (2011) showed that MongoDB as a sample of the non-relational databases is faster at storing data than MySQL, which is a relational database.
3 Method

This study uses a controlled experiment to conduct the results, and it is inspired by a similar method was performed by Wei Ping, Ming Xin, and Huan (2011) when they measured the response time for entering the database [21]. However, this experiment differs in using NodeJs instead of what they used, which is Java. The reason behind choosing Node.js is that its use overgrows when it comes to back-end development nowadays.

The experiment consists of three applications that are implemented by HTML, JS, Node.js and CSS. Each of these three applications is connected to one of the databases that are targeted in this study. Each application is designed to perform four operations, which are, INSERT, SELECT, UPDATE, and DELETE which are the fundamental operations to be done on a database. Time measurement for the response of each operation is calculated to compare the actual performance of each database with each operation. This experiment does not involve relations between elements but focuses primarily on the response time for interacting with the database.

Two different data sizes are used in the experiment where each of the four operations will be executed on. For more accurate results, each operation is executed five times, and the average time is calculated for further comparison.

The method used in this study is divided into the following five steps:

- Scope
- Planning
- Executing
- Analysis
- Results and Discussion

The scope in this paper is presented in the problem formulation and objectives where it is about figuring out which of the tested database is the most efficient for Web-applications in term of performance. Planning follows as a second step and presented in this section. It indicates the way this paper plan to answer the questions conducted from the scope. Executing is carried out in section 4 and 5 where all details used in the experiment are discussed and evaluated before the execution of the experiment. Then followed by the results section that shows what results was obtained by executing the operations required for answering the research questions. Thereafter, the fourth step which is the analysis is presented in section 6. The analysis is performed on the actual results highlighted in the experiment. Lastly, Discussing the findings and validate how they answered the problem formulated in this paper. A graphical Representation of the experiment’s steps that used in this paper are showed below in Figure 3.5.
3.1 Reliability and Validity

Validating the method is done by first performing a pilot study. Pilot study ensures that it is possible to get the results that answer the scope questions by using this method. Performing a pilot study is done by using a smaller scale than the main experiment, and that helps validate the quality and efficiency of the experiment. Later, we select and validate the information and the inputs that are used in this experiment. The selected inputs are create, retrieve, update, and delete. Whereas, the selected information to be measured are the book’s formats which are mentioned and described in 2.2. Once, everything is selected and validated, the execution of the experiment is performed to get the results. These results are analyzed and ensured by repeating the experiment multiple times.

3.2 Method discussion

The applied method in this paper was described and proved efficiency by Claes Wholin and a group of researchers in their book "Experimentation in Software Engineering" [22]. The paper answers questions about efficiency and reliability and henceforth, the targeted databases should be executed and examined to get reliable results. An alternative method, according to the book, is to perform a literature case study relying on things that may have already occurred or will somehow occur. However, and in our case, this method cannot be considered as an alternative as the reliability is ensured by implementing these databases and analyzing the given results.
4 Implementation

The experiment aims to examine which NoSQL database handles different file formats in the lowest possible response time. All applications are as identical as possible so that a fair comparison can be effectively made without letting the databases to gain advantages or disadvantages in case of different implementations. Tables 4.1 and 4.2 indicate the hardware and software used for the pilot study and the development of the applications. That makes it possible to repeat the experiment with the same hardware and software.

<table>
<thead>
<tr>
<th>Component</th>
<th>Company</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>Intel Core i7</td>
<td>2.9 GHz</td>
</tr>
<tr>
<td>Memory</td>
<td>DDR3</td>
<td>8 GB 1600 MHz</td>
</tr>
<tr>
<td>Graphics</td>
<td>Intel HD Graphics 4000</td>
<td>1536 MB</td>
</tr>
<tr>
<td>Hard drive</td>
<td>Macintosh HD</td>
<td>750 GB</td>
</tr>
</tbody>
</table>

Table 4.1: List of the hardware components used in this experiment

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mac OS High Sierra</td>
<td>10.13.6</td>
</tr>
<tr>
<td>Visual Studio Code</td>
<td>1.27.2</td>
</tr>
<tr>
<td>MongoDB</td>
<td>3.0.4</td>
</tr>
<tr>
<td>Redis</td>
<td>2.8.0</td>
</tr>
<tr>
<td>Neo4J</td>
<td>1.7.4</td>
</tr>
<tr>
<td>Node.js</td>
<td>10.13.0</td>
</tr>
</tbody>
</table>

Table 4.2: List of the software used in the implementation.

4.1 Test Applications Implementation

The three applications are identical as mentioned above. In each application, the user can insert, delete, update, or select a book from and to the database. Moreover, and through the three drop-down buttons, the user can choose the number of books, the book format and the operation to be executed. We can see in Figure 4.8 that the home page is simply implemented for easier execution of the experiment. The applications differ from each other by the way they interact with the database as it may vary depending on the type of the database connected to the application. The database operations performed in the applications are similar, while the time each operation takes to finish is calculated for the comparison. In other words, the less time an operation takes, the better it performs. In this work, the selected databases are Redis, MongoDB, and Neo4j since these databases are the most widely used among their categories [14].

Applications are developed in NodeJS, which is a server-side JavaScript platform for Web applications. Express is used as a framework because of the benefits it provides in creating Web applications such as simple APIs for the application and interfaces to easily create and control the applications. One of the advantages Express offers for these applications is the Router middleware. That helps developers to control routes that are
associated with the HTTP methods with easy structure and architecture. Figure 4.6 illus-
trates how Node.js/Express applications handle the HTTP request and communicate with the database.

Figure 4.6: Component Diagram for Node/Express Web applications

Routes in the implemented applications are identically developed, as shown in Figure 4.7. These routes connect the request with the appropriate function in the controller, which communicates with models to manipulate the data in the database. In the controllers, each application is implemented in a different way depending on how the database communicates with the APIs.

Figure 4.7: Structure of Routes that handles APIs communications for all applications
On the other side, some libraries are used in implementing the application such as a template engine for simplicity, e.g., Handlebars, because it separates layout from content. For reading and accessing files on the system, the FS (file server) is used which is built in NodeJS. Lastly, Morgan library is used as an HTTP request logger. It helps track what is going on the server and log into the console. MongoDB provides by default an ID for each document. However, this is not the case in Redis and Neo4j, and, for that reason, the UUID library is used to generate an ID for each added book to the system.

The home page of all applications looks identical as in Figure 4.8 with three drop-down buttons to indicate what operation to perform, how many times, and for what format.

![Welcome to Neo4j](image)

**Figure 4.8: Home Page for All Applications.**

When the Run button is pressed, the operation selected is executed and the required time by the simulation is calculated and saved in text files for later comparison. The time measurement is done by a method which is called twice, once before the execution starts where it returns the current time, and once after the execution where it returns the time since the first recorded time the method was called, see the example codes below.

```javascript
function measureTime(hrstart)
{
    if(!hrstart){
        return process.hrtime()
    }
    var hrend = process.hrtime(hrstart)
    return Math.round((hrend[0]+1000) + (hrend[1]/1000000))
}

Listing 1: Method used to measure time for operation
```

This mechanism is provided in Node.js by default. It returns high-resolution real-time in nanoseconds [23]. However, in this method, time is converted into milliseconds for simplicity.
4.2 Redis Implementation

To link a Redis database to the application, the Redis module is called by using NPM\(^7\). This module works as a Redis client for Node.js, where it supports all commands and operation that Redis provide.

In the implemented application, all books are saved using string as a key. The string is formed by the book format followed by a dot and a randomly generated key. Similar to two other applications, four main methods implemented which handle inserting, updating, retrieving, and deleting data from the database.

![Figure 4.9: Implementation of the insert method in Redis](image)

Figure 4.9 illustrates how the insert method in Redis works. First, by the help of the parameters the method indicates the desired book format, then a for loop to insert the number of books the user requested. Finally, when all the books are inserted into the database, the time the operation took is measured and saved in a text file on the local machine. Noteworthy that `client.set()` is used here because the key that is used to store the data is String, otherwise another command could be used depending on the key, i.e., `client.hset()` for hash keys.

Similarly, the three other methods that deal with communications with APIs are implemented. However, different commands are used based on the functionality of the method i.e. `client.del()` to delete data from the database, and `client.get()` to retrieve data.

\(^7\)Node Package Manager
Updating data in Redis is done by first retrieving the necessary data by calling `client.get()` and then running `client.set()` which checks if the given key exists so it updates the value.

Finally, for retrieving all keys in the database the command `client.keys("*")` that iterates in the database and returns all keys.

### 4.3 Neo4j Implementation

Two requirements are needed to send a valid statement; the query string and the parameters that are necessary for the statement execution. Once the operation is done, the session needs to be closed by using the command `session.close()`

Typically, a connection with the database is established by executing the command `session.run()` and the statement needed to be executed is sent in it. For instance, to retrieve all books from the database, the statement line will be `{MATCH (n) RETURN n}`, while `n` is a label used for all books regardless of their format.

Creating a new node uses the statement `{CREATE (n) SET n = value}`. Create keyword is for creating an empty node, and `SET` is to give the node value. However, the value that is assigned to a node is specified in the second parameter in the command `session.run()`

Updating a node in Neo4j is so similar to creating, but instead of using the `CREATE` keyword that creates a new node, `MATCH` is used. Furthermore, to look up a node by one of its proprieties, `WHERE` keyword followed by the parameter that we need to find. For example, this statement `{MATCH (n) WHERE n.id = 25}` tries to find a book with an id 25.

Lastly, deleting in Neo4j is performed by using the keyword `DELETE` after matching and finding the node meant to be removed. The code snippet below is from the test application and illustrates how data is retrieved in render to the client application.

```javascript
// retrieving a number of HTML books from the database
session.run('MATCH (n:htmlBook) RETURN n LIMIT {number}', { number: parseInt(number) })
    .then(function (results) {
        results.records.forEach(function (record) {
            resultArray.push({
                name: record._fields[0].properties.name,
                id: record._fields[0].properties.id
            })
        })
        res.render('index', { title: "Neo4j", count : resultArray.length , items: resultArray })
    })
Listing 2: Retrieving a number of HTML books from the Neo4j database
```

It is noteworthy that after `MATCH` the label is specified to `htmlbooks` while in the actual implementation the query string is decided by using a switch statement where it checks the format given by the user and edits the query depending on it. Moreover, the query here is parameterized where the number is assigned outside the query string.
4.4 MongoDB Implementation

Initially, MongoDB is supported natively in Node.js by the driver "mongodb" which provide a Mongo client to connect to the database and access it. A database’s URL is required to initiate the connection. This driver supports almost all MongoDB operations and covers all the desired test application functions.

The application connected to MongoDB is designed to have four collections that represent the four formats respectively in which the books are represented by documents in those collections. Each book has a unique ID provided by MongoDB as presented in Figure 4.10. Therefore, the UUID used to generate IDs in the other applications is not used here.

![Figure 4.10: Illustration of book document stored in MongoDB](image)

The query language of MongoDB is well structured, and, thus, it is an easy language to work with. In the test application, all operations start by connecting to the database and then get the collection that is requested by the user.

For instance, the query the retrieves certain amount of data looks as follows:

```javascript
var db = client.db(dbName)
var cursor = db.collection(format + "books").find()
    .limit(parseInt(number))
Listing 3: Retrieving data in MongoDB
```

The format here changes according to the user’s input in the HTTP’s request and the same applies for the number. In case of looking for a specific document, a parameter needs to be used for search are inserted in find method arguments. All other operations work in a similar manner but with different commands. For instance, insertion uses `insertOne` instead of `find` and the number of documents needs to be inserted is done by having these codes in a for loop stops as the number indicated by the user.
4.5 Validating the implementation

With the implemented applications, a pilot study was performed to validate the quality and efficiency of the applications. To that extent, one operation of each database is performed on a small number of books. The time each database takes to execute the operations is measured.

This is done to present whether the experiment is feasible or not. Executing the pilot study on one operation ensures that the application is correctly connected to the database in order to start the experiment.

To ensure the accuracy of the measurements, each operation will be carried out five times in order to calculate the average time. More importantly, this ensures that the applications are correctly working and the results are reasonable.

Figure 4.11: Inserting one book of type EPub in all databases

Figure 4.11 shows the time each database takes to add one book. It is clear that Redis took more time than Neo4j and MongoDB and that might differ in the actual experiment as this execution was done on only *epub* books which might be the worst case scenario for Redis. However, this execution validates that all functions are implemented successfully and the interaction with the database is carried out correctly so the environment is set up to execute the experiment.
5 Results

In this section, an overview of the results of the experiment is provided with graphical representation. All operations that were performed are briefly described with the conducted results.

All operations are performed on two amounts of data, 10 and 500 books, to observe how the increase of the data amount can affect the performance. In addition, each execution was done 5 times and the results shown below are the average result of the five executions.

5.1 Insert operation results

Insertion was the first test case for the experiment and the results shown in Figure 5.12 and Figure 5.13 were obtained after executing the operation in all databases. Operations have been carried on 10 and 500 books, in the four formats to observe how the response time is affected when data is greatly increased.

We can see that MongoDB showed the lowest response time in all formats in Figure 5.12 in which 10 books were inserted. While Redis was the worst option in all formats, especially when inserting an HTML book, as the time was exponentially increased. However, in Figure 5.13, where the experiment was done on 500 books, we see that the book format was a key factor that influences the response time. Redis still showed worse results but surprisingly not when the book format is Epub. Moreover, Neo4j gave a better result than MongoDB when data was increased.
Figure 5.13: Response time for inserting 500 books

5.2 Update operation results

Similarly, Figure 5.14 and Figure 5.15 shows the time each database took in updating books in all formats. The first execution was done on 10 books where all three databases performed equally good. However, when updating a book of type Mobi, Redis performed 20 times worse than other databases as shown if Figure 5.14.

Figure 5.14: Response time for updating 10 books
On the other hand, the execution of the *UPDATE* operation on 500 books, as illustrated in Figure 5.15, showed that Neo4j response time was so small, which was not the case for Redis. Redis and especially in *text* and *Mobi* book formats gave a very big response time.

![UPDATE 500 BOOKS](image)

**Figure 5.15:** Response time for updating 500 books

### 5.3 Select operation results

Figure 5.16 and Figure 5.17 show the time in milliseconds each of the databases took when executing the *SELECT* operation. From Figure 5.14 we notice that MongoDB is not as efficient as it is in other operations where we see the response time is high and the difference is big compared to the other databases. Conversely, Redis and Neo4j gave almost the same results and proved efficiency in this operation where the time was less than three milliseconds.

However, and by executing the same operation on 500 books, the bar graph in Figure 5.17 shows that Redis gave the lowest response time as it was in the last execution, while Neo4j’s response time was higher and especially when the format tested was *Epub.*
5.4 Delete operation results

The last test case of the experiment was the DELETE operation. When first was executed on 10 books, MongoDB and Redis performed faster and gave low response time. On the other hand, Neo4j took more time in this execution. Figure 5.18 contains the results of the execution.
Figure 5.18: Response time for deleting 10 books

Similar to the last execution, Figure 5.19 shows that Neo4j is the slowest when deleting data, followed by MongoDB and Redis.

Figure 5.19: Response time for deleting 500 books
6 Analysis

The problem to be solved by this study is which of the three NoSQL technologies will perform better for the four database operations INSERT, UPDATE, SELECT, and DELETE.

This section is divided into four parts, where every part analyzes the results obtained from an operation on the chosen databases.

6.1 INSERT Operation

When it comes to the INSERT operation, the results were different when the amount of data was increased, as illustrated in Figures 5.12 and 5.13. First, after executing on only ten books, MongoDB and in all formats gave the lowest response time while Redis with a big difference gave the highest response time. However, and against the expected results, Neo4j performed better when the data amount was increased, and Redis gave the highest response time.

The reason behind that is that Redis is an in-memory database as it writes first to the memory and later at some point it writes asynchronously to the disc. Thus, when the amount of data is increased, the execution time of the operation will get bigger as Redis will start writing to the disc. The file format has a significant influence on the performance of Neo4j as the results were greatly different from one to another. In the case of a large amount of data, the mobi format had an insignificant impact on the performance of Neo4j and MongoDB.

6.2 UPDATE Operation

Updating an existing file is somehow similar to inserting it, but the difference is that the file is already created which supposedly means that the database has less to do and the response time shall be shorter accordingly which is the case in most results was obtained from executing the UPDATE operation on the databases.

However, Figure 5.15 shows that Redis took a long time to update a file than what it took to insert when the file format was text or mobi. That can be due to the expensive look-up that Redis make when the data is not anymore saved on the memory but on the disc. In this case, Redis first search the memory database, and if the required file is not found, it will search the disk.

6.3 SELECT Operation

When it comes to the results of SELECT operation, which is shown in section 5.3, MongoDB took longer time than other databases with a notable difference. This could be due to the complexity of reading data from the database as of the non-structured nature of MongoDB. Redis, on the contrary, gave a short response time in both small and big amounts of data. That tells that retrieving the data in Redis did not affect the response time of updating, but the writing to the exists data which is explained in section 6.1.

Additionally, here we see that the book formats have a slight impact on the performance of the databases where all the results were almost the same.
Neo4j, on the other hand, showed that the amount of data affected the results wherein the case of adding ten books, the performance was superior, and, almost the best, out of them. However, when the execution was done on 500 books, Neo4j’s performance was profoundly affected, and in some of the file formats, it was close to being the highest response time. It could be that if we do a similar execution on more amount of data, the performance of Neo4j will be the worst compared to the others.

6.4 DELETE Operation

The results obtained from applying DELETE are showed in Figures 5.18, and 5.19 show that each database reacted differently and performed differently from the others in both executions. Neo4j gave the highest response time both in small and big data amounts. However, and by deleting 500, the performance varied depending on the file format. Deleting mobi and html took more time than the other two formats.

On the other hand, Neither the file format nor the increased amount of data affected the performance of Redis in both executions, and it always gave the lowest response time and the best efficiency compared to the other databases. Whereas increasing the amount of data increased the response time of MongoDB in spite of the file format used in the experiment.
7 Discussion

The procedure that is done to answer the problem uses various independent variables such as different operations and different file formats. The aim is to outline a better insight into how the studied NoSQL technologies perform in different situations. The results, as expected, do not show that any of these databases is more efficient for all cases in general. They provide a guide to which database to be used in a particular situation where the operation and the data type mostly used in the application are known.

In summary, from all experiments that were done, Redis shows better efficiency in reading data than writing data which might be affected by the fact the Redis is in-memory database and its performance depends on where the data is located.

On the other hand, MongoDB as a document database obtained an advantage in writing operations but not in reading. Furthermore, MongoDB shows that the performance gets worse when the data amount is increased, which is similar to what E. Tang and Y. Fan found in their research [6]. Another study done by V. Abramova, J. Bernardino, and P. Furtado (2014) has shown the same results when it comes to MongoDB [24]. Based on these results, we conclude that MongoDB’s performance is dependent on the amount of data rather than the data type. Moreover, the SELECT operation also showed the lowest response time compared to other operations.

Resulting from the executions performed on Neo4j, one can observe that it behaves in a similar way to MongoDB where its response time is lower when an operation is about writing data, and higher in searching for data. Additionally, the data type and amount affected Neo4j in some operation but not as strong as the other databases were affected, which might lead to favor it over the others.

In conclusion, the results show that these databases perform differently in a different situation. Thus each of them could benefit a certain application but not necessarily all applications depending on the variables used in the application. MongoDB, For example, is not the best solution if the application needs to download books regularly, whereas it might be the best solution if downloading books is not the primary function of the application. Neo4j could, on the other hand, be considered as the most appropriate solution for applications that do not rely on one operation over the others, due to its balanced obtained results.
8 Conclusion

This paper presents a comparative analysis of three NoSQL technologies, namely: document store, key-value store and graph store in the context of performance. To that extent, an experiment is conducted by implementing three identical applications connected respectively to the databases. The applications perform the basic four operations on the databases and compare their response time in different situations and scales to better analyze how these databases behave in different situations and conditions.

The paper presents in the results and the discussion parts that applying these operations on the databases did not favor any of them for all operations, yet the results show what databases are suitable for different situations. To choose the right solution that satisfies all the application’s requirements, the results presented in the paper shall be taken into consideration. Thus, Depending on what each application needs, we can choose the most suitable database.

The results obtained from the experiment are credible for the simulation environment and the various variables chosen for it. Although these results can be different depending on the simulation environment, nevertheless they are reliable and can be considered as a base for organizations and developers to select a suitable database for their particular situation.

The chosen variables for the implementation, which are the formats of books, are the only thing that would most likely affect the results if they are changed. For instance, using different book formats such as AWZ might increase the response time of a database and vice versa.

8.1 Future work

Deciding on which database to choose for a project is always challenging and performance is not the only quality attribute to be considered. This research could, however, be extended to include other attributes of quality, i.e., maintainability, robustness and reliability, which will undoubtedly benefit NoSQL.

Moreover, Big data Web application grows rapidly and becomes a significant factor in many industries. Therefore, a performance analysis of NoSQL technologies in BigData is required to help organizations select the right NoSQL solution.
References


