



**A COMPREHENSIVE COMPARISON OF NOSQL AND RELATIONAL  
DATABASE MANAGEMENT SYSTEMS**

**Ihsan Ahmed Taha**

**January 2017**

**A COMPREHENSIVE COMPARISON OF NOSQL AND RELATIONAL  
DATABASE MANAGEMENT SYSTEMS**

**A THESIS SUBMITTED TO  
THE GRADUATE SCHOOL OF NATURAL AND APPLIED  
SCIENCES OF  
ÇANKAYA UNIVERSITY**

**BY**


**IHSAN AHMED TAHA**

**IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE  
DEGREE OF  
MASTER OF SCIENCE  
IN  
THE DEPARTMENT OF  
MATHEMATICS AND COMPUTER SCIENCE  
INFORMATION TECHNOLOGY PROGRAM**


**Title of Thesis: A COMPREHENSIVE COMPARISON OF NOSQL AND RELATIONAL DATABASE MANAGEMENT SYSTEMS**

Submitted by **Ihsan Ahmed Taha**


Approval of the Graduate School of Natural and Applied Sciences, Çankaya University.

  
\_\_\_\_\_  
Prof. Dr. Halil Tanyer EYYUBOĞLU  
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

  
\_\_\_\_\_  
Assoc. Prof. Dr. Fahd JARAD  
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

  
\_\_\_\_\_  
Assist. Prof. Dr. Murat SARAN  
Supervisor

**Examination Date: 03.01.2017**

**Examining Committee Members**

Assoc. Prof. Dr. H. Hakan MARAŞ (Çankaya Univ.)

Assist. Prof. Dr. Murat SARAN (Çankaya Univ.)

Assist. Prof. Dr. İ. Tolga MEDENİ (Yildirim Beyazıt Univ.)

  
\_\_\_\_\_

  
\_\_\_\_\_

  
\_\_\_\_\_

## STATEMENT OF NON-PLAGIARISM

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname : Ilhsan Taha

Signature



Date

: 03/01/2017

## **ABSTRACT**

### **A COMPREHENSIVE COMPARISON OF NOSQL AND RELATIONAL DATABASE MANAGEMENT SYSTEMS**

Ihsan Ahmed Taha

M.Sc., Department of Information Technology

Supervisor: Assist. Prof. Dr. Murat SARAN

In the last four 'decades, relational database management systems have been used widely to manage structured data in different disciplines. Nowadays, data resources increase rapidly in different forms (including structured, semi-structured and even unstructured data), requiring us to find another way to manage data. NoSQL database management systems previously were exhibited a few years ago, to meet the requirements of a variety of data forms. Many papers, and blogs have claimed that there is a trade-off between using relational database systems and non-relational database systems (NoSQL). In this study, we propose a novel investigation of the performance of relational and non-relational database systems in terms of the main operations that every database performs, such as insertion, reading, updating, and deleting by using the recent version of Yahoo Cloud Serving Benchmark (YCSB-0.9.0) in order to evaluate the performance of MySQL (RDBM) versus Mongo DB and Apache Cassandra (NoSQL). A testing environment is set up for each workload and the responses for each database management system used in the study are examined for each workload. It is worth mentioning that, although we use the latest version of Yahoo Cloud Serving Benchmark, it does not support the latest release of most database management systems, be they relational or non-relational. The results of this study show that the performance of each system differs due to the differences in their respective data storing mechanisms. In addition, the results of this study present the weaknesses and strengths of each database system used in the study.

**Keywords:** Relational Database Management system, Non-relational Database system, NOSQL, Performance comparison



## ÖZ

# NOSQL VE İLİŞKİSEL VERİTABANI YÖNETİM SİSTEMLERİNİN KAPSAMLI KARŞILAŞTIRILMASI

Ihsan Ahmed Taha

Bilgi Teknolojileri Yüksek Lisans Bölümü

Danışman: Yrd. Doç. Dr. Murat SARAN

Son kırk yıllık sürede ilişkisel veri tabanı yönetim sistemleri, farklı alanlardaki yapılandırılmış verileri yönetmek için geniş çapta kullanılmıştır. Bugünlerde, farklı yapılarıdaki (yapılandırılmış, yarı yapılandırılmış ve hatta yapılandırılmamış) veri kaynaklarının büyüklüğü hızlı bir şekilde artış göstermiştir. Birçok kaynak ilişkisel veri tabanı sistemleri ve ilişkisel olmayan veri tabanı sistemlerinin tercih edilmesi hususunda farklı görüşler ortaya koymuşlardır. Bu çalışmada Yahoo Cloud Serving Benchmark (ycsb-0.9.0 sürümü) uygulamasını kullanarak; veri ekleme, okuma, güncelleştirme ve silme gibi temel işlemler bakımından ilişkisel veri tabanı sistemleri (MySQL (RDBM)) ve ilişkisel olmayan veri tabanı sistemlerinin (Mongo DB ve Apache Cassandra) performanslarının detaylı karşılaştırılması sunulmaktadır. Bu amaçla, temel veri tabanı işlemlerini içeren farklı iş yükleri tasarlanmış ve her bir iş yükü için bir test ortamı oluşturulmuştur. Bu çalışmada kullanılan her bir veri tabanı yönetim sistemi için her bir iş yükü test edilmiş ve sonuçlar raporlanmıştır. Bu çalışmada, Yahoo Cloud Serving Benchmark uygulamasının en son sürümünü kullanıyor olmamıza rağmen, ilişkisel veya ilişkisel olmayan çoğu veri tabanı yönetim sistemlerinin son sürümlerinin bu uygulama tarafından destelenmediğini belirtmek gerekir. Bu nedenle, bu çalışmada Yahoo Cloud Serving Benchmark uygulamasının desteklediği en güncel sürümler kullanılmıştır. Bu çalışmanın sonuçları, her bir sistemin performansının kendi veri depolama mekanizmalarındaki farklılıktan dolayı değişiklik gösterdiğini ortaya koymaktadır. Bunun yanında, bu çalışmanın sonuçları bu çalışmada kullanılan her bir veri tabanı sisteminin zayıf ve güçlü yanlarını sunmaktadır.

**Anahtar kelimeler:** İlişkisel Veritabanı Yönetim Sistemi, İlişkisel Olmayan Veritabanı Sistemi, NOSQL, Performans Karşılaştırma





## ACKNOWLEDGEMENTS

I would first like to thank my thesis advisor, Dr. Murat Saran of the Computer Engineering Department at Çankaya University, without whose helpful advice, valuable comments and guidance this thesis could not be completed. His door was always open for me whenever I needed his help. To whom who lighted up my way through darkness, to whom who made me what I'm today and yet didn't have the chance to see what a great man he left behind, my dear father god bless u. I owe you my past and I owe you my future, with my best regards and love to you father Rest in Peace. To whom who stayed up late many nights and could never blink an eye from me. To Allah's paradise on earth, my mother god bless you wish you many years ahead by my side. And Finally, I would like to thank my wife, friends and teachers for everything.

# TABLE OF CONTENTS

STATEMENT OF NON PLAGIARISM.....	iv
ABSTRACT.....	v
ÖZ .....	vii
ACKNOWLEDGEMENTS.....	ix
TABLE OF CONTENTS.....	x
LIST OF FIGURES.....	xiii
LIST OF TABLES.....	xiv
LIST OF ABBREVIATIONS .....	xv

## CHAPTERS:

<b>1. INTRODUCTION .....</b>	<b>1</b>
<b>1.1 Aim of the study.....</b>	<b>2</b>
<b>1.2 Significance of the study.....</b>	<b>2</b>
<b>1.3 Research Questions.....</b>	<b>3</b>
<b>1.4 Relational database management systems (RDBM).....</b>	<b>3</b>
<b>1.5 NoSQL database.....</b>	<b>4</b>
<b>1.6 Related work. ....</b>	<b>4</b>
<b>2. DISTRIBUTED DATABASES.....</b>	<b>10</b>
<b>2.1 Massive Data.....</b>	<b>10</b>
<b>2.2 Technical terms.....</b>	<b>11</b>
<b>2.2.1. ACID.....</b>	<b>11</b>
<b>2.2.2. Scaling.....</b>	<b>12</b>
<b>2.2.3. Sharding .....</b>	<b>13</b>
<b>2.2.4. Replication .....</b>	<b>14</b>
<b>2.2.5. MapReduce .....</b>	<b>16</b>
<b>2.2.6. CAP Theorem .....</b>	<b>16</b>
<b>3. DATABASES SYSTEMS .....</b>	<b>19</b>

3.1 NoSQL Database classification.....	19
3.2 Relational database types .....	20
3.3 Mongo DB.....	21
3.3.1. Models of Data.....	21
3.3.2. CRUD in Mongo DB .....	22
3.3.3. Mongo DB Aggregation.....	23
3.3.4. Mongo DB Characteristics .....	23
3.3.5. Replication in Mongo DB .....	24
3.3.6. Mongo DB Sharding .....	25
3.3.7. Mongo DB Failure .....	27
3.4 Apache Cassandra .....	28
3.4.1. Data Model in Cassandra .....	28
3.4.2. Apache Cassandra Architecture .....	29
3.4.3. Cassandra writing.....	31
3.4.4. Fault Tolerance Handling in Cassandra .....	32
3.5 MySQL .....	32
3.5.1. MySQL Architecture.....	33
3.5.1.1. Application Layer.....	34
3.5.1.2. Logical Layer .....	34
3.5.1.3. Storage Management.....	36
3.5.2. Storage Engine .....	36
3.5.3. MySQL Replication .....	38
4. TEST ENVIRONMENT .....	39
4.1 Nodes Features .....	39
4.2 Yahoo cloud serving benchmark.....	40

4.3 Apache Cassandra .....	41
4.4 Mongo DB.....	42
4.5 MySQL.....	43
4.6 User Table .....	44
5. RESULTS .....	46
5.1 Load Phase .....	47
5.2 Workload A (50% read, 50% write) .....	48
5.3 Workload B (95% read, 5% update) .....	51
5.4 Workload C .....	53
6. CONCLUSIONS.....	55
REFERENCIES .....	R1
APPENDIX.....	A1
CURRICULUM VITAE.....	A1

## LIST OF FIGURES

### FIGURES

<b>Figure 1</b>	Testing database using two different size of workload.....	5
<b>Figure 2</b>	Throughput versus increased number of nodes.....	6
<b>Figure 3</b>	Relationship between the numbers of threads versus queries (simple query).....	7
<b>Figure 4</b>	Using INNER JOIN QUERY (complex query).....	8
<b>Figure 5</b>	Effect of increasing number of nodes.....	8
<b>Figure 6</b>	Scaling-up versus scaling-out.....	13
<b>Figure 7</b>	Sharding.....	14
<b>Figure 8</b>	Replication.....	15
<b>Figure 9</b>	MapReduce functions for word counting .....	16
<b>Figure 10</b>	CAP Theorem and databases.....	17
<b>Figure 11</b>	Documents in Mongo DB.....	22
<b>Figure 12</b>	BI tool in Mongo DB .....	24
<b>Figure 13</b>	Mongo DB Replication.....	25
<b>Figure 14</b>	Mongo DB Sharding.....	27
<b>Figure 15</b>	Cassandra architecture.....	30
<b>Figure 16</b>	RDBM layers.....	33
<b>Figure 17</b>	Logical Layer Subsystems.....	35
<b>Figure 18</b>	Replication in MySQL.....	38
<b>Figure 19</b>	YCSB Architecture.....	41
<b>Figure 20</b>	Cassandra cluster.....	42
<b>Figure 21</b>	Mongo DB Cluster.....	43
<b>Figure 22</b>	MySQL Cluster.....	43
<b>Figure 23</b>	Load phase.....	47
<b>Figure 24</b>	Workload A 50% update.....	48
<b>Figure 25</b>	Workload-A 50% read.....	50
<b>Figure 26</b>	Workload-B (mostly read).....	51
<b>Figure 27</b>	Workload C (Read only).....	53

## LIST OF TABLES

### TABLES

<b>Table 1</b>	Load Phase	47
<b>Table 2</b>	Mongo DB update (W.A)	48
<b>Table 3</b>	Cassandra update (W.A)	49
<b>Table 4</b>	MySQL update (W.A)	49
<b>Table 5</b>	Mongo DB Read (W.A)	50
<b>Table 6</b>	Cassandra Read (W.A)	50
<b>Table 7</b>	MySQL Read (W.A)	51
<b>Table 8</b>	Mongo DB 95% Read (W.B)	52
<b>Table 9</b>	Cassandra 95% Read (W.B)	52
<b>Table 10</b>	MySQL 95% Read (W.B)	52
<b>Table 11</b>	Mongo DB 100% Read (W.C)	54
<b>Table 12</b>	Cassandra 100% Read (W.C)	54
<b>Table 13</b>	MySQL 100% Read (W.C)	54

## LIST OF ABBREVIATIONS

RDBMS	Relational Database Management System
NOSQL	Non-Relational Database Management System
CRUD	Create, Read, Update, Delete
ACID	Atomicity, Consistency, Isolation, Durability
CAP	Consistency, Availability, Partition Tolerance
YCSB	Yahoo Cloud Serving Benchmark

## **CHAPTER 1**

### **INTRODUCTION**

Due to the revolution of technology, a huge number of different data sources have appeared, that produce huge data sizes, which is considered one of the biggest obstacles to managing giant data. A couple of decades prior, organizations began managing information by presenting Relational Database Management Systems (RDBM). In 2009, many different versions of database management systems appeared when NoSQL database systems appeared. Numerous establishments have asserted that their database managements systems are superior to others. In order to clarify these assertions, we have conducted a study to compare the performances of NoSQL and relational database management systems. In order to do such a comparison, the tool used to benchmark the database management systems in this study is the latest version of Yahoo Cloud Serving Benchmark (0.9.0). Two of the NoSQL database management systems we used in this study are Mongo DB and Cassandra. MySQL database management system represents the relational database management systems that have been used in this study. Although there have been many papers introduced in this field, our study considers a novel approach due to the latest releases of the database systems used in this study which is supported by the Yahoo Cloud Serving Benchmark (YCSB). Relational databases have been used for more than 40 years. Since the revolution of web applications that exploded in the 1990's, the need for efficient databases has become more important. NoSQL database was found in 1960, but it was not in wide use until 2009 [1]. The foundations built their databases in a variety of ways with different programming languages. This variation leads to different performances and responses even for the same type of databases. For instance, NoSQL database has more than 150 database management systems according to a blog in [2]; all these products 'do not share the



same performance due to differences of design. Therefore, we need to investigate it in this field in order to clarify the characteristics of both relational database systems and non-relational database systems (NoSQL).

### **1.1 Aim of the study**

Relational database management systems have been used in the last four decades to manage different types of data, but recently, a new era of managing data started upon the appearance of the NoSQL database. Our goal is to test the performances of two NoSQL database management systems against one relational database management system in terms of creating, reading, updating and deleting (CRUD) operations. We examined the performance of each system from different perspectives by applying different workloads via the Yahoo Cloud Serving Benchmark (YCSB). The performance of each system differs due to the differences in their respective data storing mechanisms. The results of this study show the weaknesses and strengths of each database system used in the study. We set up a testing environment for each workload and examined the responses for the three systems.

### **1.2 Significance of the study**

Many papers and blogs introduced comparison studies between RDBM and NoSQL in order to clarify a number of claims about these two types of database. For instance, there is a claim saying that NoSQL displaces RDBM simply because NoSQL is a newer technology. The second issue that needs to be investigated is to see whether or not it is correct that NoSQL is better than RDBM. Another issue on which we focused in this study was to see how both NoSQL and RDBM responded to several requests. Nowadays, there are in excess of 150 different NoSQL databases with different characteristics [2]. Moreover, many relational databases have appeared. Due to this variation, it is not easy for IT managers to select the best database that meets requirements. The results of this study will help IT decision

makers with regard to understanding the characteristics of both NoSQL and RDBM databases. In addition, the results will guide them while selecting a suitable NoSQL or RDBM database management system that best suits their needs. Although many studies have introduced comparisons between NoSQL and RDBM database management systems, in this study, we have used the latest versions of NoSQL database systems supported by the latest release of the Yahoo Cloud Serving Benchmark (YCSB-0.9.0). In this study, we have highlighted the most widely used NoSQL databases: MongoDB (3.2.4), Apache Cassandra (3+) and from RDBM we selected MySQL (5.5).

### **1.3 Research Questions**

1. Which database management system (relational database systems, or NoSQL database systems) is better according to low average latency and high throughput in terms of create, read, update operations?
2. Which NoSQL database (MongoDB (3.2.4) or Apache Cassandra (3+)) is better (according to low average latency and high throughput) than the other in terms of create, read, and update operations?
3. Are the foundations presenting any improvements to their products comparing with the last published studies?

### **1.4 Relational database management systems (RDBM)**

The relational model and the transactional model theories appeared during the '1970s and '1980s, which developed the concepts of these two theories to be considered the basics of relational database management systems. The relational model was used to present full access to the data stored in the databases while the transactional model was used to perform different operations on data reliably. These two concepts are explained in detail in the following chapter. In general, it is safe to say that RDBM is

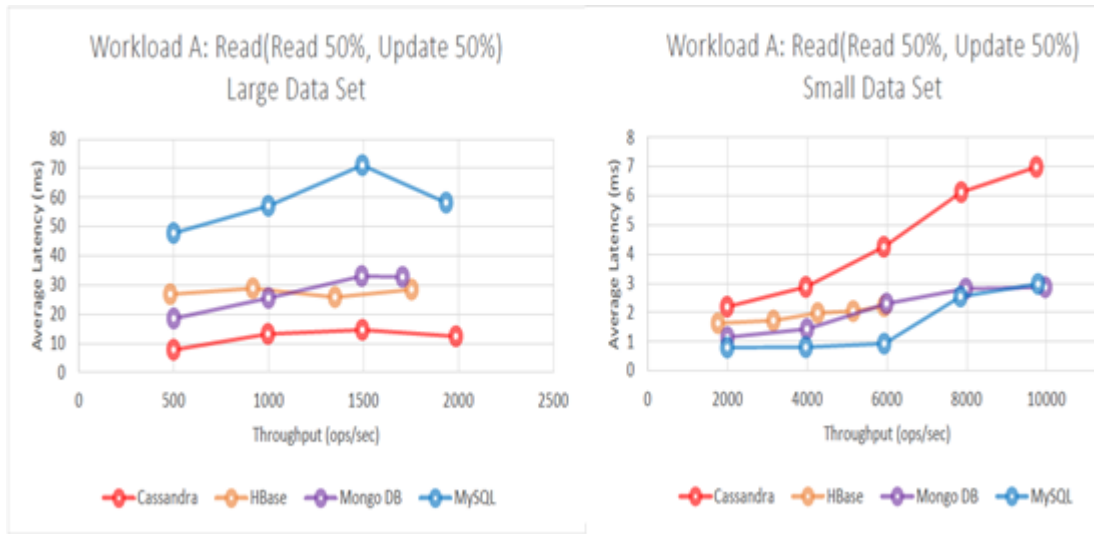
built to scale-up [3]. With a rapid increase of data every day, the challenge of producing a perfect database system becomes more difficult.

### **1.5 NoSQL database**

NoSQL is a stand for (not only the SQL database). There are more than 150 different products [2], and this new type of database system appeared when large companies such as Google, Facebook and Amazon endeavored to discover how to create a new database that is able to support scaling-out beyond the limits of relational database systems. NoSQL database systems have the ability to scale-out up to more than a factor of hundreds [4]. It is worth mentioning that NoSQL database systems do not support relational and transactional models as there is no need for their services. NoSQL databases are practically designed to work on clusters comprised of computers (nodes).

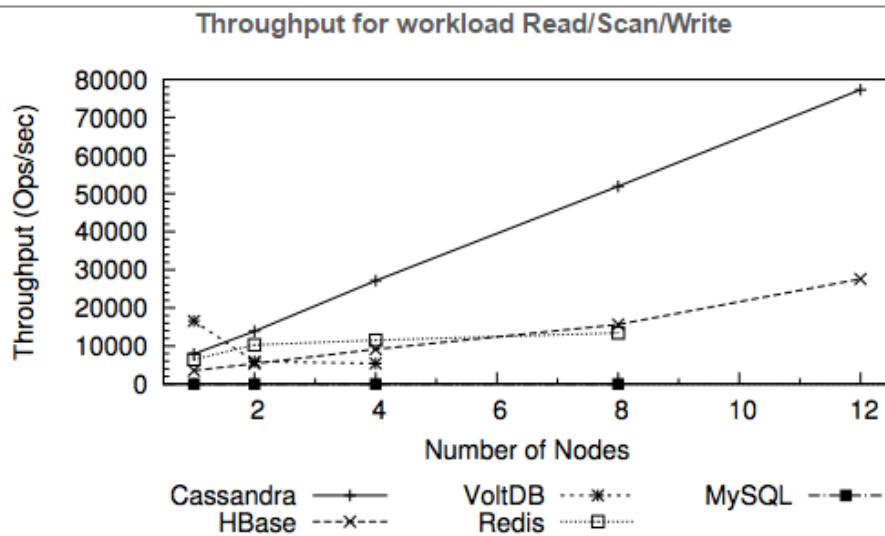
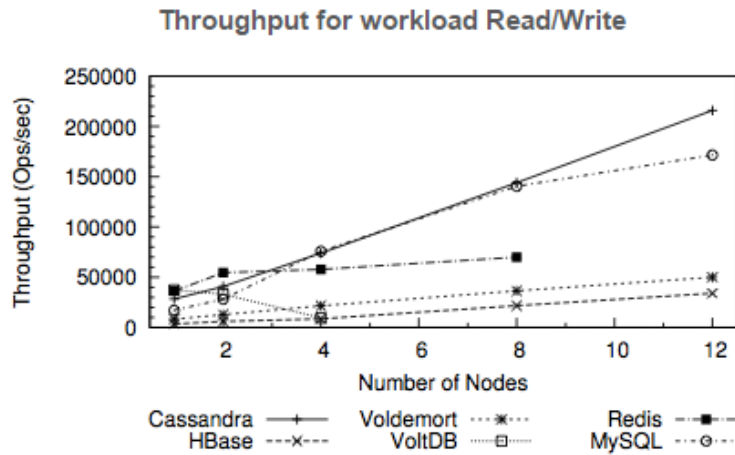
### **1.6 Related work**

Recently, many studies have introduced comparisons between NoSQL and RDBM in order to clarify the features of these two types of database. In [5], the author used the NoSQL database systems (Mongo DB, Apache Cassandra, and Apache HBase) with one RDBM database (MySQL) using the Yahoo Cloud Serving Benchmark (YCSB) to test their performance by applying the standard workloads that come with (YCSB). The results are shown in the Figure 1:



**Figure 1** Testing database using two different size of workload

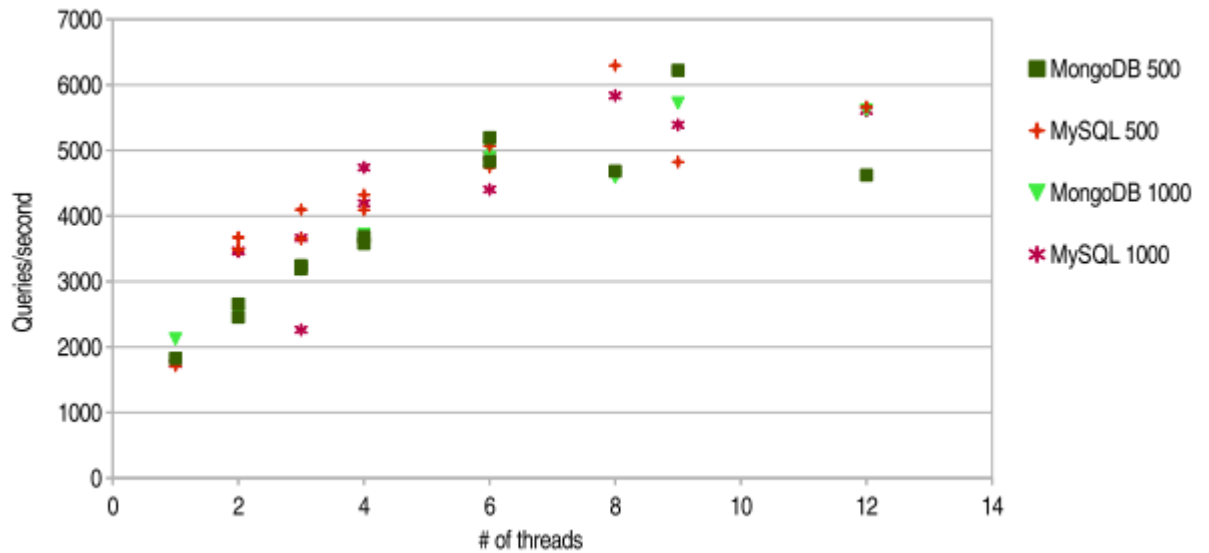
The author used two different sizes of data set, namely a small size (1, 000, 000) and large data set (100, 000, 000). The author concludes that, of all the systems, Cassandra performs better with large data sets, and the throughput is the highest with lowest average latency. MySQL performs well with small data sets. In addition, the author concludes that there is no good or bad database system, and that choice of database depends on need. HBase performed well in general, but not as well in the update. Cassandra has impressive throughput with latency cost, and Mongo dB performed well with small data sets. Although HBase and Cassandra were better than Mongo dB in terms of throughput and input, Mongo dB has a better read and update than Cassandra and HBase. The foundation of Apache Cassandra presented a comparison in [6] of several database systems as shown in the Figure 2:



**Figure 2** Throughput versus increased number of nodes

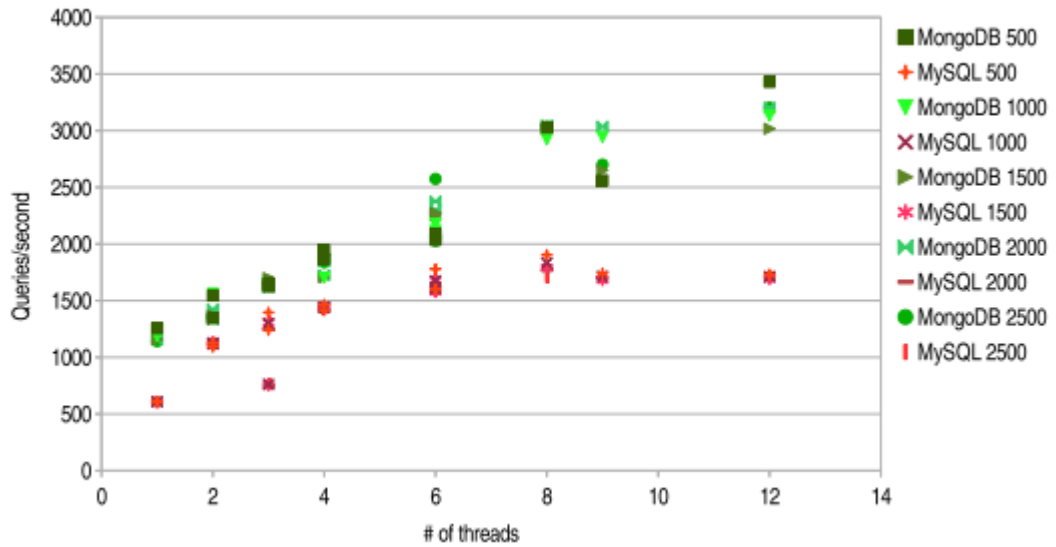
According to these results the author concluded that as the number of nodes increases, the performance of some database systems also increases. Apache Cassandra ranks first with the highest throughput (operations per second) with an increase in the number of nodes. MySQL takes the second place during the (read write) operation; however, we 'do not see the same good performance during the (read, scan, write). In fact, there is no throughput at all. The other NoSQL databases show good reasonable throughput with an increase in the number of nodes. In [7], a comparison of a NoSQL database against RDBM databases was proposed. The author tested Mongo DB against MYSQL in terms of number of threads using Yahoo Cloud Serving Benchmark to analyze the performance of both systems while

increasing the number of threads and its effect on the number of queries. The author found that when the number of threads was increased to 4, the winner was MYSQL. However, when the threads were increased past 4, the performance of both systems was almost identical.



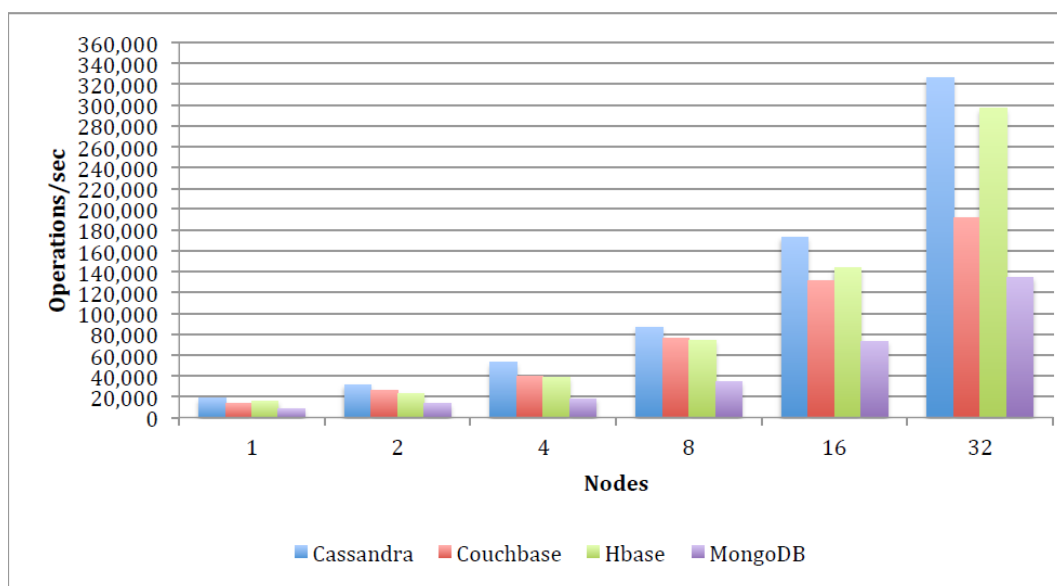
**Figure 3** Relationship between the numbers of threads versus queries (simple query)

Unlike Figure 3, we can see a significant difference in Figure 4 that shows that Mongo DB performs better when the author used complex queries which included an inner join in SQL terms. In contrast to Mongo DB performance, MYSQL showed a linear performance when compared with MYSQL performance without using the inner join in Figure 4.



**Figure 4** Using INNER JOIN QUERY (complex query)

The team of end point in [8] performed a series of tests on several NoSQL database systems based on the number of nodes. They increased the number of nodes so as to be doubled in each test starting with 2 nodes and proceeding up to 32 nodes. They noticed that the performances changed with an increased number of nodes. Each system shows different responses for each number of nodes and for each workload, as can be seen in the Figure 5[8].



**Figure 5** Effect of increasing number of nodes

We can recognize that Cassandra throughput (operation/sec) increases more than the other systems when the number of nodes increases, which means that the number of nodes affects the system performance. In [9], the authors compared the relational and non-relational database systems of 14 different NoSQL databases in different perspectives, such as their data models, query possibilities, concurrency control, partitioning and replication opportunities. They concluded that NoSQL databases are better for operations that are very fast and simple for very large datasets than they are for relational database systems.





## CHAPTER 2

### DISTRIBUTED DATABASES

The aim of this section is to clarify the obstacles facing distributed databases and the techniques that are ordinarily used to overcome them. It indicates how databases can be scaled and why distributed databases need to make a few trade-offs to accomplish this. We additionally concentrate on scaling, replication and sharing.

#### 2.1 Massive Data

Due to the fast growth of data, which comes from a very large number of sources such as satellites, sensors, social media etc., it becomes almost impossible to manage this huge volume of complex data. Big data can be defined as the capability of managing a huge volume of data in a timely manner and at proper speed [11]. Massive data can be described in three terms: volume, velocity, and variety. Volume refers to the quantity of data stored and generated, velocity refers to how fast that data is generated and processed, and variety refers to the nature of the data and to various types of data. Foundations have attempted to find solutions to handle this huge volume data. The first action was to use better hardware; however, this approach was insufficient as the hardware enhancement reached a point where the growth of data volume outpaced computer resources [12]. There are three types of massive data, the first one being structured data, a term referring to the fact that the format of the data and the length are known. Examples of structured data include email, phone numbers, IDs, names, addresses, etc. There are two sources that provide structured data: data generated by human intervention such as gaming data and input data. The second source is the data generated by machines such as sensor data, web log data and financial data. The second type is unstructured data, which is the data

that do not have specific formats or known lengths. These are found everywhere and are used widely. The sources of these unstructured data are human-generated and include website content, mobile data and social media. The data generated by machines are the second source of unstructured data, and these can be found as radar data, sonar data and satellite data. The third type is semi-structured data; this kind of data combines structured and unstructured data. Dealing with this degree of data complexity is not easy. Tall data and wide records lead to long running queries; therefore, new methods are needed in order to overcome this challenge and manage huge data.

## **2.2 Technical terms**

In this section, we mention a number of terms used throughout this thesis and terms relevant to database systems in order to present a good understanding of database systems.

### **2.2.1 ACID**

The term ACID stands for Atomicity, Consistency, Isolation, and Durability. The ACID acronym appeared in 1980 in order to present a static standard for the required properties which enable RDBM to run in such a way that prohibited data loss and exceptions.

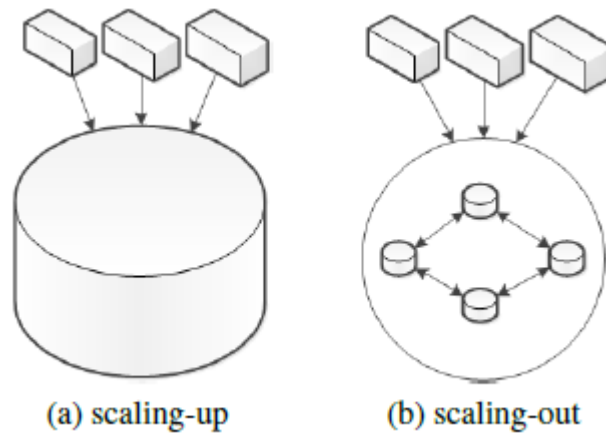
1. Atomicity: meaning that every transaction should be atomic. This feature is used to guarantee that whenever the database faces an error during the transactions, at that point, none of the calls made will be committed to that database. Therefore, a log file is used in Standard RDBMS to ensure these demands. If a write operation to a database occurs, either all the updates that occurred during the write operation will be available to every client or none will be available.

2. Consistency: meaning that the database is always in a consistent state before and after transactions occur. If the changes of a transaction are against the consistency rule, then all changes of the transaction must be cancelled to ensure that only valid data is written to the database.
3. Isolations: meaning that each transaction should be run separately from every other transaction. The importance of this property appears when there is a concurrent transaction.
4. Durability: this property is important when an update occurs to the database and a system failure takes place. In other words, this property is used to ensure that the transaction is committed to the database and it will not be lost.

### **2.2.2 Scaling**

Scalability can be achieved using two technologies: Sharding and replication. Scalability for databases can be done in three different ways: read operation, write operation and the volume of database. There are two terms related to databases that need to be explained:

- Scale-up: this concept means that increasing the power of the machines, such as increasing their memories, using faster processors, increasing the number of cores, using faster memories, etc. in order for the database to be able to process more operations.
- Scale-out: increasing the performance of database systems can be achieved by distributing the machines to work together in a distributed manner so as to serve the requests of managing data, as seen in the Figure 6:

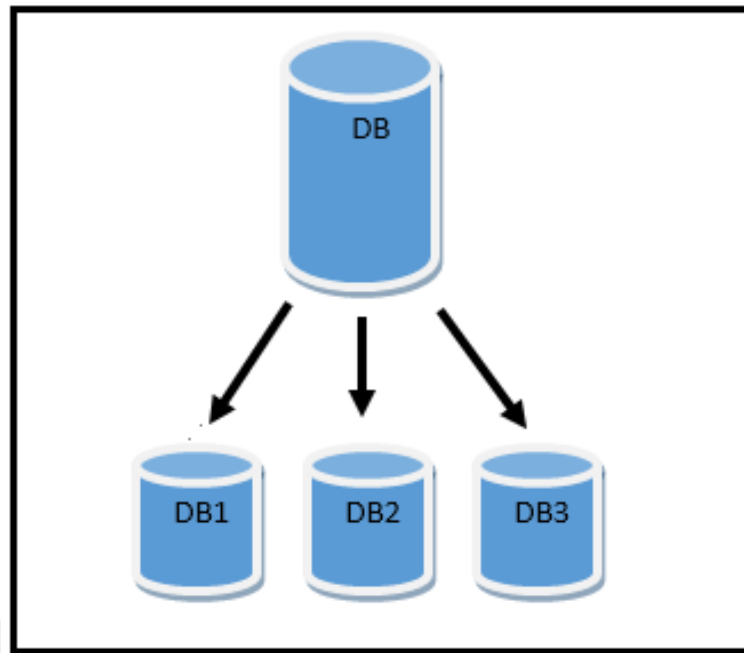


**Figure 6** scaling-up versus scaling-out

Although scaling-up and scaling-out are used to overcome the problem of increasing large amounts of data, scaling-up still has limits that cannot be passed. It is impossible to keep adding memory or cores to increase the power of a machine. Based on this fact, we can say that scaling-up is limited by hardware. In contrast, we can simply add more machines to clusters and expand our clusters to face the challenges of the tremendous increase of data.

### 2.2.3 Sharding

The concept of Sharding can be defined as the process of splitting data inside a database into numerous shards; these shards can be distributed through the nodes. This definition leads us to another understanding: instead of saving data on one node, it will be stored within a cluster of nodes as shards and not as a single large volume of data. A very important advantage achieved while using Sharding is that nodes can be added to the cluster easily, which means that capacity will increase, thereby affecting the performance of the cluster positively. The performance of the read and write operations is enhanced while adding additional nodes without the need to modify the database system.



**Figure 7** Sharding

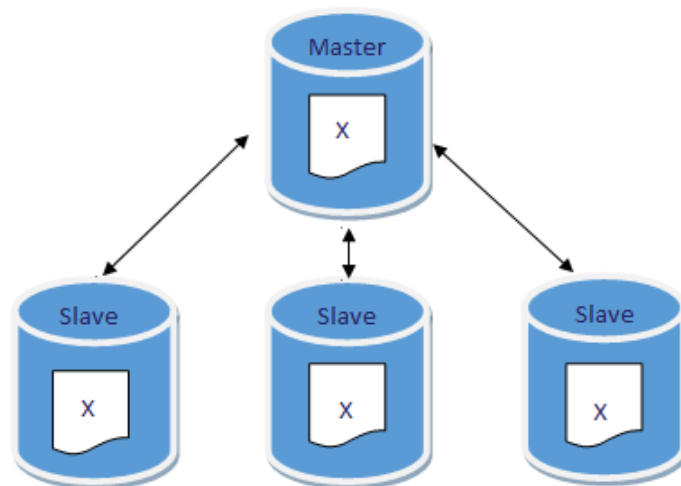
On the other hand, there is also a downside to Sharding on some databases operations as Sharding makes these operations more complex and inefficient. For instance, the join operation in relational database systems (RDBM) is considered one of the most important operations. It is used to materialize the relations of data items [10]. A distributed join in a sharded database requires many requests over the cluster nodes to perform an operation on an item, 'leading to an increase in network traffic.

#### **2.2.4 Replication**

Replication means that the data in a distributed system is repeated through every node within a cluster, thereby leading to an increase in the performance of the read operation. This is due to the fact that the load balancer will distribute the read requests throughout the nodes. There are two techniques to execute replication:

1. Master-slave replication: The possibility of this structure is to set one node as a master which handles write operations, and a slave which is synchronized with the master and conducts activities, such as read operations.
2. Peer-to-peer replication: This provides the capacity to remain in contact with any node and to synchronize data crosswise over nodes. Master-slave replication diminishes the likelihood of update conflicts, while peer-to-peer replication prevents loading all writes onto a single server and creating a single point of failure. Nowadays, a system may use either technique or both techniques.

Replication is very useful as it renders database systems immune to a number of catastrophic occurrences. On the other hand, there is a disadvantage such that once there is a write an operation, the data must be written on every other node, and acknowledgment will not be returned until the data is written through all the nodes.



**Figure 8** Replication

## 2.2.5 MapReduce

In 2004, Google presented a new technique called MapReduce, which is described in a corresponding article in [13]. MapReduce can be defined as a technique that is used to acquire a huge amount of data within a short time. Several computers are used to divide the tasks of filtering and fetching data. The MapReduce concept comprises two main functions, mapping and reducing. The map function is used to generate intermediate (key-value) pairs by combining a key and a value. After all the intermediate keys are generated, the reduced function will use these generated keys to move all the intermediate values with their intermediate keys.

```
map(String key, String value):
    // key: document name
    // value: document contents
    for each word w in value:
        EmitIntermediate(w, "1");

reduce(String key, Iterator values):
    // key: a word
    // values: a list of counts
    int result = 0;
    for each v in values:
        result += ParseInt(v);
    Emit(AsString(result));
```

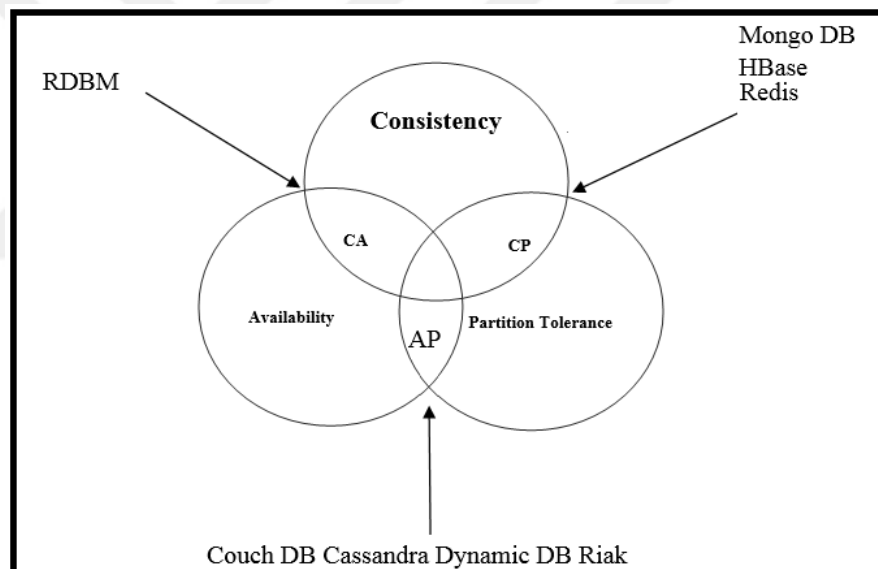
**Figure 9** MapReduce functions for word counting [10]

## 2.2.6 CAP Theorem

CAP is an acronym that stands for three concepts: Consistency, Availability, and Partition tolerance. It was proposed by Dr. Brewer in a keynote speech it was then formalized by Gilbert and Lynch [14]. This theorem says that two out of the three features can be provided in any distributed system.

1. Consistency: While performing read or write operations within a cluster from/to any node, the data will remain the same through the nodes in that cluster.

2. **Availability:** In case a node within a cluster goes down, we can still access that information in the cluster. Failure of that node will not affect access to the data.
3. **Partition Tolerance:** This concept means (connection loss) between the nodes of the cluster. Although there is a misconnection between nodes within a cluster, the cluster will continue working. For any distributed system, it is impossible to satisfy all the properties simultaneously. Partition tolerance must be available in every distributed system; otherwise, this system is not a distributed system. Therefore, any distributed system will have two properties at most, namely partition tolerance and one of availability or consistency.



**Figure 10** CAP Theorem and databases

Based on this theorem, databases are divided into three categories:

1. **Availability and Partition Tolerance (AP) systems:** This type of feature is found in Cassandra, Couch DB, and Simple DB. If there is a misconnection between nodes within a cluster, once the partition has resolved the nodes, they will communicate between each other and resync data; then the data



will be available once again. It is difficult to know whether all nodes will sync their data.

2. **Consistency and Availability:** Unlike distributed systems which should have partition tolerance, CA systems cannot be described as a distributed system because of the absence of the partition tolerance feature in these kinds of systems. RDBM database systems are good examples of CA systems.

**Consistency and Partition Tolerance:** HBase and Big table are good examples of CP systems. Because there is no availability, once one of the nodes within a cluster goes down, the cluster will not be accessible. According to some studies, they consider Mongo DB a CP system. Other recent studies have refuted that claim because Mongo DB does not always behave identically to CP systems.

## CHAPTER 3

### DATABASE SYSTEMS

In this chapter, we present both the RDBM and NOSQL database system types and we present particularly the database systems used in this study, namely Mongo DB, Cassandra, and MySQL, and their respective characteristics. We present features such as their respective architectures, data models, and query models. We also discuss how each database deals with replication, Sharding and how they respond in the event of failure.

#### 3.1 NoSQL Database Classification

There are more than 150 types of NoSQL database used in different disciplines, all of which can be classified into four main classes according to the mechanism of storing data.

1. Key value store

This type uses a hash table to manage data. The idea of the hash table is simple, as there is a unique key and pointer to point to a specific item of data. A key-value is considered the simplest type of data store. It is also considered inefficient if the main purpose of using it is only for reading and writing. It is used in Riak, Tokyo Cabinet/Tyrant, Redis, Voldemort, Oracle BDB, and Amazon Simple DB.

## 2. Documents data stores

This deals with semi-structured data and is similar to the key-value store. However, it differs in the way it processes data. Documents are used to store data in the JSON format. It is sometimes called a document-oriented database. It is used in Couch DB, and Mongo DB.

## 3. Column Family Stores

The data in this type is stored in columns. There are keys, but they are used to point to columns instead of data items. The columns are arranged in column families. It can be found in Cassandra, and HBase.

## 4. Graph data store

Unlike the other three types of NoSQL, the graph data store provides a graph model to store data instead of rows, and columns. It is used in Neo4J, and InfoGrid.

### **3.2 Relational Database Types**

There are several classifications for the (RDBM). For instance, we can categorize (RDBM) based on the relationships between tables as follow:

1. One-to-one: meaning a record in the database table is connected to one, and only one, record in another table. For instance, if we have table A and table B, each record in table A is linked to each row in table B, which means the number of rows in both tables should be equal.
2. One-to-many: In this relationship, each record in table A might be linked to one, or many records, in table B. It is useful when saving data once in one table and referenced several times to every other table. In this case, the number of records in table A is usually less than the number of records in table B.

3. Many-to-many: One record, or many, in table A is linked to one, or many, records in table B. It is important to mention that a third table is used to implement such a relationship, known as a mapping table.

### **3.3 Mongo DB**

Created by the 10gen foundation, Mongo DB is an NOSQL database system which appeared in the last couple of years. Mongo DB is considered a schema less document-oriented database. It is an open source build using the C++ language and it is used to store large volumes of data with a vast range and variety of data types. Mongo DB is considered to be fast and scalable.

#### **3.3.1 Models of Data**

The format used to store documents as Binary JSON (BSON) objects. Unique keys “\_id” in documents in each collection in order to avoid collision and manage the data efficiently. MongoDB is both case and type sensitive. There are common features for these keys that can be stated as follows:

1. Keys are unique so there is no possibility of duplication.
2. Null characters cannot be used to generate keys as they are used to indicate the ends of keys.
3. To define a scenario, dots and dollar signs are used.

```
{
  "name": "learnwebtutorials",
  "version": "0.0.1",
  "description": "First nodejs app",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "BSD-2-Clause",
  "dependencies": {
    "express": "~3.4.8",
    "consolidate": "0.10.0",
    "mongodb": "1.4.0"
  },
  "engines": {
    "node": "0.10.x"
  }
}
```

**Figure 11** Documents in Mongo DB

### 3.3.2 CRUD in Mongo DB

In this section, we focus on the performance of Mongo DB in terms of the four main operations that every database uses: create, read, update, and delete. Create, update and delete operations are considered data modification operations. The read operation is considered to be a query and in Mongo DB, it is a query that deals with a specific document within a collection. To read from a specific field from a document, a Mongo DB query may use a projection to specify that field. MongoDB is bundled with several types of queries. The query model of MongoDB allows queries over all documents inside a collection, including embedded objects and arrays [15]. Mongo DB shows several query behaviors as follows:

1. Mongo DB queries address one collection.
2. Queries in Mongo DB can be modified to put limits, sort orders, and skips.
3. To obtain the order of documents by query, a sort method should be used.

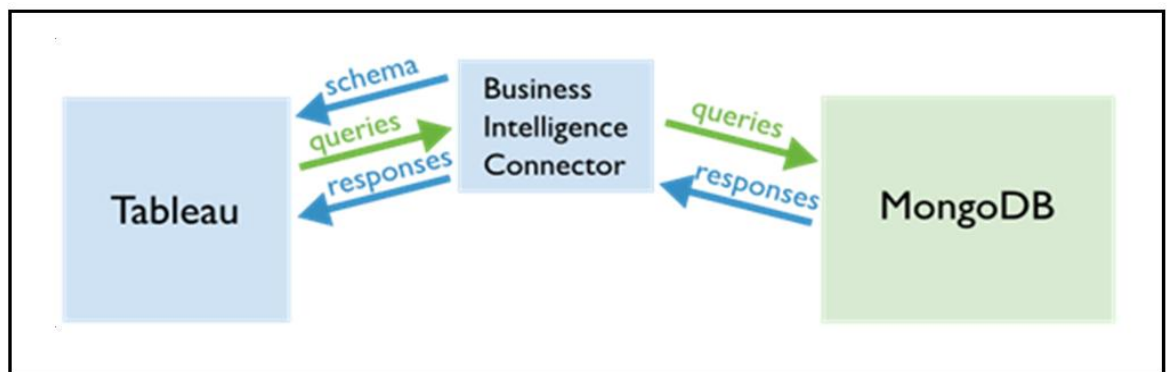
### **3.3.3 Mongo DB Aggregation**

The processing of data and the returning the computed results of that data is called the aggregation process. Values are collected from several documents; several operations are applied on any data that is not similar to each other, and one result is returned. Mongo DB presents a vast set of aggregation operations. The input to the aggregation operation is the documents within collections and the results are returned.

### **3.3.4 Mongo DB Characteristics**

Mongo DB is considered an elite database system. It is an open source database which, is also a capable and adaptable document oriented database. Data is put away in records and collections rather than in tables. This enables the process of representing complex collections more effectively. Mongo DB is an adaptable database with rich auxiliary files including geospatial and TTL records. It is able to store different types of data such as structured, semi-structured, and unstructured data. It can also perform very large numbers of operations. It shows high scalability due to the architecture that is used to design Mongo DB. It is suitable to be used in vastly different applications such as social networks, Internet of Things, and cloud computing. In cases of failure, it is easy to administer and overcome any issues. Version 3.2 of MongoDB is packaged with a memory mapped storage engine. With every update for Mongo DB, we can see improvements, such as a new connector for business intelligence, which was added to Mongo DB recently, thereby giving users the ability to visualize enterprise data in Mongo DB by using relational business intelligence tools such as Tableau. The aim is to connect these tools to a datacenter and to find data in tabular form. This is not an easy task when working with MongoDB. The purpose of creating a MongoDB connector for BI components is to

connect the MongoDB server with Business Intelligent tools without storing any data. We can describe how the tool works in the Figure 12:



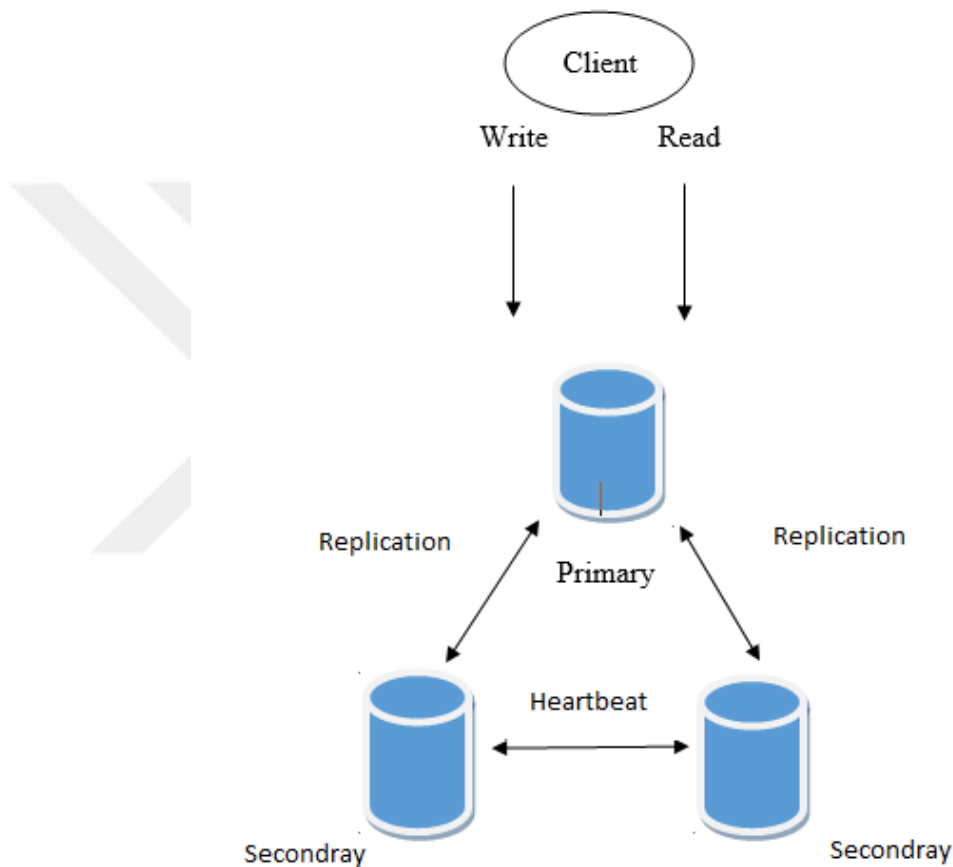
**Figure 12** BI tool in Mongo DB [16]

In addition, Mongo DB is designed to visualize a scheme, which means it uses a graphical interface. Mongo scout is used to deal with several operations, including analyzing collections in order to visualize the availability of fields and the cardinality of their values. It also supports dynamic lookups, which is a new method for modeling data and providing great flexibility. Another feature was added to Mongo DB is document validation, which is simply used to ease the burden for companies during the process of verifying data types. Security is one of the greatest concerns in all data base management systems. Mongo DB is considered to be a reliable database system due to how it is used to deal with data by encryption of the data at reset.

### **3.3.5 Replication in Mongo DB**

As explained in Chapter 2, the concept of replication in Mongo DB is similar to master-slave. The replica set composed of several mongod processes which are used to maintain the data set [17]. The Figure 13 shows replication in Mongo DB. Unlike master-slave, a replica set has an automatic failover mechanism in case the primary node becomes unavailable. If any connection problems occur between the primary

and secondary nodes, one of the secondary nodes will become the primary. Because of this mechanism, a replica set provides redundancy and availability. A replica set can be expressed as a cluster of nodes of size N. A replica set cannot have more than one primary node, which is the only node that can accept writing operations. When the client starts sending data to the primary replica, these data will be copied and passed to the secondary replica nodes. Replication is very important to reduce the risks of losing data.



**Figure 13** Mongo DB Replication

### 3.3.6 Mongo DB Sharding

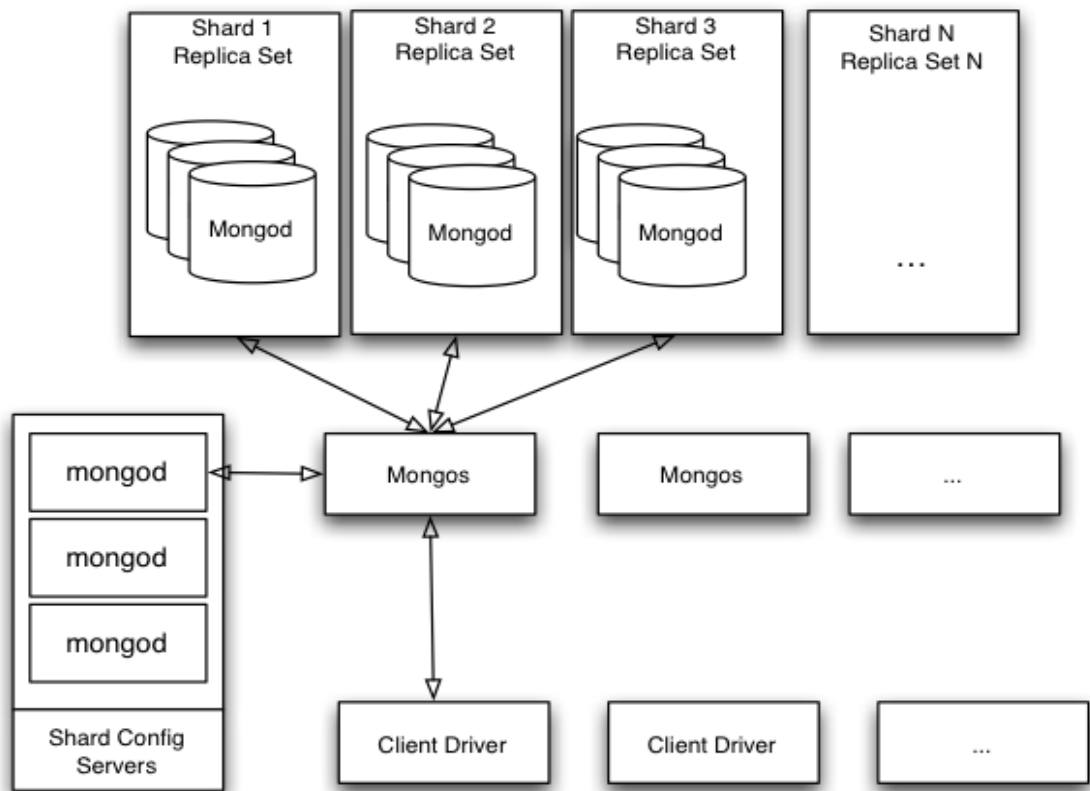
Mongo DB presents a Sharding procedure which can be characterized as a way of putting away data on a few machines, which is an excellent position with quick



incrementing of data. A solitary machine that is utilized to store data may not be suitable due to low throughput between the read/write processes. Sharding overcomes this issue by utilizing even scaling. The Mongo DB bunch has three parts:

- Mongos (routing server)
- Shards
- Configuration server

Mongo DB is designed to support Sharding, which is the process of breaking a huge volume of data into smaller volumes in order to overcome the limits of hardware, as was explained in Chapter 2. Bottlenecks in RAM or disk I/O are considered examples of hardware limitations. Mongo DB consequently adjusts the data in the sharded cluster as the data develops to the extent of the group increments or abatements. Unlike in social databases, Sharding is programmed and incorporated with the database, minimizing the weight for designers and operations groups. Every shard contains a replica set and the shards are utilized to store real information. Expanding the quantity of nodes inside every shard prompts expanding repetition and accessibility. Configuration servers (mongod) are utilized to hold metadata which contain mappings for the genuine data in the shards. Directing servers utilize metadata to course operations to specific shards. Mongo DB disperses data, or shards, at the collection level. Sharding allots a collection's data with the shard key [18]. The initial step to shard a collection can be done with a Sharding key. A shard key is not unlike ordering. A shard key is partitioned into chunks equitably over the shard by Mongo DB. Every chunk contains a few records all together. The fundamental advantage of the pieces lies in adjusting the shards. In the event that a shard estimate becomes larger than the alternate shards, a few substances of the piece will be relocated to other smaller shards so as to rebalance the sizes of the shards. In the event that another node is included or expelled from the cluster, the pieces will redistribute the information over the group.



**Figure 14** Mongo DB Sharding

### 3.3.7 Mongo DB Failure

A Mongo DB cluster may fail for many different reasons, so Mongo DB is designed in such way as to overcome node failure while managing data. If a node within a cluster goes down, the cluster will not be affected by this problem and it will continue to perform. In the crashed node, the data may be corrupted, so whenever this node recovers and works correctly, the data in it will need to be maintained. The worst-case scenario occurs when every node within a shard is broken. MongoDB then will not be able to execute any operation on the data in this shard. The same situation occurs when one of the configuration servers fails. MongoDB will also lose the ability to split and merge data between shards. In general, Mongo DB is considered to be a reliable system due to the ways it follows to prevent data loss.

### **3.4 Apache Cassandra**

Apache Cassandra is an open source database management system, written in Java and one of the NoSQL database systems used widely in many disciplines. Cassandra presents high scalability and it is described as being a reliable database system, that also it supports fault tolerance. Facebook originally represented the developer of Cassandra in order to deal with the inbox search feature that provides users with the ability to search through their Facebook index [19]. After that it took all the concentration of Apache foundation. Cassandra does not require the matching of a column within a row. Due to Cassandra's decentralized architecture, any node can respond to requests, thereby enabling Cassandra to avoid single node failure.

#### **3.4.1 Data model in Cassandra**

In Casandra, the data model composed of the following items:

1. Keyspace: In a Cassandra hash, the first dimension is represented by the keyspace; the family columns are contained in the Keyspace.
2. The rows in Cassandra comprise collections of columns with keys for identification purposes. Each row contains several files and each file contains a column family. In order to know which node stores data, a key row is used.
3. Columns comprise three parts, timestamp, name and value. A timestamp is used to clarify the resolution during the last stage of writing.
4. Super columns is a term that refers to a column whose values are columns.

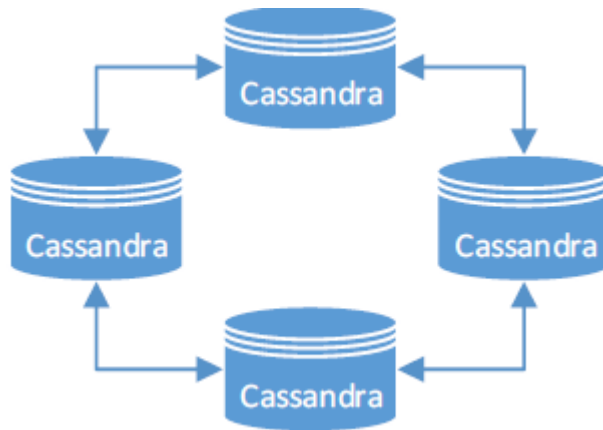
Apache Cassandra a covers wide range of data that deals with the following:

- ASCII
- Lexical UUID

- Long
- Timer UUID
- UTF8
- Integer
- Bytes

### **3.4.2 Apache Cassandra Architecture**

A Cassandra cluster is based on a peer-to-peer relationship. This approach to designing such a relationship is very useful because it can be used to overcome the problems that occur from time to time in some database systems using master-slave relationships. The master represents the part which handles data management, while the slave synchronizes data with the master. Therefore, in cases of the master going down, the relationship will crash. In Cassandra, the peer-to-peer relationship avoids or solves such problems. Every node in the cluster plays an identical role. There is no master in the Cassandra architecture which represents a point of failure. The data are split among all the nodes within the cluster. Due to peer-to-peer networking, the performance of the database improves. We should mention that the node characteristics within Cassandra clusters, the first feature of the node, and each node within the clusters perform the same function. Although the query data is not located on a specific node, every node in the cluster responds for different operations, such as read/write. In addition, if a node fails, the cluster continues to perform. There are some terms and concepts related to Cassandra architecture.



**Figure 15** Cassandra architecture

We highlighted the concepts related to Cassandra architecture as follows:

1. Nodes: the place where data are stored.
2. Data center: a group of connected nodes. There are two types of datacenter: physical and virtual.
3. Cluster: A cluster is composed of one or more datacenters.
4. Commit log: For durability, the data are first written to the commit log. When the data are flushed to the SSTable, they can be updated, deleted, or even archived.
5. Table: a group of columns organized in order that can be fetched by using a key. Each row has a primary key.
6. SSTable: an acronym for Stored String Table which is a permanent data file to which Cassandra composes memtables intermittently.
7. Boom filter: a term concocted in 1970 by Burton sprout. It is a probabilistic data structure that is utilized to inform the user regardless of whether data for a specific row exists in the SSTable by utilizing a quick, non-deterministic calculation. It assumes an imperative part particularly with vast volumes of information as a sponsor. It is thought to be a store memory which permits fast inquiries.

There is a configuration for the key in Cassandra, and we present the components with details as follow:

1. Gossip: used to open connections between nodes so that they can communicate with each other. It used for broadcasting and receiving data every second through the cluster.
2. Partitioner: used to determine where the first copy of data takes place within a cluster. It is used to manage how the data are distributed over the cluster. It is a hash function used for token evaluation. The partition key is used to identify every data row.
3. Replica placement mechanism: As explained in the previous chapter, replication means saving copies of data on more than one node to guarantee reliability in addition to fault tolerance. Once a keyspace is created, the number of replicas needs to be determined.

### **3.4.3 Cassandra writing**

In order to understand the functions of the Cassandra structure, we state how the writing operation is performed in Cassandra. Cassandra's durability is accomplished with the aid of commit logs; therefore, if a writing operation is taking place, it will be quickly caught by the commit logs, which present a crash-recuperation instrument. A write operation will not be considered successful until it is written to the commit log. The advantage of a commit log becomes apparent when a write operation fails the in-memory store. However, it is still possible to recover that data. After the data is written to the commit log, it is written to the MemTable, which is designed in such way to flush the values to disk in a file called SSTable when the number of values stored in the MemTable reaches the threshold; then a new MemTable will be created. For each memTable, there is a bit flag to determine whether it needs flushing. To determine the number of copies for each piece of data in the system, we can simply set the replication factor to the required number of copies, which is not based on the

number of nodes in the cluster. The replication assists Cassandra to achieve high scalability and durability. Consistency in Cassandra comes with different levels that can be set. It is maintained by the quorum. The consistency level determines the number of replicas on which to write and which must succeed before returning an acknowledgment to the client application [20].

#### **3.4.4 Fault Tolerance Handling in Cassandra**

Cassandra shows high reliability due to the architecture of Apache Cassandra, which is a peer-to-peer relationship (as shown in Figure 15). The data in Cassandra is distributed equally between the nodes across the cluster. The nodes within the cluster are similar to each other in terms of the functions that they perform, so if we have a specific number of nodes (for instance X nodes), the data will be distributed throughout all these nodes. The system in charge of choosing a leader of the group is known as the zookeeper. After a leader is selected, the X – 1 node will receive the keys. In case one of the nodes goes down, different nodes inside the cluster will receive the information by having a place with the fizzled nodes. At the point when the fizzled node returns to the ring again, the information having with it a place will be moved back to it from an alternate node. In the event that the leader node goes down, another node will be selected as leader. Purposes behind disappointment may incorporate equipment bugs, control, cuts or common calamities.

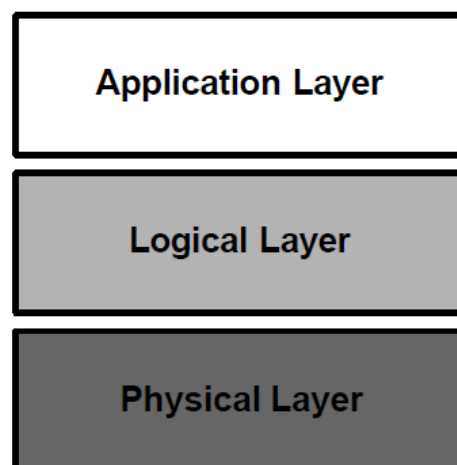
#### **3.5 MySQL**

MySQL is a well-known RDBMS owned by the Oracle Corporation. Its people group adaptation is open source and it can continue to run on a wide assortment of working systems. MySQL stores every item of data on one machine and uses B-Tree for indexing. It was initially created by the Swedish organization MySQL AB and

now it is own by the Oracle Corporation [21]. InnoDB is a productive stockpiling motor which is outlined to give higher execution vast measure of database. InnoDB's approach to sorting data on the circle is productive for serving any questions which channels utilizing essential key. Therefore, selecting an essential key which is utilized as a part of most questions will help to accomplish higher execution. MySQL gives the greater part of the elements that accompanies a RDBMS database. MySQL has a specific correspondence convention which is utilized for verification, questioning and dealing with the server utilizing a subset of the standard Structured Query Language (SQL) orders. Customer libraries with different libraries which actualize the convention are composed for JDBC (Java Database Connectivity) and for the .NET stages. MySQL gives APIs to the C, C++, Eiffel, Java, Perl, PHP and Python dialects. Furthermore, OLE DB and ODBC suppliers can likewise interface with MySQL in the Microsoft environment.

### 3.5.1 MySQL Architecture

In general, most relational databases share the same basic architecture, comprising three components, as shown in the Figure 16:



**Figure 16** RDBM layers



MySQL also shares this feature with other RDBM. It has three layers as follows:

### **3.5.1.1 Application Layer**

This layer is used to enable MySQL to interact with a client. It is composed of three elements: administrator, clients, and query users. The administrator uses several different administrative interfaces and utilities, including isamchk, mysqladmin, etc. The client also communicates with the MySQL RDBM using different interfaces such as MySQL APIs, while query users communicate with MySQL RDBM by using MySQL. We can conclude that MySQL is a query interface because it allows users to issue SQL statements; then it will show the results.

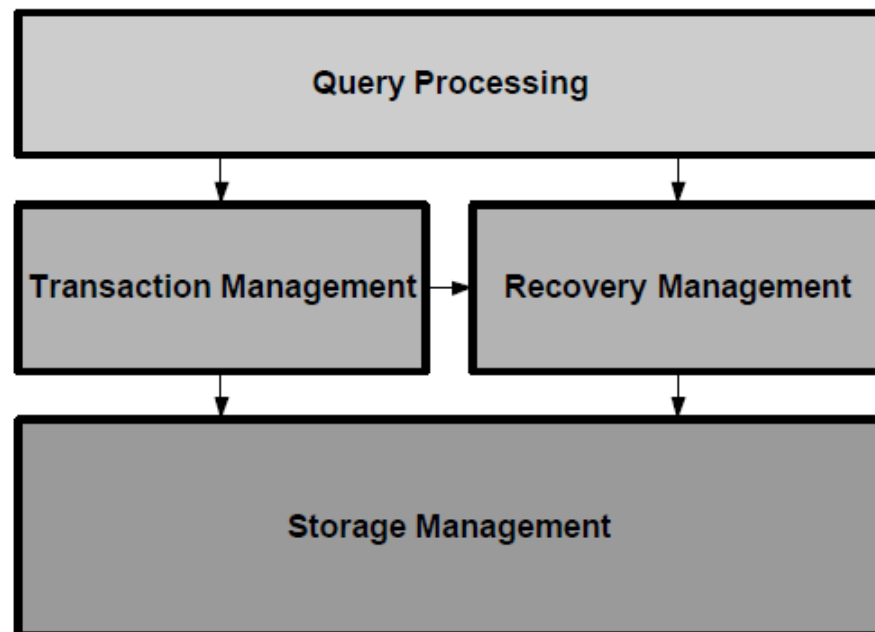
### **3.5.1.2 Logical Layer**

This layer consists of four subsystems: query processor, transaction management, recovery management, and storage management. All these subsystems work to handle requests issued to the MySQL database server. We highlight these systems as follows:

1. Query processor: when the user wants to view the implicit data within storages the number of interaction in the systems will increase. The query processor used to analyze and optimize these requests.
2. Transaction management: A transaction can be defined as a unit of work which contains one or several MySQL commands within it. The main purpose of the transaction manager is to guarantee that the transaction can be executed atomically and logged. In addition, it is also in charge of removing deadlock cases that might occur especially when two transactions are in progress and they need to process the same data. Another responsibility of the transaction manager is to issue the ROLLBACK SQL commands and

COMMIT commands. ROLLBACK commands take place when crashes occur while performing a transaction. On the other hand, the transaction cannot be considered completed until the COMMIT command executes a transaction.

3. Recovery management: This subsystem consists of two parts: the log manager and the recovery manager. The log manager is responsible for logging all the operations that are performed in the database by keeping the log on disk by using the buffer manager. Each operation in the log is saved as a MySQL command. To bring the database back to the last stable state when a crash occurs, commands within the log are executed. The recovery manager is used to bring back the database to the last stable state. This can be achieved by logging the database that is fetched from the buffer manager followed by performing the operations within the log.



**Figure 17** Logical Layer Subsystems

### **3.5.1.3 Storage Management:**

Storage Management is composed of three elements: the storage manager, the buffer manager, and the resource manager.

1. **Storage manager:** The storage manager is used as an interface with the operating system. Its major function is to write data existing in user tables, logs, internal system data and indexes to disk efficiently.
2. **Buffer Manager:** The buffer manager is used to reserve the memory resources in order to view and manage the data. Based on formatted requests, the buffer memory makes a decision to evaluate the memory that should be allocated to each buffer and it calculates how many buffers are needed for the requests.
3. **Resource manager:** Once the execution engine sends requests, the resource manager accepts these requests, followed by putting them into a table called table requests. The second task is to request the tables that exist in the buffer manager. It is also used to pass data from the buffer manager to the upper layers.

### **3.5.2 Storage Engine**

There are several storage engines that come with the MySQL database system:

1. **InnoDB:** This is one of the storage engines that MySQL uses. It contains standard ACID-compliant transaction characteristics. These features are capable of performing several tasks, including commit, data recovery after crash, and rollback. For all the previously mentioned abilities of data managing, data in InnoDB considers protected data. The InnoDB engine is suitable for dealing with large volumes of data because it can be configured for maximum performance. We should mention that InnoDB in different

disciplines (for instance, in some famous sites such as Slashdot.org and the storing capacity of data in this site) has reached 1 TB. In 2005, InnoDB was owned by the Oracle Corporation [22]. In MySQL, to know whether InnoDB is supported, we can use the command:

```
mysql>show engines;
```

2. MyISAM: This engine is considered the default engine which was used in the previous release of MySQL, more specifically before the (5.5) version of MySQL. It supports many extensions. Tables in MyISAM are divided into three variant files on disk. Within these three files, there are useful descriptions of the table's format, data and information on indexes.
3. Tina: This storage engine does not support indexing and the format file used to store the data in it is CSV.
4. HEAP: This engine is in-memory data store and it cannot keep any data after rebooting.

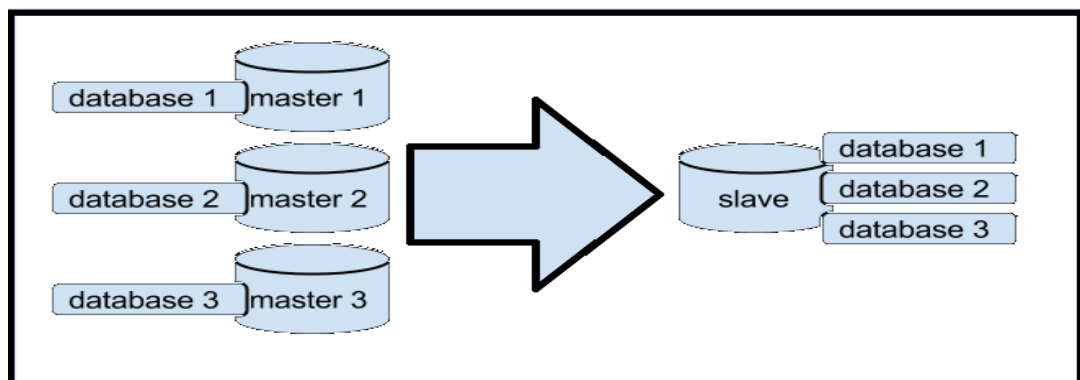
Although there are several engines dealing with MySQL, they share a number of features, as follows:

- Providing full table scans
- A look up through the index
- Scanning through all rows
- Dealing with text entries
- Searching in row levels
- Row updating
- Automatic incrementing
- The ability to perform unique index checks

### 3.5.3 MySQL Replication

The concept of replication means repeating data through a cluster of nodes as explained previously. MySQL, similar to other database systems, supports this concept. The master (MySQL database server) copies the data to the slaves. We should mention that the default replication in MySQL is asynchronous replication, which means slaves do not always need to be connected in order to obtain updates from the master. Based on the method used to configure the cluster, it is possible to replicate the database completely, or select one database from many in order to update it. Alternatively, we can even select a table in a database to update it. There are several advantages of replication, as follows:

1. Scale-out solutions: Several slaves work together to perform a task on loads, which also leads to an increase in the overall performance and the efficiency of the cluster.
2. Data-security: due to data replication within the cluster, and the ability of the slaves to maintain the replication process while running backup services without interrupting the master slaves.
3. Analytics: data analysis occurs within the slaves while the live data take place in the master, and not affecting performance.
4. Data distribution for long distances: local copies of data may be used for remote sites.



**Figure 18** Replication in MySQL

## CHAPTER 4

### TEST ENVIRONMENT

This chapter discusses the test environment of the used cluster in our study by focusing on the used nodes, their characteristics, the operating systems, and the benchmark tool used in this study with the Yahoo Cloud Serving Benchmark (YCSB-0.9.0), and the configuration of each database system. The authors in many studies used Mongo DB, Apache Cassandra, Apache HBase and apache couch dB, as NoSQL database systems. For relational database systems, the most used system is MySQL database. We preferred to choose systems from these most widely used systems in our study, so we chose Mongo DB, Apache Cassandra and MySQL database systems.

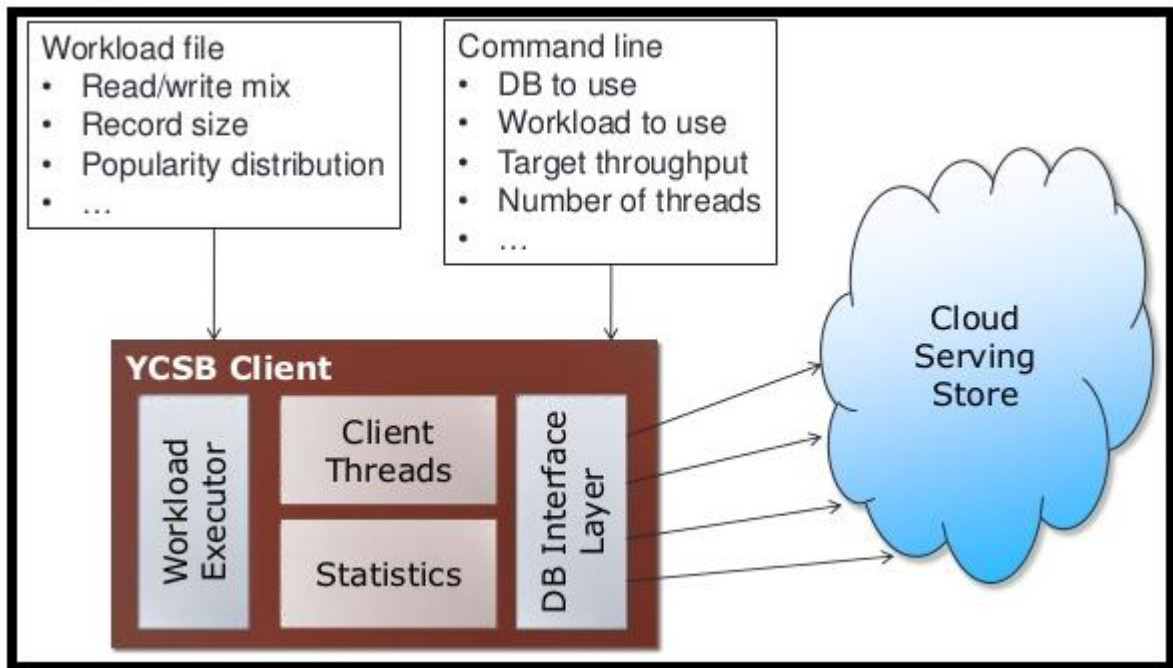
#### 4.1 Nodes features

The cluster is composed of four computers, each of which has the following specifications:

- 4 GB RAM
- Intel Core i3 processor
- 3.33 GHz processor speed
- Ephemeral storage in each unit.
- Ubuntu 14.04 LTS (64-bit)
- Java 8

## 4.2 Yahoo cloud serving benchmark

Performance evaluation were made less demanding by the promotion of the Yahoo Cloud Serving Benchmark (YCSB), proposed and executed by Cooper et al. [25]. Although there are several database benchmark tools such as Hammer DB, but still generally utilized today, permits testing the read/compose, inertness and versatility capacities of any database. The Yahoo! foundation created an open source tool in order to test database systems for both types of database, namely relational and non-relational database systems. There are several that have been released for this tool, and in our study, we used the latest version of Yahoo Cloud Serving Benchmark (version 0.9.0). We should mention that after we started our test on the cluster, another release of this tool was created by the Yahoo Foundation (i.e., 0.11.0). The YCSB is an open source benchmarking framework designed by Yahoo to compare the performance of distributed NoSQL data stores such as Cassandra, HBase, and PNUTS [23]. The first benchmark was designed in 2010 to simplify the performance comparisons of cloud data stores. YCSB performs several types of operations, such as creation, deletion, updating and reading. We used the last release of the YCSB, (0.9.0) available on GitHub, which does not support the last versions of many database management systems. The YCSB comes with six standard workloads which mix different scenarios, such as read, write, update, and search. These workloads give us a good rounded picture of the performance of the database systems. There is also a second workload which can be generated by the client to highlight another aspect not covered by the core workloads.

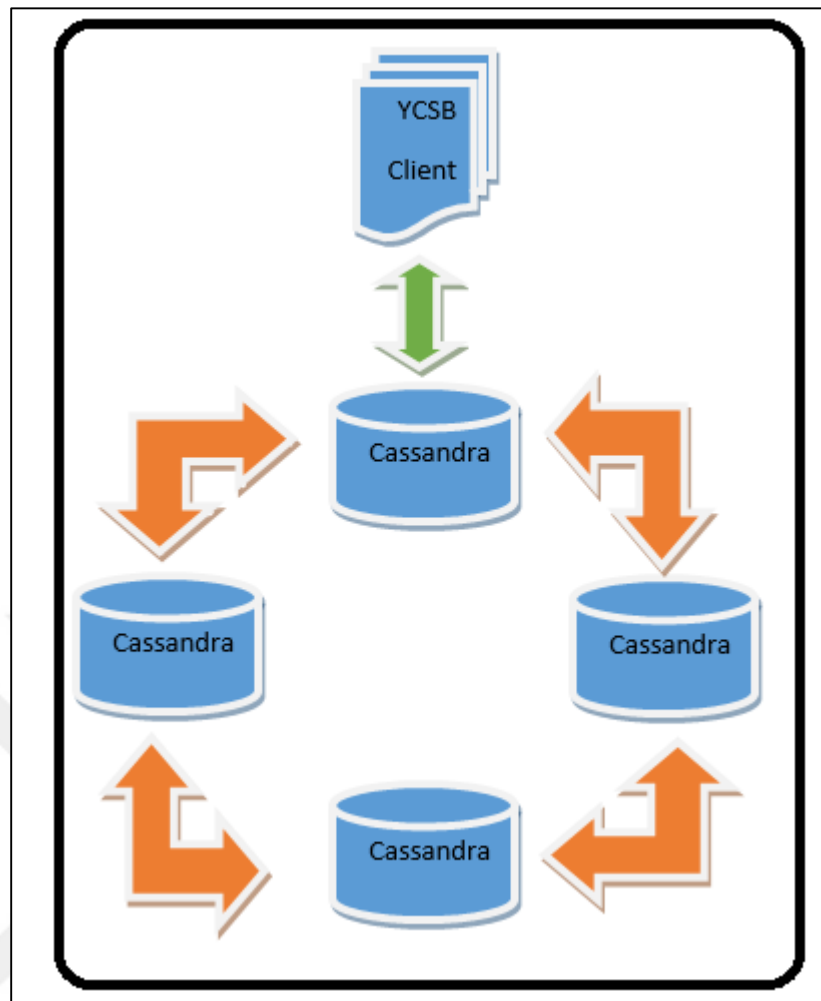


**Figure 19** YCSB Architecture

### 4.3 Apache Cassandra

The cluster of Apache Cassandra was set up as shown in Figure 20. The architecture of Cassandra is a peer-to-peer ring-shaped structure [24]. According to the abilities of the Yahoo Cloud Serving Benchmark (YCSB-0.9.0), we tested the last version of Apache Cassandra (3.7) that can be supported by YCSB. During the configuration progress, we followed some important steps: in (Cassandra.yaml) we created three folders (data, commit log, saved caches) and set their locations in Cassandra.yaml. We set the initial token value to 0. As mentioned previously, the Cassandra architecture is peer-to-peer and ring-shaped. Due to its structure, we selected two seeds for each local node as its neighbors, in addition to setting the listening address according to the local IP address of the nodes, we changed the simple snitch to Rack inferring snitch, and we enabled the broadcast-rpc-address.

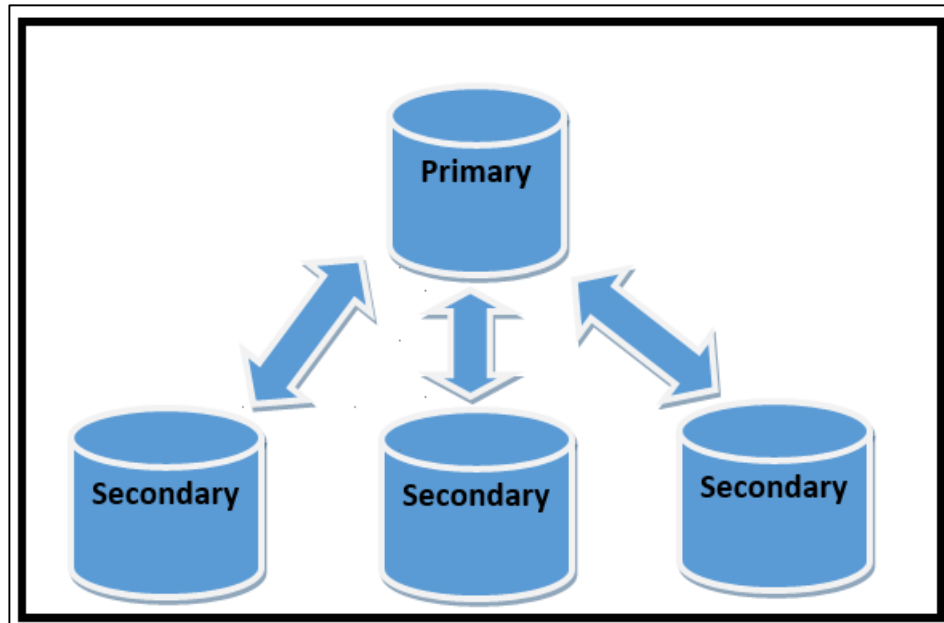




**Figure 20** Cassandra cluster

#### 4.4 Mongo DB

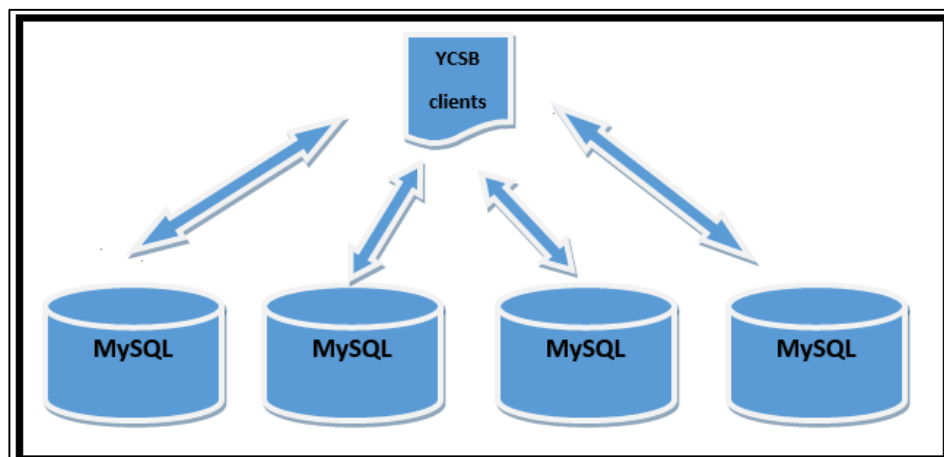
As mentioned previously, we used Mongo DB 3.2.7, which is the last release. Although we can configure the cluster in Mongo DB to enable the cluster to shard (Sharding) the data, we preferred the configure it to repeat the data (replication). In this case, the cluster comprises 4 nodes, three of which performed as secondary nodes, and the fourth as a primary node. The Figure 21 shows the architecture of the cluster.



**Figure 21** Mongo DB Cluster

#### 4.5 MySQL

We installed MySQL (5.5), which is compatible with YCSB (0.9.0). We did this in order to use the MyISAM engine with a 6-GB key buffer size. The MySQL cluster consisted of 4 nodes, with the data distributed evenly throughout all the nodes in the cluster. The Figure 22 shows the architecture of the cluster.



**Figure 22** MySQL Cluster

## 4.6 User Table

In each system, we created a user table according to the requirements of YSCB. The default of this table consisted of 10 fields in each record, the length of each field was 100. We created the user table in Mongo DB using the command:

```
db.createCollection ("ycsb");  
use ycsb;  
create usertable;
```

In Cassandra, we used:

Create keyspace ycsb

```
WITH REPLICATION = {'class': 'SimpleStrategy','replication factor': 3};
```

```
cqlsh> USE ycsb;
```

```
cqlsh> create table usertable (
```

```
  y_id varchar primary key,
```

```
  field0 varchar,
```

```
  field1 varchar,
```

```
  field2 varchar,
```

```
  field3 varchar,
```

```
  field4 varchar,
```

```
  field5 varchar,
```

```
  field6 varchar,
```

```
  field7 varchar,
```

```
  field8 varchar,
```

```
field9 varchar);
```

Whereas the command used in MySQL is:

```
Create table usertable
```

```
(
```

```
  ycsb_key varchar (100),
```

```
  field1 varchar (100), field2 varchar (100),
```

```
  field3 varchar (100), field4 varchar (100),
```

```
  field5 varchar (100), field6 varchar (100),
```

```
  field7 varchar (100), field8 varchar (100),
```

```
  field9 varchar (100), field10 varchar (100),
```

```
  } primary key (ycsb_key);
```

## CHAPTER 5

### RESULTS

We present the results of our study in this chapter. The results of the three tested database systems (MongoDB, Apache Cassandra, and MySQL) are illustrated in figures and tables, using the Yahoo Cloud Serving Benchmark (YCSB). The different specifications between relational database management systems (RDBM) and the NoSQL database systems have influenced the performance of the three systems. YCSB creates data to test performance. There are 6 inbuilt core workloads to simulate different operations related to database systems, including read, write and update operations. These workloads come in several scenarios:

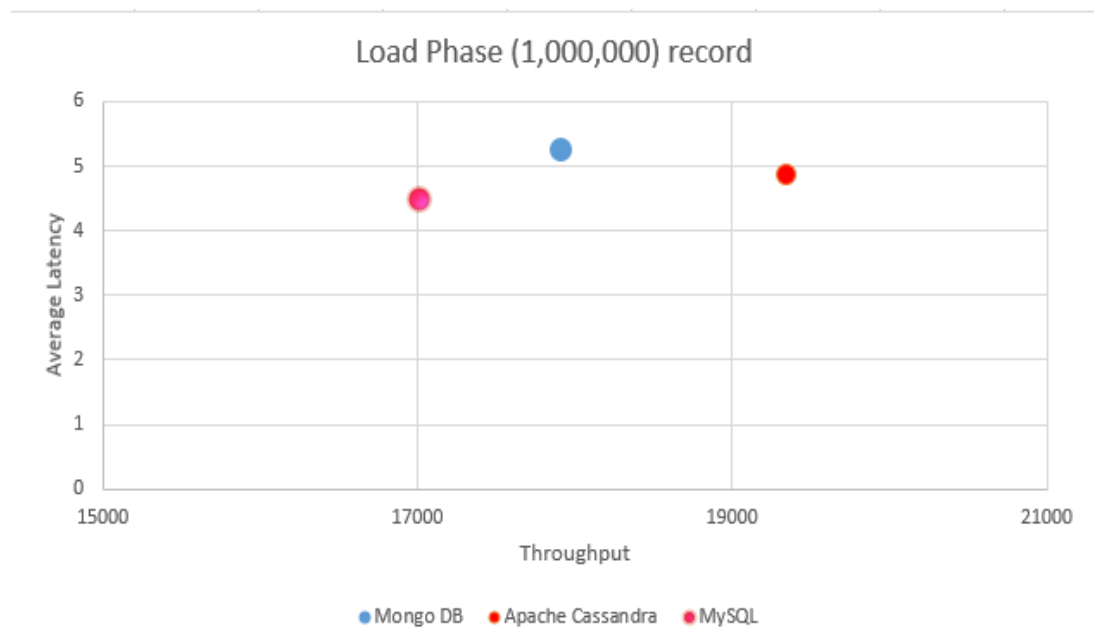
- Update heavy workload: This workload mixes 50% of read and 50% of write.
- Read mostly workload: This workload mixes 95% reads and the remainder for writing.
- Read only: In this workload, it is a 100% read.
- Read latest workload: In this workload, the last inserted records are more famous.
- Short range: In this workload, the query is for a short range of values instead of a particular record.
- Read-Modify-Write: The client in this workload reads a record, modifies it and finally writes it.

## 5.1 Load Phase

We loaded 1000, 000 records and into each system MySQL ranked in the last place in terms of throughput but the lowest average latency (the highest throughput and lowest average latency are considered the best performance). Due to the improved architecture of Apache Cassandra, it showed good performance with the highest throughput of over 19, 200 operations/second. This high performance is due to Cassandra's data update occurring in memory, while these data are simultaneously written to disk. Mongo DB ranked second in terms of throughput (see Table 1 and Figure 23) exhibited the values of load phase.

**Table 1.** Load Phase

Database	Throughput	Average latency
Mongo DB	17919.86237	5.222317
Apache Cassandra	19342.3452	4.84564
MySQL	17020.4563	4.45184

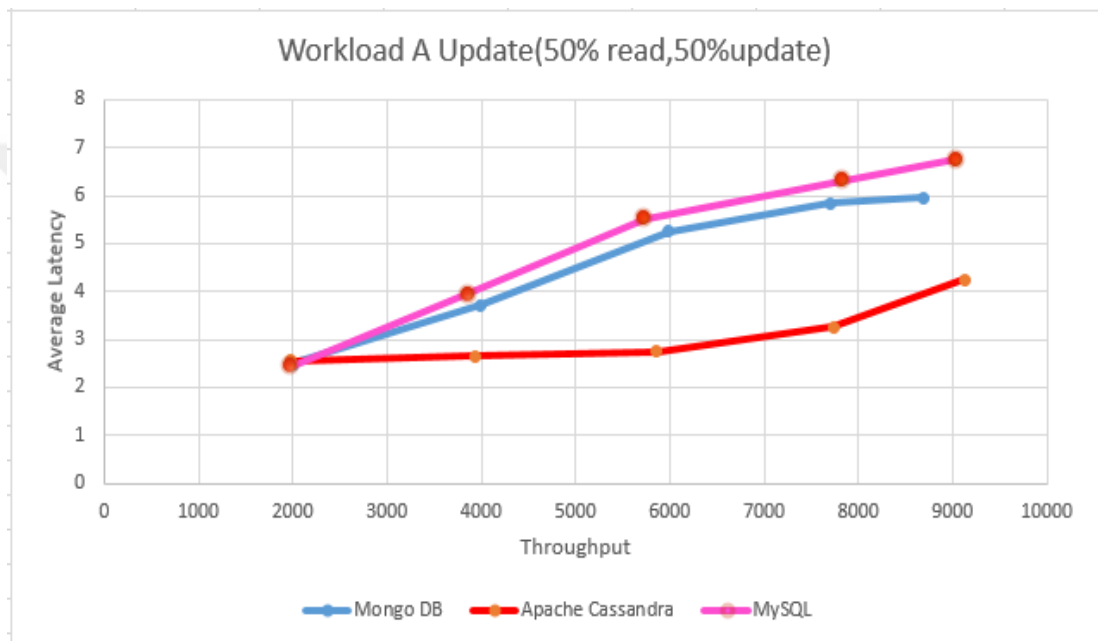


**Figure 23** Load Phase

## 5.2 Workload A (50% read, 50% update)

There are two scenarios in this workload, which we present to show the differences that occurred in the average latency for the read and update (write) operations:

- We can observe in Figure 24 that Cassandra performed well in comparison with the other two systems. Cassandra showed lower average latency, than both Mongo DB and MySQL. Tables 2, 3, 4 show values of workload A (update).



**Figure 24** Workload A 50%update

**Table 2.** Mongo DB update (W.A)

Throughput	Average Latency
1998.4252	2.501436
3994.3759	3.7142846
5988.848	5.245696
7704.5345	5.844159512
8692.4353	5.959048

**Table 3.** Cassandra update (W.A)

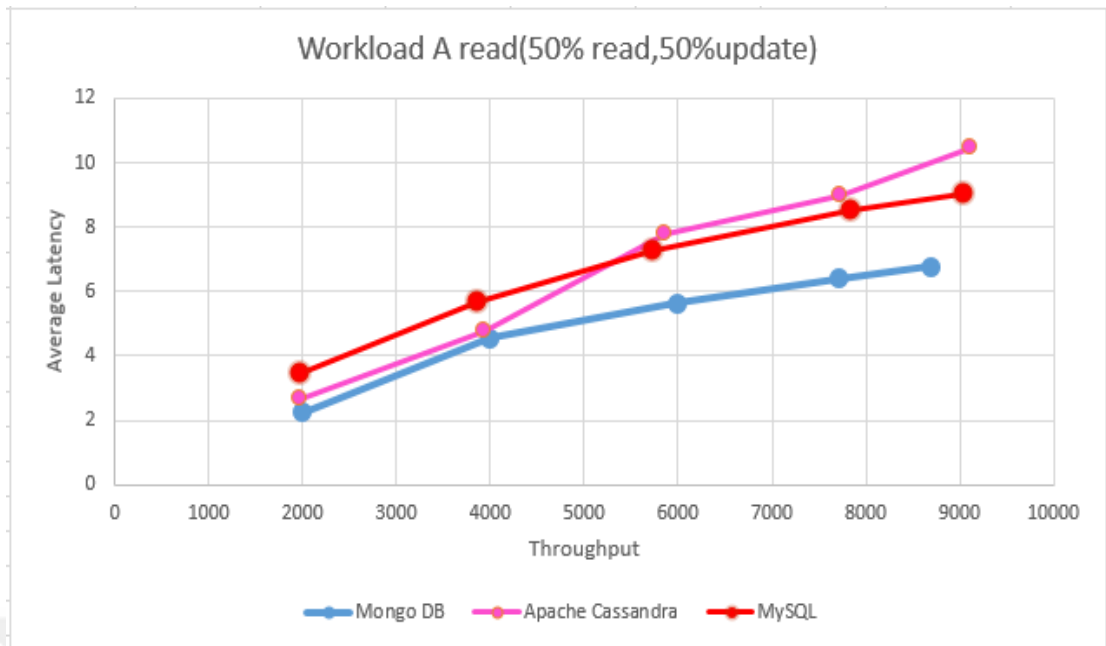
Throughput	Average Latency
1983.10788	2.5535343
3933.74	2.6455234
5855.31516	2.7498353
7732.87554	3.2523443
9125.2532	4.2431345

**Table4.** MySQL update (W.A)

Throughput	Average Latency
1982.3452	2.446246
3856.5682	3.94562
5732.9541	5.52346
7834.9183	6.32546
9045.3164	6.754356

- In Figure 25, we can see a clear influence of the read operation on the average latency. For instance, Mongo DB showed a very high performance and compared in terms of average latency, Apache Cassandra performed better than MySQL with the small number of throughput. However, with the increase of throughputs, we can see that MySQL showed lower average latency than Cassandra's average latency because MySQL makes good use of its key cache. Tables 5, 6, 7 show values of workload A (read).





**Figure 25** Workload-A 50% read

**Table 5.** Mongo DB Read (W.A)

Throughput	Average Latency
1998.4252	2.2505646
3994.3759	4.5572932
5988.848	5.6300574
7704.5345	6.41566
8692.4353	6.776194

**Table 6.** Cassandra Read (W.A)

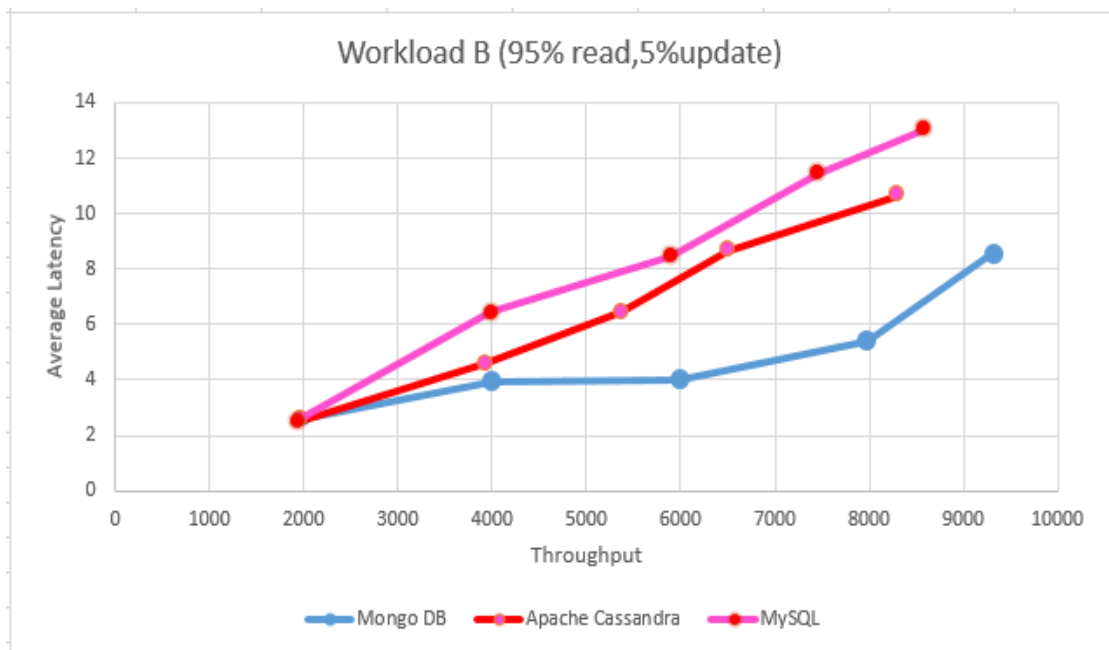
Throughput	Average Latency
1983.10788	2.64534
3933.74	4.789056
5855.31516	7.765043
7732.87554	8.967804
9125.2532	10.4535

**Table 7. MySQL Read (W.A)**

Throughput	Average Latency
1982.3452	3.44523
3856.5682	5.6546
5732.9541	7.2765
7834.9183	8.5434
9045.3164	9.05134

### 5.3 Workload B (95% Read, 5% Update)

Workload-B tested the ability of the read operation for different systems. In our study, MySQL showed poor performance (see Figure 26). Moreover, Cassandra did not perform very well while the reading process was occurring due to its key-row caching. MongoDB performed the best with lowest latency, which increased with the increase of the target throughput. This was due to its support of memory mapped caching. Tables 8, 9, 10 show values of workload B.



**Figure 26** Workload-B (mostly read)

**Table 8.** Mongo DB 95% Read (W.B)

Throughput	Average Latency
1998.7367	2.61511
3995.17383	3.9388724
5987.4503	4.018284
7977.3443	5.4060915
9320.5331	8.57913459

**Table 9.** Cassandra 95% Read (W.B)

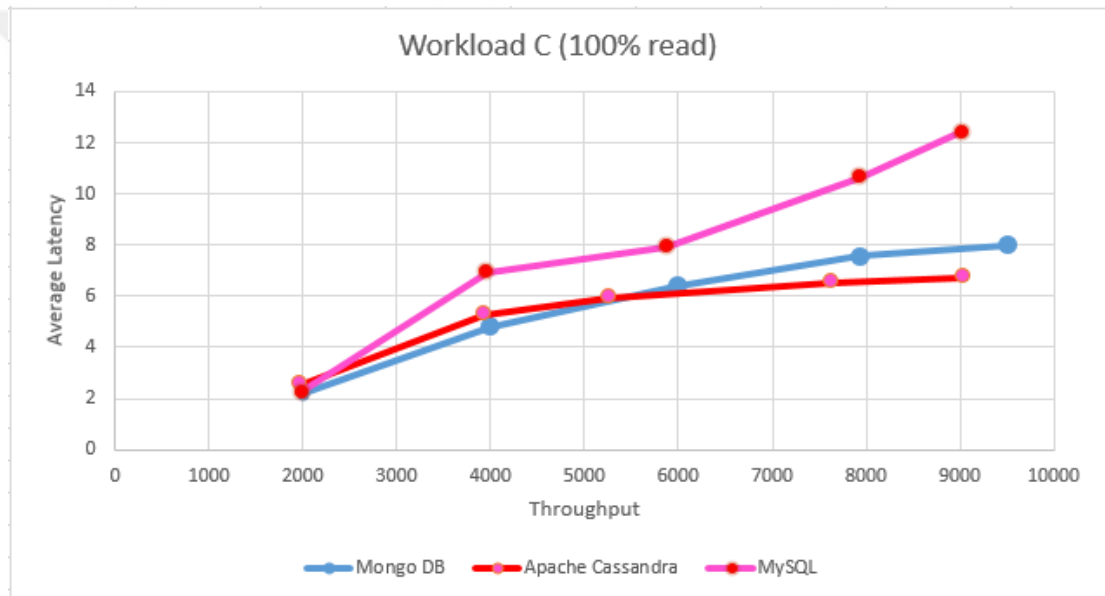
Throughput	Average Latency
1983.39108	2.5646
3933.9257	4.5643
5375.1881	6.45367
6506.87	8.65479
8304.534	10.643654

**Table 10.** MySQL 95% Read (W.B)

Throughput	Average Latency
1934.9865	2.53461
3986.654	6.42345
5894.543	8.463451
7454.738	11.45436
8584.564	13.05134

## 5.4 Workload C

By running this workload (100% read), we can conclude from the results illustrated in Figure 27 that Mongo DB, with its low throughput, exhibited a better performance in comparison with Cassandra and MySQL. However, at some point, Cassandra showed better performance than Mongo DB in terms of average latency even though the throughput in Mongo DB was slightly higher than in Cassandra. MySQL was reasonable with its low throughput, but once the average latency increased with the growth of throughput, MySQL showed poor performance. Tables 11, 12, 13 show values of workload C.



**Figure 27** Workload C (Read only)

**Table 11.** Mongo DB 100% Read (W.C)

Throughput	Average Latency
1998.6529	2.178755
3995.381	4.801241
5989.4944	6.392678
7941.0456	7.5707191
9508.32453	7.992707

**Table 12.** Cassandra 100% Read (W.C)

Throughput	Average Latency
1982.9191	2.5658656
3933.9257	5.27629
5262.6039	5.9260831
7647.5491	6.55022
9046.6543	6.732

**Table 13.** MySQL 100% Read (W.C)

Throughput	Average Latency
1992.432	2.2432
3958.345	6.95325
5873.564	7.9345
7935.654	10.6453
9023.546	12.4345

## CHAPTER 6

### CONCLUSION

In our study, we presented the comparison results of two NoSQL databases (Mongo DB and Apache Cassandra) and one relational database system (RDBM), namely MySQL. In order to answer the questions of the study as mentioned in the first chapter (see part 1.3), we created a test environment that is explained in Chapter 4.

To answer the first question, it is safe to say that generally that we have noticed that NoSQL database systems performed better than the MySQL relational database management system. However, as a limitation of the study, we should mention an important thing, that according to the limitations of the used version of Yahoo Cloud Serving Benchmark (YCSB) which is version (0.9.0) (now there is a newer release 0.11.0), the last supported version of MySQL is version (5.5) and in the present time the latest version is (8.0).

The test results showed that in some cases, Mongo DB has better results than Apache Cassandra. For instance, Mongo DB has the lowest average latency while carrying out a reading operation. MySQL shows the highest average latency. However, it has the highest throughput during a read operation. In other cases, (see Figure 27), we see that Cassandra performed better than Mongo DB and MySQL, and we noticed that with high throughput, Cassandra has the lowest average latency between the other two databases and that is the answer of the second question.

We found out that both the foundations of Mongo DB and Apache Cassandra improved their products comparing with the study in [5], which did not show any great difference between the MySQL database and the other NoSQL database used in our study and that is the answer of the third question. Mostly during the read operation, we noticed that Mongo DB took the lead with low throughput in contrast with the writing operation, when Cassandra took the lead. We can say that there is no

linear relationship that describes the performance of the databases. However, we can conclude that for every database, be they relational or non-relational databases, there are strong and weak sides in terms of performance. Cassandra has a powerful design that influences the writing process to give high throughput with average latency. Due to the design of the NoSQL database which gives advantages to the database, such as overcoming the scalability limitation compared with the relational database, and according to the results that we achieved, we can say that NoSQL performed better than (RDBM) in this study due to the ability to overcome the scalability limitations in (RDBM) in addition to the performance. We cannot ignore the powerful features of (RDBM), such as the ability store structured data in tables, which enables the user to access data without the need to know how the data are structured. Moreover, (RDBM) uses SQL (structured query language) for defining and manipulating the data, which is very powerful. In addition, databases are a best fit for heavy duty transactional type applications.

## REFERENCES

- [1] “SQL vs NoSQL: The Differences” retrieved from: <https://www.sitepoint.com/sql-vs-nosql-differences> [accessed on 2 August 2016].
- [2] “NoSQL databases,” [Online]. Available: [nosql-database.org](http://nosql-database.org) [Accessed 10 2 2016].
- [3] S. Harizopoulos, D.J. Abadi, S. Madden, and M. Stonebraker. Oltp through the looking glass, and what we found there. In Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, pp. 981-992. ACM, 2008.
- [4] F. Chang, J. Dean, S. Ghemawat, W.C. Hsieh, D.A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R.E. Gruber. Bigtable: A distributed storage system for structured data. ACM Transactions on Computer Systems (TOCS), 26(2): 1–26, 2008.
- [5] Christopher Jay Choi. 2014: “A Study and Comparison of Nosql Databases, California State University, Northridge”.
- [6] Apache Cassandra NoSQL Performance Benchmarks, Retrieved from <http://www.planetcassandra.org/nosql-performance-benchmarks/> accessed on August 27, 2016.
- [7] RDBMS vs NoSQL: Performance and Scaling Comparison, retrieved from <http://docplayer.net/6379950-Rdbms-vs-nosql-performance-and-scaling-comparison.html>. On August 3, 2016.
- [8] Benchmarking Top NoSQL Databases, Apache Cassandra, CouchBase, HBase, and MongoDB, retrieved from <http://www.endpoint.com/> on June, 10, 2016.



- [9] Hecht, R. and Jablinski, S. 2011. "NoSQL Evaluation A Use Case Oriented Survey". Proceedings International Conference on Cloud and Service Computing, pp. 12-14.
- [10] Lith A, Mattson J (2013) Investigating storage solutions for large data: A comparison of well performing and scalable.
- [11] "Big data for dummies", Dr. Fern Halper, Marcia Kaufman, Judith Hurwitz, Alan Nugent 2013.
- [12] "Challenges and Opportunities with Big Data". CRA.org. Retrieved June 2016.
- [13] Jeffrey Dean, Sanjay Ghemawat, 2004, MapReduce: Simplified Data Processing on Large Clusters, OSDI'04: Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation Volume 6.
- [14] Gilbert, Seth and Nancy Lynch: Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. SIGACT News, 33(2):51\_59, 2002.
- [15] 10gen.com: Querying - MongoDB.  
<http://www.mongodb.org/display/DOCS/Querying>, 2009. [Online; accessed 14-June-2016].
- [16] Mongo DB Connector for Business Intelligence Retrieved from <https://docs.mongodb.org/bi-connector> on August, 17, 2016.
- [17] Mongo DB Inc. retrieved from <https://docs.mongodb.org/manual/replication/> on / (Accessed: September, 25, 2016).
- [18] Mongo DB Inc. retrieved from <https://docs.mongodb.org/manual/core/sharding-introduction/> on / (Accessed: September, 2, 2016).
- [19] "Facebook's Cassandra paper, annotated and compared to Apache Cassandra 2.0". DataStax.com. Retrieved April 2014.

- [20] Teddyma, Learn Cassandra, retrieved from [https://teddyma.gitbooks.io/learncassandra/content/replication/turnable\\_consistency.html](https://teddyma.gitbooks.io/learncassandra/content/replication/turnable_consistency.html) on September, 12, 2016.
- [21] “Sun Microsystems Announces Completion of MySQL Acquisition; Paves Way for Secure, Open Source Platform to Power the Network Economy”. Sun Microsystems. 26 February 2008.
- [22] “Oracle Announces the Acquisition of Open Source Software Company, Innobase”. Oracle. Retrieved 2012-01-30.
- [23] Cooper, B. F., Ramakrishnan, R., Srivastava, U., Silberstein, A., Bohannon, P., Jacobsen, H., Puz, N., Weaver, D., and Yerneni, R. PNUTS: py.
- [24] Kuldeep Singh. 2015. Survey of NoSQL Database Engines for Big Data, Aalto University
- [25] Cooper BF, Silberstein A, Tam E, Ramakrishnan R, Sears R (2010). Benchmarking cloud serving systems with ycsb In: Proceedings of the 1st ACM Symposium on Cloud Computing, 143–154. ACM, Indianapolis, Indiana, USA.

## APPENDIX

### CURRICULUM VITAE

#### PERSONAL INFORMATION

**Surname, Name:** Ihsan Ahmed Taha

**Nationality:** Iraqi

**Date and Place of Birth:** 25 Mar.1989, Salah El-Den, Iraq

**Marital status:** Married

**Phone:** 009647722203613

**E-mail:** [hololh@yahoo.com](mailto:hololh@yahoo.com)



#### EDUCATION

Degree	Institution	Year of Graduation
M.Sc.	Çankaya University Mathematics and Computer Science	2017
B.Sc.	University of Tikrit	2013
High School	Al-Nedaa School	2006

#### WORK EXPERIENCE

Year	Place	Enrollment
2013- Present	Al-Esraa university	Teaching Assistant

#### FOREIGN LANGUAGES

English