



**SECURITY AND PERFORMANCE COMPARISON OF NOSQL DATABASE
SYSTEMS**



MUSTAFA MUSLIH SHWAYSH

JULY 2018

**SECURITY AND PERFORMANCE COMPARISON OF NOSQL DATABASE
SYSTEMS**

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



**BY
MUSTAFA MUSLIH SHWAYSH**

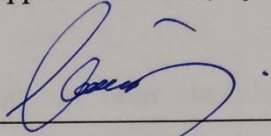
**IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF
MASTER OF SCIENCE
IN
THE DEPARTMENT OF
COMPUTER ENGINEERING,
INFORMATION TECHNOLOGY PROGRAM**

JULY 2018

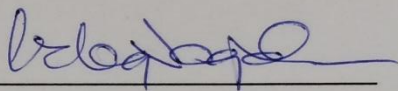
Title of Thesis: **SECURITY AND PERFORMANCE COMPARISON OF NOSQL DATABASE SYSTEMS.**

Submitted by **Mustafa Muslih Shwaysh**

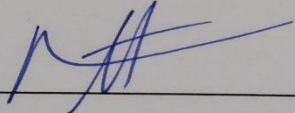
Approval of the Graduate School of Natural and Applied Sciences, Çankaya University.


Prof. Dr. Can ÇOĞUN
Director

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


Prof. Dr. Erdoğan DOĞDU
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.


Dr. Instructor Murat SARAN
Supervisor

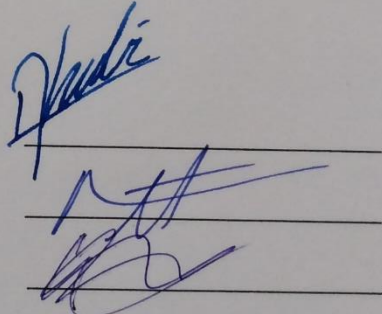
Examination Date: 06.07.2018

Examining Committee Members

Dr. Instructor Abdül Kadir GÖRÜR (Çankaya Univ.)

Dr. Instructor Murat SARAN (Çankaya Univ.)

Assoc. Prof. Dr. İ. Tolga MEDENİ (Ankara Yıldırım Beyazıt Univ.)

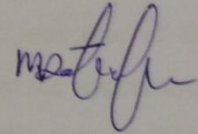


STATEMENT OF NON-PLAGIARISM

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 : Mustafa SHWAYSH

Signature :



Date

: 06.07.2018

ABSTRACT

SECURITY AND PERFORMANCE COMPARISON OF NOSQL DATABASE SYSTEMS

SHWAYSH, Mustafa Muslih Shwaysh

M.Sc., Department of Computer Engineering, Information Technology Program

Supervisor: Dr. Instructor Murat SARAN

July 2018, 93 Pages

The advent of Big Data has led the development of several key technologies and platforms to answer the ever-growing nowadays data. NoSQL was one of these technologies. Although NoSQL databases began to develop before the appearance of big data, the adoption of NoSQL databases did not occur until big data created the need. However, many versions of NoSQL databases appeared each with different architectural design, its query language, and its own set of solutions for scalability, compression, security, clustering, which render it to become difficult to decide which NoSQL database provides the solution to a given problem or issue. Therefore, studies should be dedicated to testing these databases concerning several factors such as query processing speed, security of data, and its readiness to be used in the scalable environment. In this study, we propose an investigation in two-folds: first, examining the performance of two NoSQL databases, MongoDB (3.6.3) and Cassandra (3.11.1) in single node and multi-node (cluster) configurations using the recent version of Yahoo Cloud Serving Benchmark (YCSB-0.12.0). 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. This study will help IT decision makers to determine which database is better according to its deployment requirements. The

second fold of this study is to investigate the security of both databases. The first step is a comparative study of both databases' security features according to ten selected features from the literature. The second step of our security investigation is data encryption overhead. Overhead is the time spent by the database engine to encrypt all incoming data streams plus the time spent to decrypt the data to answer queries. The results of this study have shown that the performance of each system differs due to the differences in its respective data storing mechanisms. The results have also revealed that the performance of MongoDB is better than Cassandra in a single node test, whereas in proposed multi-node test Cassandra performed better in one workload and took the lead in two other workloads when testing bigger record size. Security comparative investigation has shown that both databases have improved significantly concerning previous studies. However, Cassandra took the lead as it has better supported and the implementation of the selected security features is presented in this study. Lastly, MongoDB encryption performance regarding runtime and throughput was between nearly 2x and 2.5x faster than Cassandra which makes it more suitable to be used in the environments when encryption is a requirement.

Keywords: NoSQL, database, performance, security, comparison

ÖZ

NOSQL VERİ TABANI SİSTEMLERİNİN GÜVENLİK VE PERFORMANS KARŞILAŞTIRMASI

SHWAYSH, Mustafa Muslih Shwaysh

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü, Bilgi Teknolojileri Programı

Danışman: Dr. Öğretim Üyesi Murat SARAN

Temmuz 2018, 93 sayfa

Günümüzde giderek büyüyen verilerin ortaya çıkardığı ihtiyaçlara cevap vermek için çeşitli temel teknolojiler ve platformlar geliştirilmektedir. NoSQL bu teknolojilerden birisidir. NoSQL veri tabanları büyük verilerin ortaya çıkmasından önce gelişmeye başlamış olsa da büyük verilerin saklanmasına ve işlenmesine ihtiyaç duyulana kadar NoSQL veri tabanlarının benimsenmesi gerçekleşmemiştir. Bununla birlikte, NoSQL veri tabanlarının her biri farklı mimari tasarıma, sorgulama diline ve ölçeklenebilirlik, sıkıştırma, güvenlik, kümelenme için kendi çözümlerine sahiptir ve bu da hangi NoSQL veri tabanının mevcut ihtiyaca bir çözüm sunacağına karar vermeyi zorlaştırmaktadır. Bu nedenle, NoSQL veri tabanlarını sorgulama hızı, veri güvenliği ve ölçeklenebilir ortamlarda kullanılmaya hazır olma gibi çeşitli faktörlerle test etmeye yönelik çalışmalar yapılmalıdır. Bu çalışmada iki aşamalı bir araştırma yapılmıştır: Birinci aşamada, iki farklı NoSQL veri tabanının, MongoDB (3.6.3) ve Cassandra'nın (3.11.1), Yahoo Cloud Hizmet Ölçümü (YCSB-0.12.0 sürümü) kullanarak performansını tek bir düğümde ve çok düğümlü (küme) konfigürasyonlarda inceleyerek karşılaştırılmıştır. Bu aşamada, farklı iş yükleri için bir test ortamı kurulmuş ve çalışmada kullanılan her bir veri tabanı yönetim sisteminin yanıtları her iş yükü için incelenmiştir. Bu çalışma, özellikle karar vericilerin, hangi veri tabanının gereksinimlerine göre daha iyi olduğunu

belirlenmesine yardımcı olacaktır. Bu çalışmanın ikinci katmanı, her iki veri tabanının güvenliğini araştırmaktır. İlk adım, her iki veri tabanının literatürdeki on seçilmiş özelliğe göre güvenlik özelliklerinin karşılaştırmalı bir çalışmasıdır. Güvenlik incelememizin ikinci adımı, veri şifreleme yükünün karşılaştırılmasıdır. Veri şifreleme yükü, veri tabanı motorunun tüm gelen veri akışlarını şifrelemek için harcadığı süreyi ve sorguları yanıtlamak için verilerin şifresini çözmek için harcanan toplam süredir. Bu çalışmanın sonuçları, her bir sistemin performansının, veri saklama mekanizmalarındaki farklılıklar nedeniyle farklı olduğunu göstermiştir. Sonuçlar, MongoDB'nin performansının tek düğümlü bir testte Cassandra'dan daha iyi olduğunu göstermiştir. Bununla birlikte, çok düğümlü testte Cassandra'nın özellikle büyük kayıtlarda daha iyi performans gösterdiği ortaya çıkmıştır. Güvenlik karşılaştırması araştırması, her iki veri tabanının da önceki çalışmalarda ortaya çıkan sonuçlara göre önemli ölçüde iyileştiğini göstermiştir. Ancak Cassandra, daha iyi güvenlik özelliklerine sahip olarak öne çıkmaktadır. Son olarak, MongoDB şifreleme performansı Cassandra'dan yaklaşık 2x ve 2,5 kat daha hızlı olduğu ortaya çıkmıştır, bu da şifrelemenin bir gereklilik olduğu ortamlarda MongoDB'nin kullanılmasının daha uygun olduğunu göstermektedir.

Anahtar Kelimeler: NoSQL, veri tabanı, performans, güvenlik, karşılaştırma

ACKNOWLEDGEMENTS

My greatest gratitude and warmest thanks are due to my supervisor Dr. Murat SARAN of the computer Engineering Department at Çankaya University. His understanding, encouraging and personal guidance have provided a great basis for this thesis. His ideas and constructive notes have had a remarkable influence on carrying out this thesis. I also wish to express my gratitude to my advisor Dr. Tolga PUSATLI for his help, advice and useful profound discussions.

I must not forget to record my deep gratitude to the members of my family for their endless patience and continuous support during the period of my study. Special thanks are due to my parents for their generous support, love and understanding. I owe a great debt of thanks to my brothers and sister who always encourage me to complete my study. Thanks are also due to my teachers and friends.

TABLE OF CONTENTS

STATEMENT OF NON-PLAGIARISM	iii
ABSTRACT	iv
ÖZ	vi
ACKNOWLEDGEMENTS	viii
TABLE OF CONTENTS	ix
LIST OF FIGURES	xiv
LIST OF TABLES	xvi
LIST OF ABBREVIATIONS	xvii

CHAPTERS:

1. INTRODUCTION	1
1.1. Aim of the study	2
1.2. Significance of the study	4
1.3. Research questions	4
1.4. Related work	5
1.4.1. NoSQL Databases Performance	5
1.4.1.1. NoSQL databases vs Relational Databases	5
1.4.1.2. NoSQL vs NoSQL databases	7
1.4.2. NoSQL Databases Security	11
1.5. Thesis structure	12
2. THEORETICAL BACKGROUND	13
2.1. NoSQL Data Stores types	13
2.2. Scaling	14
2.2.1. Sharding	15
2.2.2. Replication	17
2.3. MapReduce	19

2.4. NoSQL data store foundation	20
2.4.1. (CAP) Theorem	20
2.4.2. BASE	22
2.4.3. The Model of Consistency	22
2.5. Security Issues in NoSQL.....	23
2.6. Security Threats in NoSQL	25
2.6.1. Distributed Environment.....	26
2.6.2. Authentication	26
2.6.3. Safeguarding Integrity.....	27
2.6.4. Fine-Grained Authorization and Access control.....	27
2.6.5. Protection of Data at Rest and in Motion.....	28
2.6.6. Privacy of User Data	28
3. NOSQL DATABASE MANAGEMENT SYSTEMS USED IN THE STUDY	30
3.1. MongoDB	30
3.1.1. MongoDB Architecture.....	30
3.1.2. MongoDB Data Model.....	31
3.1.3. MongoDB Query Model	32
3.1.4. CRUD in MongoDB	33
3.1.5. Mongo DB Aggregation.....	34
3.1.6. MongoDB Features	34
3.1.7. Replication in MongoDB	35
3.1.8. Sharding in MongoDB	36
3.1.9. Failure Handling in MongoDB	38
3.2. Apache Cassandra.....	38
3.2.1. Apache Cassandra Architecture	39
3.2.1.1. Key Space	39
3.2.1.2. Commit Logs, Memtables and SSTables	40
3.2.1.3. Hinted Handoff	41
3.2.1.4. Compaction.....	41
3.2.1.5. Bloom Filter.....	41
3.2.1.6. Staged Event-Driven Architecture (SEDA)	41

3.2.2. Data Model.....	42
3.2.3. Apache Cassandra Features.....	43
3.2.4. Fault Tolerance in Cassandra.....	44
3.3. Security in NoSQL Databases	44
3.3.1. Cluster Security in NoSQL	44
3.4. Authentication.....	45
3.4.1. Authentication in MongoDB.....	46
3.4.2. Authentication in Apache Cassandra	47
3.5. Authorization	47
3.5.1. Authorization in MongoDB	47
3.5.2. Authorization in Apache Cassandra.....	47
3.6. Auditing	48
3.6.1. Auditing in MongoDB	48
3.6.2. Auditing in Apache Cassandra.....	49
3.7. Transport Encryption	49
3.7.1. Transport Encryption in MongoDB	49
3.7.2. Transport Encryption in Apache Cassandra.....	49
3.8. Encryption at Rest.....	50
3.8.1. Encryption at Rest in MongoDB.....	50
3.8.2. Encryption at Rest in Apache Casssandra.....	51
3.9. JMX Authentication.....	51
3.10. Authentication Caching	52
3.11. Proxy Roles.....	52
3.12. Node-to-Node encryption	53
3.13. Client-to-Node encryption	53
4. TEST ENVIRONMENT	55
4.1. Hardware Features	55
4.2. Software Features	55
4.2.1. Yahoo Cloud Serving Benchmark (YCSB)	55
4.2.2. Mongo Database (Open Source)	57
4.2.2.1. MongoDB Cluster Installation.....	58
4.2.3. Cassandra Database (Open Source)	61

4.2.3.1. Cassandra Cluster Installation	62
4.3. NoSQL Databases Security	63
4.4. MongoDB Enterprise	64
4.5. MongoDB Enterprise Encryption	64
4.6. Cassandra Enterprise	65
4.7. Cassandra Enterprise Encryption	65
5. RESULTS	66
5.1. Single Node Test	67
5.1.1. Workload A	67
5.1.1.1. Workload A Findings	68
5.1.2. Workload B	69
5.1.2.1. Workload B Findings	70
5.1.3. Workload E	71
5.1.3.1. Workload E Findings	72
5.1.4. Workload F	73
5.1.4.1. Workload F Findings	74
5.2. Multi Node Test (Cluster)	75
5.2.1. Workload A	75
5.2.1.1. Workload A Findings	77
5.2.2. Workload B	77
5.2.2.1. Workload B Findings	78
5.2.3. Workload E	79
5.2.3.1. Workload E Findings	80
5.2.4. Workload F	81
5.2.4.1. Workload F Findings	82
5.3. NoSQL Security	83
5.3.1. Security Features Findings	83
5.3.2. Cassandra:	85
5.3.3. MongoDB:	86
5.4. Encryption at Rest Benchmark	86
5.4.1. Encryption Overhead Test Findings	90

6. CONCLUSION	91
6.1. Suggestions for Future Study.....	93
REFERENCES	R1



LIST OF FIGURES

FIGURES

Figure 1.1. Databases performance compression	6
Figure 1.2. Overall average performance of three workloads	7
Figure 1.3. Throughput test results	8
Figure 1.4. Workloads test time results	9
Figure 1.5. Small and Large workload test performance results	10
Figure 1.6. Scalability performance results	11
Figure 2.1. Scaling-up versus Scaling-out	15
Figure 2.2. Sharding	16
Figure 2.3. Replication	17
Figure 2.4. Master-Slave Replication.....	18
Figure 2.5. Peer-to-Peer Replication	19
Figure 2.6. MapReduce functions for word counting	20
Figure 2.7. CAP THEOREM	21
Figure 2.8. Security threats in NoSQL databases.....	25
Figure 2.9. Security Threats with Distributed environments	26
Figure 3.1. MongoDB JSON Document	32
Figure 3.2. The BI connector in MongoDB	35
Figure 3.3. Replication in MongoDB	36
Figure 3.4. Sharding mechanism in MongoDB	38
Figure 3.5. Writing operation in Apache Cassandra	40
Figure 3.6. Data Types in Cassandra.....	43
Figure 3.7. Read/Repair node procedures in Cassandra.....	43
Figure 3.8. NoSQL Security Model	45
Figure 3.9. Proxy Roles in Cassandra	53
Figure 4.1. YCSB Architecture	56
Figure 4.2. MongoDB Cluster Design	59

Figure 4.3. MongoDB Cluster Design Overview	59
Figure 4.4. Cassandra Cluster Design Overview	63
Figure 5.1. Workload A Test Performances Graph.....	68
Figure 5.2. Workload B Test Performances Graph.....	70
Figure 5.3. Workload E Test Performances Graph	72
Figure 5.4. Workload F Test Performances Graph	74
Figure 5.5. Workload A Test Performances Graph.....	76
Figure 5.6. Workload B Test Performances Graph.....	78
Figure 5.7. Workload E Test Performances Graph	80
Figure 5.8. Workload F Test Performances Graph	82
Figure 5.9. MongoDB Overhead Performances Graph.....	88
Figure 5.10. Cassandra Overhead Performances Graph	89

LIST OF TABLES

TABLES

Table 1.1.	Comparative analysis of sharding security in various nosql databases	12
Table 5.1.	Test Results of MongoDB (Workload A).....	67
Table 5.2.	Test Results of Cassandra (Workload A)	67
Table 5.3.	Test Results of MongoDB (Workload B).....	69
Table 5.4.	Test Results of Cassandra (Workload B).....	69
Table 5.5.	Test Results of MongoDB (Workload E)	71
Table 5.6.	Test Results of Cassandra (Workload E).....	71
Table 5.7.	Test Results of MongoDB (Workload F)	73
Table 5.8.	Test Results of Cassandra (Workload F)	73
Table 5.9.	Test Results of MongoDB (Workload A).....	75
Table 5.10.	Test Results of Cassandra (Workload A)	76
Table 5.11.	Test Results of MongoDB (Workload B).....	77
Table 5.12.	Test Results of Cassandra (Workload B).....	77
Table 5.13.	Test Results of MongoDB (Workload E)	79
Table 5.14.	Test Results of Cassandra (Workload E).....	79
Table 5.15.	Test Results of MongoDB (Workload F)	81
Table 5.16.	Test Results of Cassandra (Workload F)	81
Table 5.17.	Security Features Comparison	84
Table 5.18.	MongoDB Overhead Results	87
Table 5.19.	Cassandra Overhead Results.....	87

LIST OF ABBREVIATIONS

ACID	Atomicity, Consistency, Isolation, Durability
AES	Advanced Encryption Standard
AUTH	Authentication
CA	Certificate Authority
CAP	Consistency, Availability, Partition Tolerance
CBC	Cipher Block Chaining
CQL	Cassandra Query Language
CRUD	Create, Read, Update, Delete
DCL	Data Control Syntax
DDL	Data definition language
DES	Data Encryption Standard
DML	Data Manipulation languages
DOS	Denial of Service attacks
ECB	Electronic Code Book
FIPS	Federal Information Processing Standard
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IT	Information Technology
JMX	Java Management Extensions
JSON	JavaScript Object Notation
KDC	Kerberos Key Distribution Center
LDAP	Lightweight Directory Access protocol
MD5	Message Digest 5
NOSQL	Non-Only Database Management System

RBAC	Role-Based Access Control
RDBMS	Relational Database Management System
RLAC	Role-Level-Access Control
SCRAM	Salted Challenge Response Authentication Mechanism
SQL	Structured Query Language
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TDE	Transparent Data Encryption
TLS	Transport Layer Security
XML	Extensible Markup Language
XSS	Cross-Site Scripting attacks
YCSB	Yahoo Cloud Serving Benchmark

CHAPTER 1

INTRODUCTION

The increase of sources of information and the explosion of data generated by users from social media and other content creating sites led to the need for large-scale data analysis platforms capable of storing and processing a massive amount of data. The advent of big data platforms was the answer to such need as often adheres to the principle of providing a high degree of availability and scalability.

Big data can be defined as the ability to handle and manage large data within the right time [1]. Big data is also known as a huge size of data with a complexity that cannot be handled by the relational database systems (RDBMS). Several technologies have been developed rapidly including hardware such as the design of power efficient with parallel processing support, and software such as highly scalable databases and big data processing systems to accommodate the rapid growth of data [2]. There are three forms of the big data [2]:

- 1- Structured data: Any data has a known length and format is considered as a structured data Such as names, numbers, addresses, phones, emails. The structured data can be generated either human as inputs or results of applications such as gaming data. It can be generated by machines and example of this kind: weblog data and sensors.
- 2- Unstructured data: The data with unrecognized formats and length known as unstructured data. Most of the data at present are of this type. It comes from many sources like websites, mobiles, automation systems, social media, and satellites. This means that it can be generated from both humans and machines.

- 3- Semi-structured data: It is a combination of structured and unstructured data. The amount of complexity in this data is very high. It is characterized as being long data, and it is recorded with a full size which leads to running queries to handle this type of data.

The advent of Big Data has led to the development of several key technologies and platforms to answer the ever-growing nowadays data. NoSQL was one of these technologies. Although NoSQL databases began to develop before the appearance of big data, the adoption of NoSQL databases did not occur until the need was created by big data [3].

NoSQL database systems have several advantages that differ from the relational database systems to which we can conclude them as follows:

1. It is easy to implement.
2. At least one node will hold the data before replicating it to the others.
3. NoSQL database systems are open source.
4. Data will be replicated to multiple nodes, and it can be partitioned.
5. It is easy to distribute because there is no need for a schema.
6. The storage itself is a non-relational (semi-structured, schema-less).
7. It can handle large data volumes.
8. The data should not be stored in a table, but in the documents that make it available, scalable with high performance.
9. It is cheaper than the RDBMS systems.
10. It has an agile design.

1.1 Aim of the study

Recently, many versions of NoSQL databases appeared each with different architectural design, its query language, and its own set of solutions for scalability,

compression, security, clustering, which render it to become difficult to decide which NoSQL database appropriately provide the solution to a given problem or issue. Therefore, studies should be dedicated to testing these databases concerning several factors, such as query processing speed, security of data, readiness to be used in a scalable environment, programming languages.

The aim of this study is divided into two-folds, firstly, we will test two widely used NoSQL database management systems, namely MongoDB and Cassandra. The test includes using third party tool known as Yahoo Cloud Service Benchmark (YCSB) which is designed to test NoSQL databases performance. However, real-world production environment always relies on database clustering to deploy databases such that to gain better performance, provide data safety and services with high availability during failovers. Single node test of MongoDB and Cassandra should give a thorough understanding of where both systems stand in term of performance and operations latencies. However, such test cannot be generalized to include the performance of the same databases in a cluster configuration. Therefore, we find it essential to include in this work the performance test of both databases in a cluster configuration.

Secondly, security of information can be considered to be the second importance if not the first factor (besides performance) to determine which NoSQL database to use. As there are no tools to determine or measure the security in databases, we will present several security factors by which we will determine which database is comparatively better in term of information safety and security.

Lastly, Databases Encryption Overhead is the time spent by the database engine to encrypt all incoming data streams plus the time spent to decrypt them to answer queries. Encryption generally is a resource consumption process; therefore, we find it essential to include benchmark test of both databases encryption engines. The results of such test can be used by IT decision makers when considering security measures of their production environments. If the data encryption cost is too high, one should think of other solutions for data protection.

1.2 Significance of the study

Around 150 different NoSQL database systems according to [4] exist choosing the appropriate database hard and complicated. The present study covers comprehensive performance test of two widely used NoSQL databases: MongoDB and Cassandra. The study will cover testing of both databases in single-node and multi-node (cluster) configuration, and it will help IT decision makers to determine which database is better according to their deployment requirements.

Although many studies have compared several NoSQL databases systems regarding performance and other features, such as supported languages, scalability, we found no published studies on the latest version of MongoDB (3.6.3) and Cassandra (3.11.1). Similarly, through literature from 2013-2018, we have not found any scientific study dedicated to studying the overhead (cost) of data encryption in NoSQL databases. Therefore, the second significance of this study is to provide a thorough analysis of encryption engine in Cassandra and MongoDB.

1.3 Research questions

- 1 – Which NoSQL database system's (MongoDB 3.6.3 or Apache Cassandra 3.11.1) performance is better according to low average latency and high throughput concerning read, update, scan, insert, and read-modify-write operations?
- 2 – Which NoSQL database system's cluster performance is better regarding read, update, scan, insert, and read-modify-write operations?
- 3 – Which NoSQL database system is more reliable for production environment use?
- 4 – Which NoSQL database has better security features?
- 5 – Which NoSQL database provides less encryption overhead?

1.4 Related work

1.4.1 NoSQL Databases Performance

Determining the right solution by IT decision makers is decided by several factors, for example choosing the right database for a project must be according to performance, security, cost, etc. NoSQL is one of the recent trends in Big Data as it offers a high degree of availability and scalability. Moreover, therefore, it can address the characteristic of Big Data where a massive amount of generated data must be stored and accessed later by different services. A survey of the literature showed that the current research trend is divided into two main categories:

1.4.1.1 NoSQL databases vs Relational Databases

These studies test the performance in specific application-oriented environments. For example, Yishan Li compared the performance of Microsoft SQL express with an early version of NoSQL databases including MongoDB, RavenDB, CouchDB, Cassandra, Hypertable, and Couchbase. During his test, several NoSQL databases outperformed Microsoft SQL while in some other tests the contrary happened. He also suggested that like any software application, NoSQL databases implementations go through changes over time and thus performance enhancements are more likely to happen [5].

Another study published by Parker in [6], which compared SQL server to one of the recent versions of MongoDB, showed that MongoDB outperformed SQL in inserts, updates, and simple queries, while SQL performance was much better when querying non-key attributes, and when aggregating queries, see Figure 1.1.

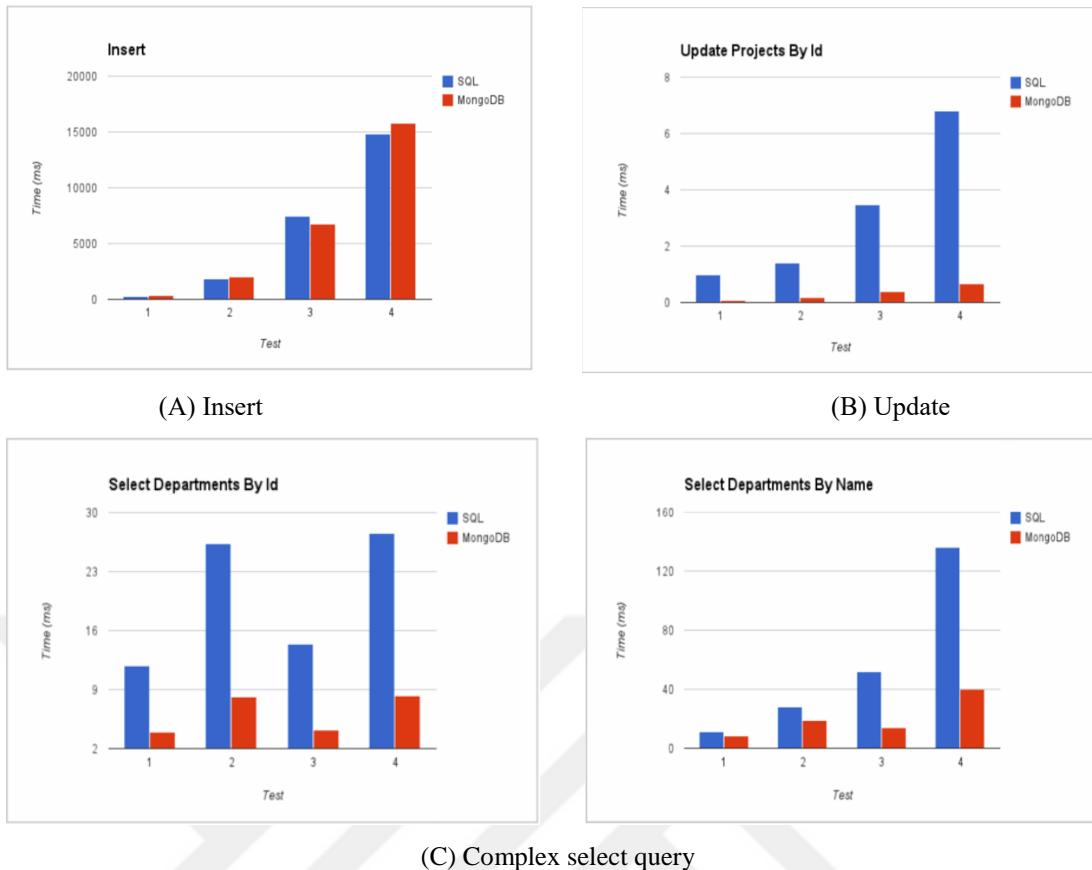


Figure 1.1. Databases performance comparison [6].

MongoDB was also the subject of another study published in [7] where the researchers tested the scalability and performance of MongoDB and MySQL to process complex queries. It showed that MongoDB was able to outperform MySQL in handling complex queries, however, when using YCSB benchmark, MySQL performance was better. When the number of simultaneous connections was increased, the performance of both databases was similar.

Cassandra, Shepra, and HBase were also compared in a similar study to MySQL according to nine selected features, such as replication, high availability, implementation language, persistence, and performance [8]. The researchers concluded that although the SQL and the NoSQL databases have several shared features, their performance behaviors are not similar in specific given tests. Therefore, they cannot be used interchangeably in a production environment. However, we would instead choose one of the two types of databases for a given instance.

It is important to note that there is more published work which studied and compared SQL and NoSQL databases, but not from performance perspective such as comparison of SQL and NoSQL in the cloud [9], and a survey and comparison of relational and non-relational database [10].

1.4.1.2 NoSQL vs NoSQL databases

NoSQL databases are compared against each other regarding performance, security, and scalability in Clouds, Big Data platforms, and building applications. The related work in this section is reviewed starting from 2013. Abubakar evaluated the performance of Cassandra, HBase, MongoDB, OrientDB and Redis in [11] using YCSB benchmark tool and he found that memory based databases are the fastest disregarding the workload while document based databases are the slowest when the number of updates is increased. It was concluded that MongoDB, Redis, and OrientDB are better in performing read operations, while Cassandra and HBase have a better execution of updates operation. The Figure 1.2 below shows the overall average time of three workloads.

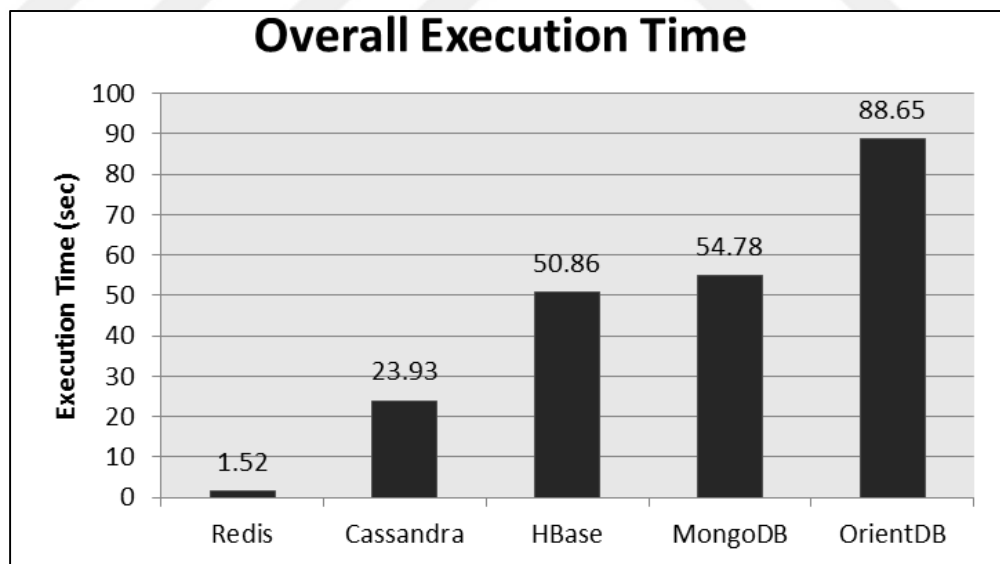
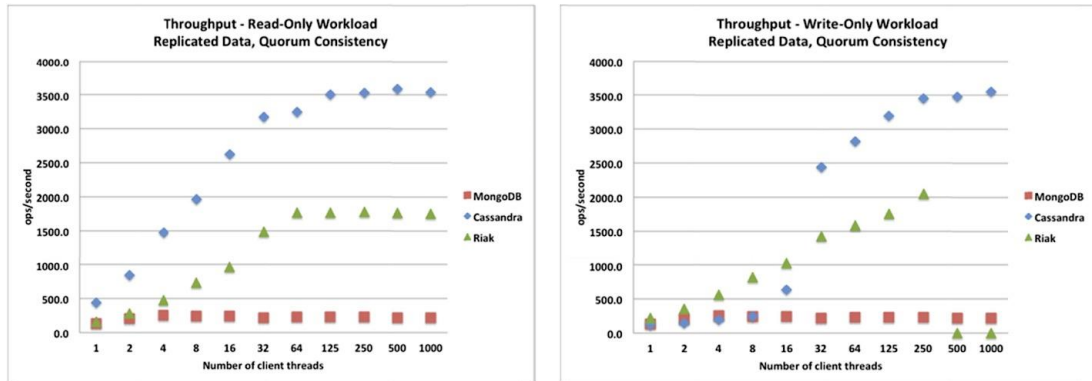


Figure 1.2. Overall average performance of three workloads [11].

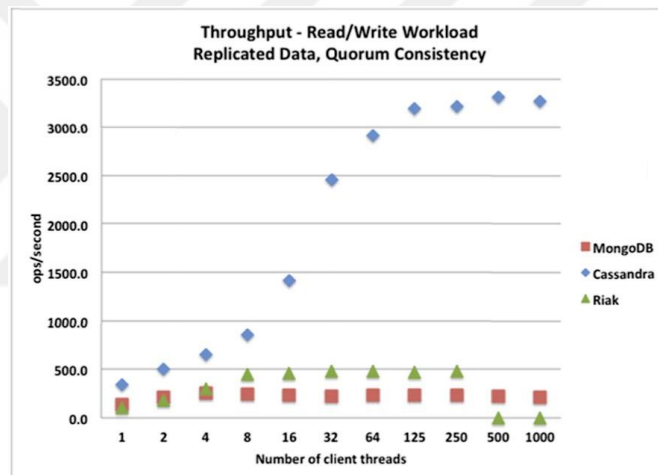
Similar research was published to study the performance behavior of MongoDB, Cassandra, and Riak. When the number of connections (users' threads) increased, each database is tested as a single node and in nine node-cluster. Figure 1.3 shows

the observed performance results for Read-only, Write-only, and Read-Write tests [12]. It shows that Cassandra outperformed MongoDB and Riak concerning throughput but with highest latency.



(A) Read only test

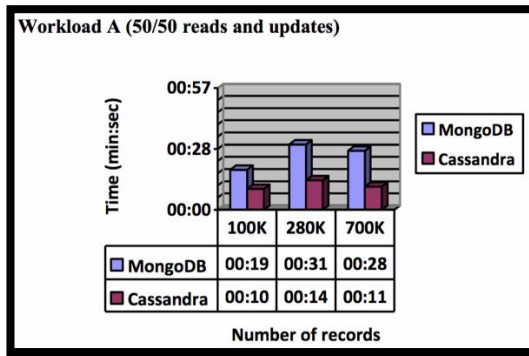
(B) Write only test



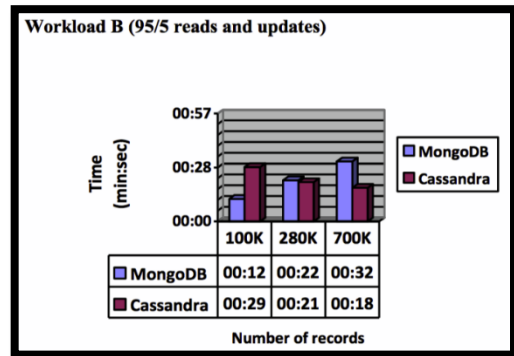
(C) Read and Write test

Figure 1.3. Throughput test results [12].

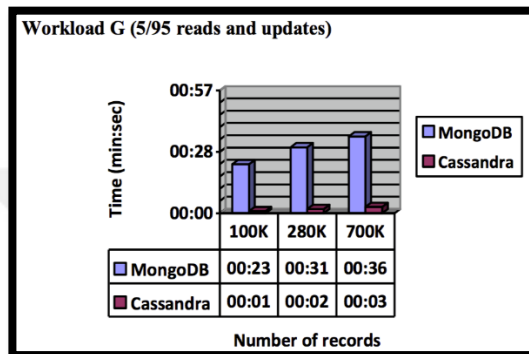
Abramova compared MongoDB against Cassandra regarding fifteen different features including performance. Six workloads were chosen to test the performance of each database at 100K, 280K, and 700K record size [13]. The Figure 1.4 below depicts the obtained results of four selected workloads A, B, G, and F.



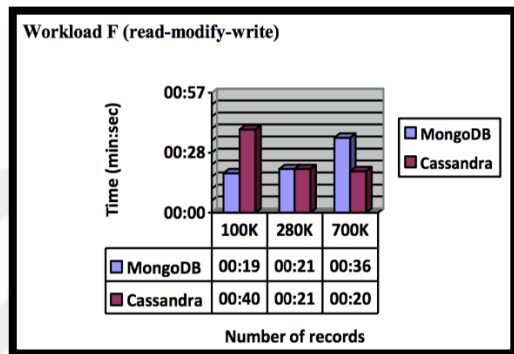
(A)



(B)



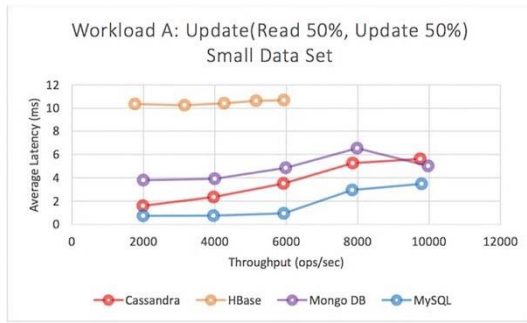
(C)



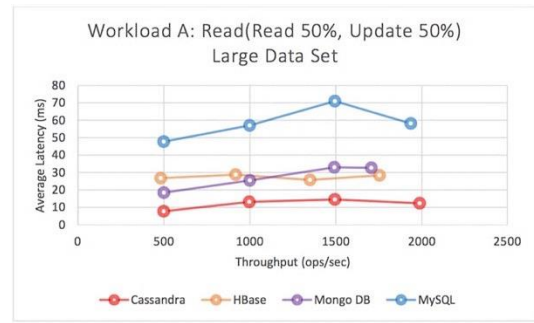
(D)

Figure 1.4. Workloads test time results [13].

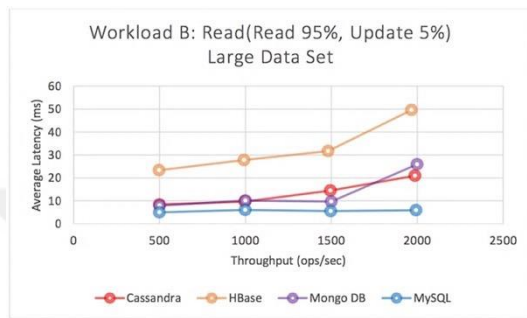
In 2014, a study was published by Jay which studied the performance behavior of several databases when configured with small and large data workloads [14]. The study included three NoSQL databases, Cassandra, HBase, and MongoDB, and one relational database, namely MySQL. The researcher concluded that each database performance is entirely different when performing operations on small dataset when performing the same operations on a large dataset. Therefore, measuring the performance of any database must come from testing several different workloads of different size. The Figure 1.5 below shows the obtained results for workload A, B, and F.



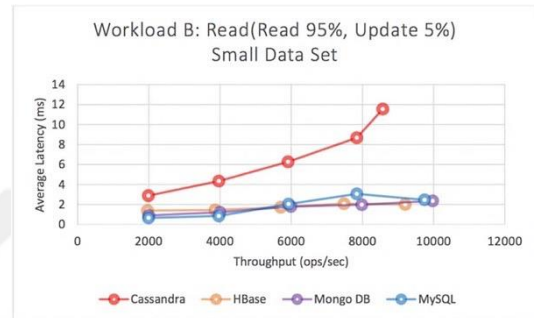
(A) Workload A – Small dataset



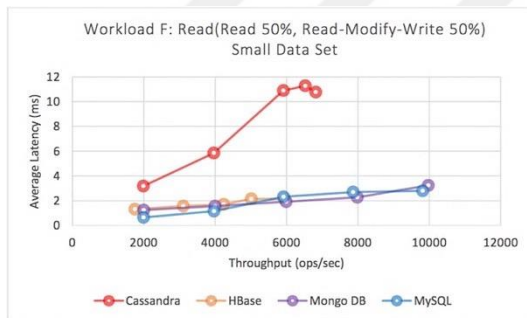
(B) Workload A – Large dataset



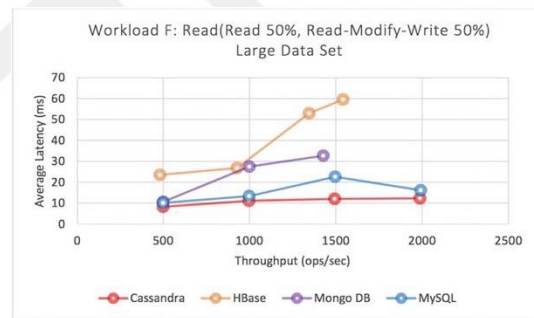
(C) Workload B – Small dataset



(D) Workload B – Large dataset



(E) Workload F – Small dataset



(F) Workload F – Large dataset

Figure 1.5. Small and Large workload test performance results [14].

Lastly, Kumar and Roseline published a work in 2017 [15] which studied the effect of increasing the number of CPU cores on the performance of three NoSQL databases. The researchers studied Cassandra, MongoDB, and HBase with a number of nodes from 10 to 40, and threads count from 30 to 60. The researchers concluded that MongoDB was the slowest concerning performance scalability of the three. Figure 1.6 depicts the scalability test results.

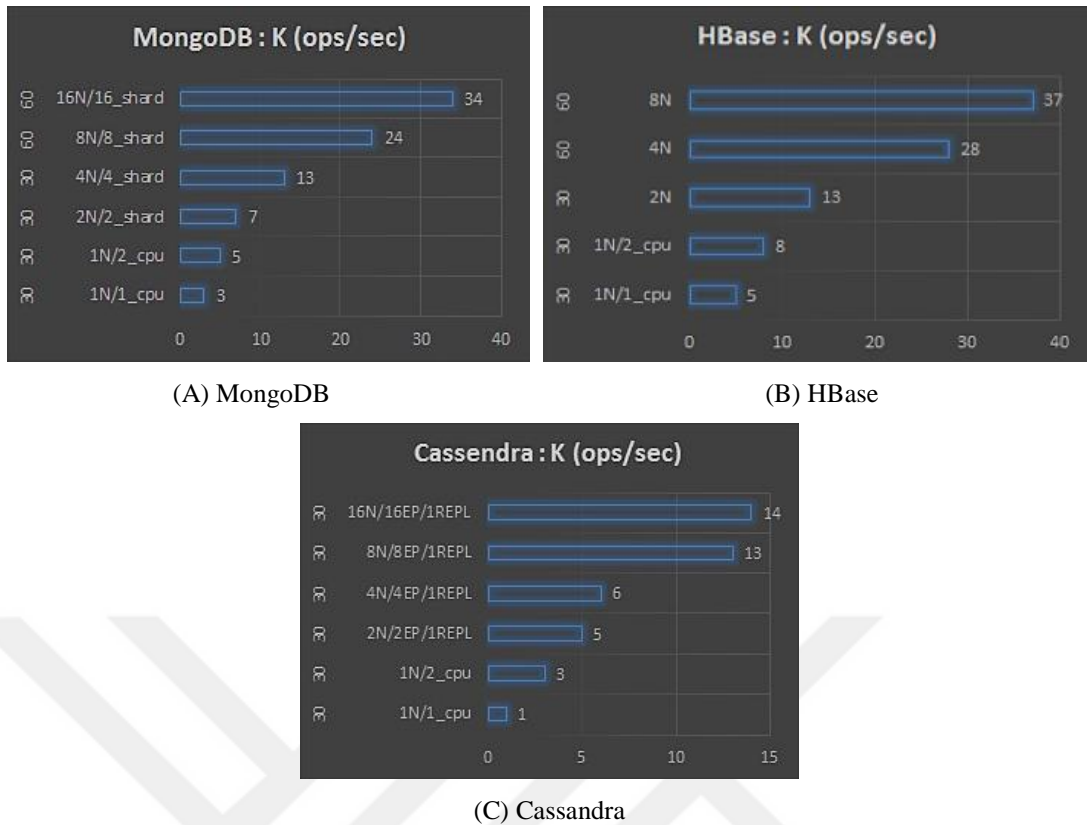


Figure 1.6. Scalability performance results [15].

1.4.2 NoSQL Databases Security

The broad adoption of NoSQL databases and big data is due to the fact that NoSQL databases are capable of handling unstructured data, scalability, and performance. However, that brings more challenging issues, and one of these issues is security. Most of the developers of NoSQL databases focused on the performance at the expense of security. Thus, NoSQL security was the subject of several pieces of research and studies. Okman studied the security features of Cassandra and MongoDB such as data encryption, authentication, authorization, query language, and auditing. He concluded that both databases are vulnerable to denial of service attacks, weak authentication, no support for file data encryption, and very simple authorization mechanism [16]. Another work was published in the year of 2014 where six NoSQL databases were studied extensively using five security features. The study concluded the following results shown in the table 1.1 below [17]. Researchers suggested that NoSQL databases need several improvements for customers and vendors. Similar observations were also made by Preecha Noiumkar and Tawatchai Chomsiri in [18].

Table 1.1. Comparative analysis of sharding security in various nosql databases [17].

Database Criteria	MongoDB	Redis	CouchDB	Cassandra	HBase	Couchbase Server
Authentication	Medium	Low	Medium	Low	Medium	Medium
Access Control	High	Low	Low	Low	Medium	Low
Secure Configuration	Medium	Low	Low	Low	Low	Low
Data Encryption	Medium	Low	Medium	Medium	Low	Low
Auditing	Low	Low	Medium	Low	Medium	Medium

In 2015, a comprehensive study was published by Srinivas [19], where he compared three NoSQL databases and two SQL databases using the three bases of data security foundation; Confidentiality, Integrity, and Availability (CIA). The research concluded that NoSQL databases rapidly evolve at high speed regarding security. They also suggested that according to the growth of open source NoSQL databases, it is safe to predict that the security reaches a comparable level with relational databases. Lastly, in 2017, Cuzzocrea and Shahriar proposed data masking technique to further improve several security concerns in NoSQL databases [20]. Several security issues were solved using the proposed method while they suggested it was difficult to use the same technique to address the rest of the issues without additional development.

1.5 Thesis structure

This thesis consists of six chapters. The second Chapter describes the theoretical background of the NoSQL systems and management techniques. Chapter three details the information regarding the databases used in this study. While Chapter four describes several implementations procedures of test environments used in this study, Chapter five illustrates and details the observed results. Chapter six includes the conclusion of this work and suggestion for future work.

CHAPTER 2

THEORETICAL BACKGROUND

This chapter focuses on the NoSQL databases and the related concepts, and the types of NoSQL databases, the MapReduce, replication, sharding and scaling concepts will also be covered in this chapter.

2.1 NoSQL Data Stores types

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

- 1- Document data stores
- 2- Key-value data stores
- 3- Column-based data store
- 4- Graph stores

Document data stores: Instead of using columns with names and data types like the traditional relational database system we can deal with data as an object. In other words, we can create an object with structure and then add the new information as an instance of an object and these techniques can be classified as a type of the key-value. It can be stored in different types such as JSON, XML, or any of the NoSQL document data. An example of this kind of stores is MongoDB and Apache CouchDB. Also, this kind of stores is known as semi-structured data, but there is a risk of working in this type of stores. This kind lacks the current requirement [21].

Key-value data store: The mechanism here works by matching the key and the value of storing, retrieving and managing the data, similar to arrays. Also, this mechanism is known as hashing [22]. Riak and Redis are examples of this kind of stores. This type is known as an in-memory database because of its dependence on the main memory of the computers data storage [23].

Column-based data store: They are also known as wide-column data stores. This type of stores is one of the most potent stores in NoSQL database systems. It is the same as the tables in the relational database systems but more expanded. The columns here are dynamic and can be modified during the process of the operations. The main benefit of using this type of stores it to provide high throughput and fast lookup but the calculation cost is high [21]. Rows of the same table can be associated with different columns. BigTable, Cassandra, and HBase are examples of this kind of stores.

Graph stores: With the development of the social networks, this kind becomes more active. It represents the data as networks [24]. An example of this type of stores is Neo4J and InfoGrid. The data representation here is nodes, edges and the features for them. This type is suitable for high level connected data like social media networks.

2.2 Scaling

We can achieve scalability with the NoSQL database systems by using additional hardware resources and loading the data flowing towards those hardware resources or by depending on a cluster with many computers that work as one powerful computer [25]. There are two types of scaling techniques: sharding and replication. The three central operations for scalability in NoSQL databases are reading, writing and the volume of databases. The two terms that are related to databases which need to be clarified are the following

- **Scale-up:** This concept means the increase of the power of the machines, like increasing their memories, using faster processors, increasing the number of cores, and using faster memories to make the database be able to process more operations.

- Scale-out: Increasing the performance of database systems can be carried out by distributing the machines to work together in a distributed manner to respond to the requests of managing data as shown in the Figure 2.1 below.

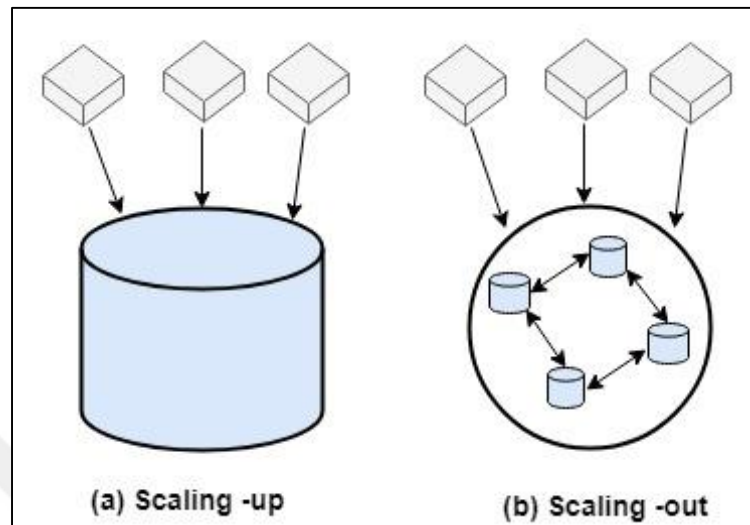


Figure 2.1. Scaling-up versus Scaling-out

Scaling-up still has restrictions which cannot be passed through scaling-up and scaling-out are used to avoid the problem that concerns the increase of vast amounts of data. It is impossible to continue adding memory or cores to increase the power of a machine. Accordingly, one can say that hardware restricts scaling-up. Conversely, we can only add more machines to clusters and increase our clusters to meet the difficulties of the considerable increase of data.

2.2.1 Sharding

To handle the big data quickly and more accurately the term sharding appears within the scalability. Sharding is a process of segmenting the data into smaller pieces called "Shards" as showing in Figure 2.2 below. This process is the same as the parallel computing with retrieving addition. The first sharding techniques were implemented in the RDBMS, but the sharding features have been designed within the tables. By dividing the tables to many storage sources, the sharding does not give the join ability between the shards like the join between tables [26]. Those parts are not sharing anything because they are just separated parts from the same source. Sharding is precisely similar to the horizontal partitioning in the RDBMS which

takes a big database and divides it into smaller databases to handle it easily. The sharding becomes more effective to the cost. To use one large size database, we need a powerful end-computer to manage it. Sharding does not require a high computation because of its size compared to the size of its origin [27].

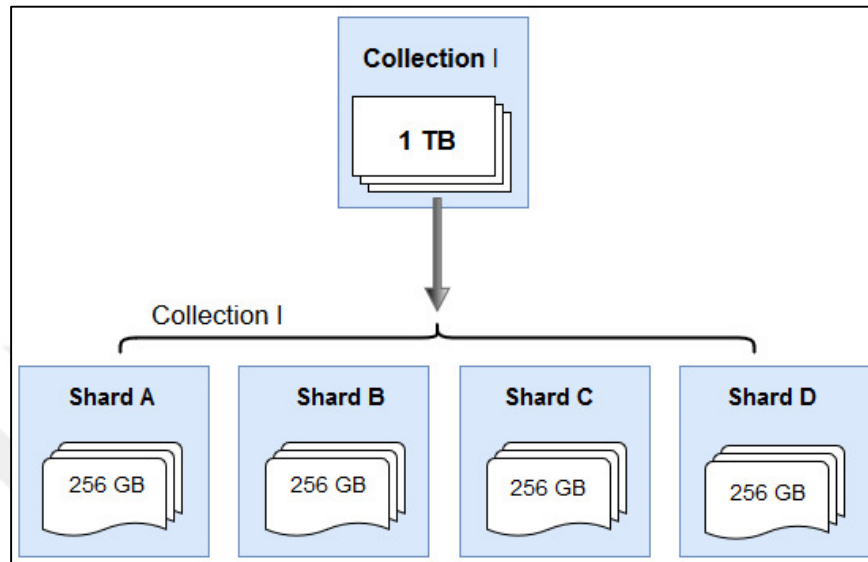


Figure 2.2. Sharding

So when we have an overload from many requests, for example, if we have five server's nodes then the work will be divided automatically into 20% for each server. However, we need to keep in mind the main factors that affect the response speed for a request, and this is called the geographical distribution. When the server receives a request from a user, then the nearest server to that request source should respond to that user [28].

One of the critical things that need to be decided earlier is when to use the sharding. Some databases provide the sharding from the beginning while other types give it as an option that can enable later. Moreover, this is related to the fact that some databases start feeding the data from scratch and starting with one node is less costing and the sharding is unnecessary. This causes many problems for many teams later when they decide to shard their data to lose their data permanently because the sharding occasionally depends on consuming the data [29].

2.2.2 Replication

The replication process in NoSQL databases involves copying the same data to many copies and making each copy in different servers. Also, this will provide us with the speed of responses, and this is the main point in web applications which depend on the significant data technology and also provide us with the ability to add and update our data on the files [30]. The Figure 2.3 below explain how we can replicate a database between numbers of serves.

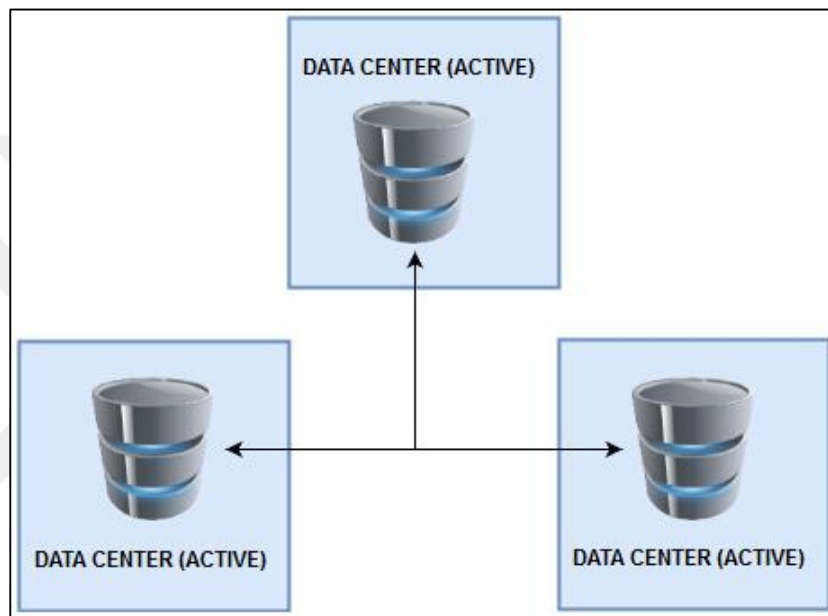


Figure 2.3. Replication

This method can be implemented by:

- **Master-slave replication:** This design includes Master/Slave, the master has the writing responsibility, and slaves handle the synchronization of data to the master to update the primary information. The main benefit of this method is to avoid the failover of one of the replication and recover it. When a master node fails, the slave nodes will change the state from read-only to read-write state and another slave node will be converted into master node then the slaves will reconnect to the new master node and return to the read-only state [31]. Figure 2.4 explains all details of master-slave replication.

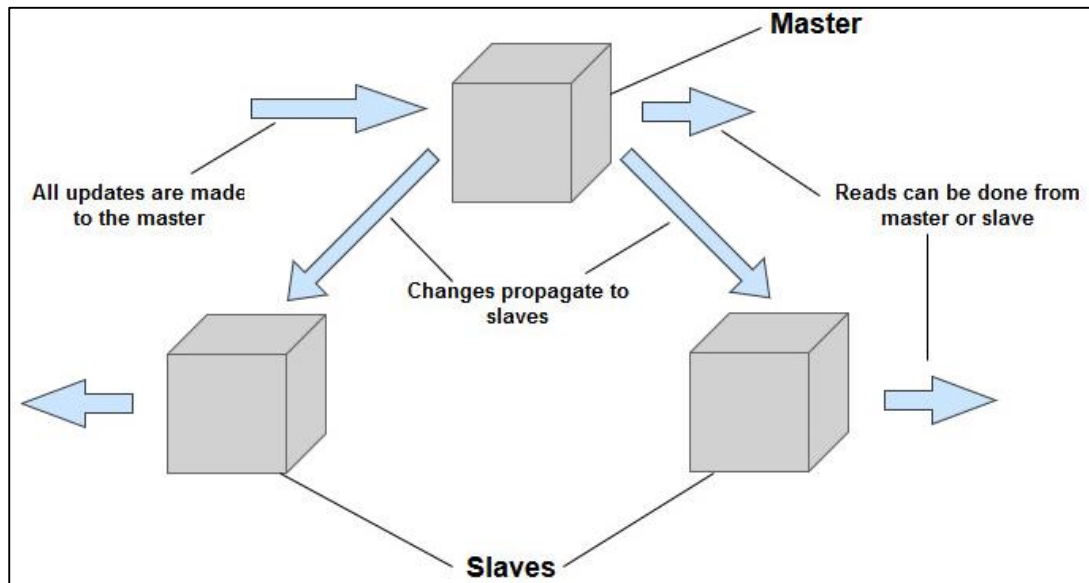


Figure 2.4. Master-Slave Replication

As concerns the advantages, the Master-Slave replication has a high scaling for reading a significant amount of data. When we need more requests, we can add more slaves to overcome that. However, this ability is sized by the master node to control the flow of the data to slave nodes. Also, the write operations are not that much good and when we need to handle this kind of operations we need to hold the read operations until the system deals with the write requests [32].

The disadvantages are when the master node fails the data in this node will be unreachable, and this will cause a loss in the data from the master node. The other disadvantage is that the new master node will take duration of time before it set and during this process the slave nodes will read and write data at the same time which can store different data on the same subject. When many users request the data, there might be a high chance that this data will differ due to this temporary process of master fail and converting the links from the slave nodes to the new master node [33].

- Peer-to-peer replication: The difference is that this design gives the ability of writing operations for each peer (node); moreover, we can add more nodes easily, and access the data store even with losing any peer while the master-slave method is not possible. The read here is good, but still, we have the write operations not effective [33]. Figure 2.5 explains all details of peer-to-peer replication.

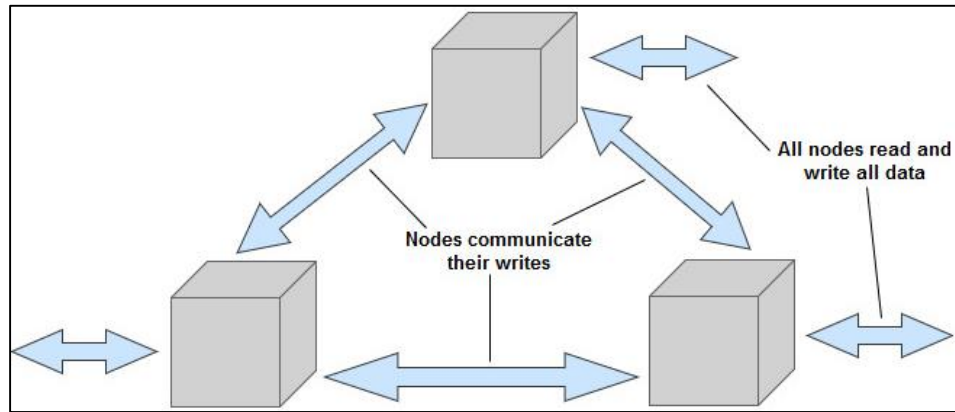


Figure 2.5. Peer-to-Peer Replication

The benefit of using this type is that we can quickly pass the failover of the node without losing the reading ability from the other nodes. Another good reason to use this type of replication is that we can quickly add more nodes without caring to the controller of those nodes. The worst case of the type is when many people reach a node and retrieve then update the same node which easily can make a conflict with the update operations [34].

2.3 MapReduce

Google, in 2004 presented a new technique which is called MapReduce, it is described in a corresponding article in [35]. It is defined as a technique used to get a significant amount of data within a short time. Several computers are used to divide the tasks of filtering and fetching data. The concept of MapReduce has two principal functions, mapping and reducing. By combining a key and value, the map function is used to create intermediate (Key-value) pairs. After the generation of all the intermediate keys, the reduce function uses these created keys to move all the intermediate values with their intermediate keys. See Figure 2.6 below.

```
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 2.6. MapReduce functions for word counting

2.4 NoSQL data store foundation

The three main principles of the foundation of the NoSQL databases: are CAP Theorem, Base property and Consistency Model [36].

2.4.1 (CAP) Theorem

In 2000, the CAP theorem was made by Eric Brewer [37]. Moreover, demonstrated in 2002 by Gilbert and Lynch [38]. The CAP Theorem theorizes that three fundamental properties represent any distributed system.

Consistency: The same data is shown to many clients, and this includes the reading and writing. The consistency of the data through the network is well distributed, and this gives us the latest form of the data since we are working on the last save of the data, but that is not guaranteed since the user will see the older version of the data. The data should be the same one on all the nodes [39].

Availability: Even with node failover, the cluster is still accessible. In more details, when we try to access a node and its failure, then we always connect to another node to provide us with our query, and it will be the same as our query [40].

Partition Tolerance: Any physical break in the communication between nodes or fail in the hardware does not affect the cluster functionality. For instance, if we have two data centers and suddenly the connection breaks then the communication between those data centers is off, and we will lose the partition tolerance property. As a conclusion, in any distribution system, there is a need to guarantee the connection to fulfill the partition tolerance concept. If the partition tolerance is not available in the system, then it is not considered as a distributed system [41]. The main idea of CAP theorem is explained in Figure 2.5.

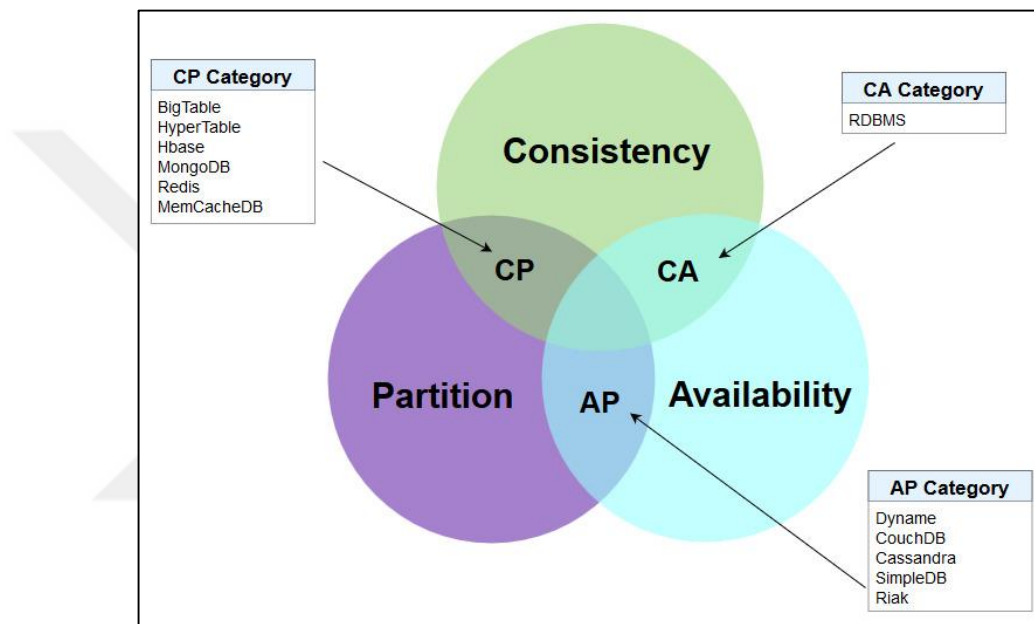


Figure 2.7. CAP THEOREM

The distributed system cannot handle all those three components concurrently. For example, the system needs the partition tolerance as the most part otherwise it cannot become distributed. Any other part (availability or consistency) with the partition tolerance is combined to present the distributed system.

The types of the data store design are:

- **CA systems:** A system consists of consistency and availability properties, but it is not considered as a distributed system because we have already mentioned that any distributed system needs the partition tolerance as one of its parts to become distributed and this design does not have it. The relational database systems are

the example of this design for the reason of low scalability, and when the load is low Aster data, and HDFS Name node is the applicable systems [42].

- **CP systems:** a distributed system design contains the consistency and partition tolerance. The main problem with this design is that any node gets corrupted will make the cluster inaccessible. Hbase is a pure example for this kind of systems. MongoDB is considered as CP system, but recently this opinion is excluded when it has proved that MongoDB does not always work as a CP system. Therefore, it cannot be considered as a complete CP system [43].
- **AP systems:** This design of programming implementation is hard, but it still provides the availability and goodness when the low-latency is required. In this system, whenever a node fails the system has the accessibility until the corrupted one is fixed. However, here there is no insurance that all nodes will have the same data. Moreover, most of the time the requests will be the initial value like the caches as an example of this type. Couch DB and Cassandra are the good examples for this kind of systems [38].

2.4.2 BASE

The BASE is the first primary element for the system reliability. The base is derived from the words Basically, Available, Soft state, Eventual consistency. The first part state that the system cannot offer the availability like the CAP theorem, but it is possible to change its state frequently, regardless no input is there. This case is called the soft state. The system becomes consistent due to the system which spends certain time with no input. Those prefer the priority rather than consistency. Also, it considered an AP system. The good thing here is that the BASE will provide the NoSQL: speed, scalability, simplicity and the horizontal distribution [44].

2.4.3 The Model of Consistency

Werner Vogels states that inconsistency the clients and servers have a different attitude and represent it in three client-side types [45]:

- **Strong Consistency:** The best option for enterprise systems because it retrieves the last data inserted into the system.
- **Eventual Consistency:** It states that when the data is written, it will be able to appear for reading for the users but not directly after finishing the written operation which means there is a chance to retrieve an updated version of the targeted data. Also, all the nodes will get the same data.
- **Weak Consistency:** There is no guarantee of retrieving the right value at the same time when access the system.

The NoSQL database systems contribute all or part of the consistency. Hbase is treated as a NoSQL system with strong consistency while Cassandra has been treated as eventual consistency.

2.5 Security Issues in NoSQL

The security is one of the main problems that is related to RDBMS and then it is inherited to the NoSQL databases and to that due to the transactions that have already been used which deserve the RDBMS and their distribution: transaction manager, query processor, security manager, metadata manager and integrity manager [46]. Moreover, the advantages of using those distributions are the providing of [46]:

- high performance
- scaling
- resiliency
- network cost reduction
- network traffic reduction

However, even with those benefits, the security is the primary concern, some companies used central databases and covered them with types of security protection like the network protection or even encryption tools. However, this issue is hard to cover with the distribution of the databases.

From all above, we need to keep in mind that the NoSQL database systems are extended of the RDBMS, and the concern that we have faced in RDBMS is still risky with technical changes in the methods [46].

The main reason underlying the use of the NoSQL databases is the performance and the real-time response for a large volume of data that does not structure like the RDBMS tables [46]. Still, the security issue is not covered so much because the foundations focused on the performance at the expense of the other fields, the security is one of them. Here we need to clarify that the NoSQL databases do not share the same architecture since that those databases designed to handle the requirements of the cloud computing applications [44]. Nowadays, most of the NoSQL databases face almost the same attacks from the RDBMS era, even with providing many solutions to improve their security many attacks made many problems such as SQL injections attacks, script injection attacks, and those attacks are still active within the NoSQL databases area [47]. Moreover, that is related to the dependency on a specific data structure in NoSQL databases (unstructured data) like the JSON files and many others which can be used to cross the firewalls and the security filtering with simple JavaScript code, and the system will consider it as data files. Another reason is the distribution environment of the NoSQL databases that are developed to handle the parallel computing and the sharing of data between nodes with a high probability of stolen data. Also, that makes the NoSQL databases open against a vast area of attacks which complicate the security problems [48]. Generally, we can list the security problem of NoSQL databases [49] as follows:

1. lack in encryption of the data files
2. does not provide enough authentication in client-server networks
3. simple authorization
4. SQL Injection Vulnerability
5. filtering at the end point input
6. encryption of the attribute
7. methodology of the attribute relationship

8. Granular access controller
9. unsafe computation
10. preserving data mining and analytics

States that the attribute relationship in NoSQL databases is to protect the information in big data and considers the attribute as a key to extract the related information and assume that the attribute with higher relevance is more important than the other attributes. In the same paper, they explained the attribute encryption method. This method gives the user the ability to encrypt the data and clarify the accessibility to this data and those users are the only people who can decrypt the data [50].

2.6 Security Threats in NoSQL

The main threats in NoSQL databases usually take place in the middleware and to overcome those threats we need to classify them, Figure 2.8 below shows security threats in NoSQL databases.

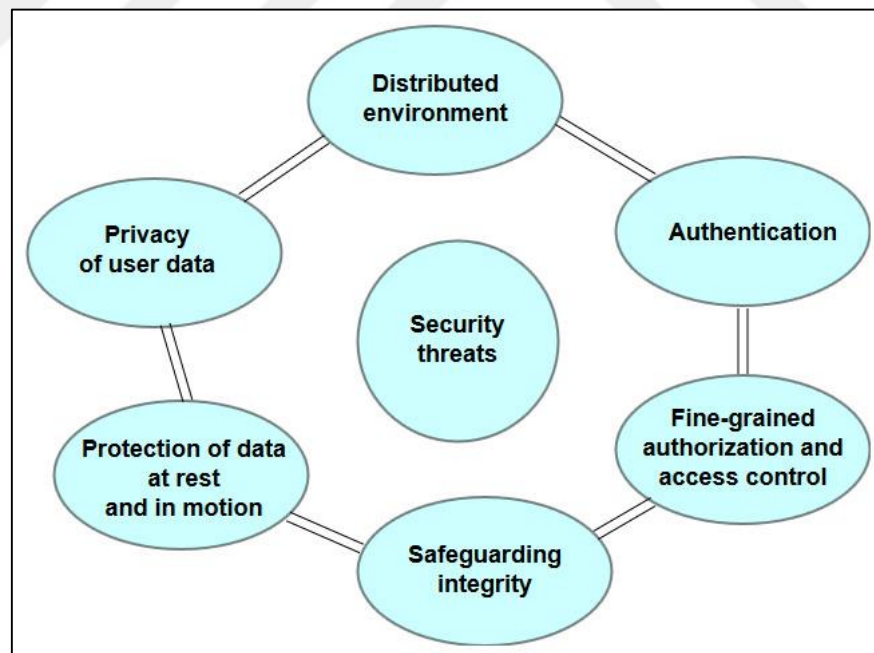


Figure 2.8. Security threats in NoSQL databases

2.6.1 Distributed Environment

Most of the databases are distributed in the design by a network to give the user the accessibility remotely and locally. The synchronization of the writing and updating of the content of the databases are controlled by a central database [51]. In the distributed environment, there are many nodes spread over the network, and they work in parallel which gives more area for the attackers as shown in Figure 2.9.

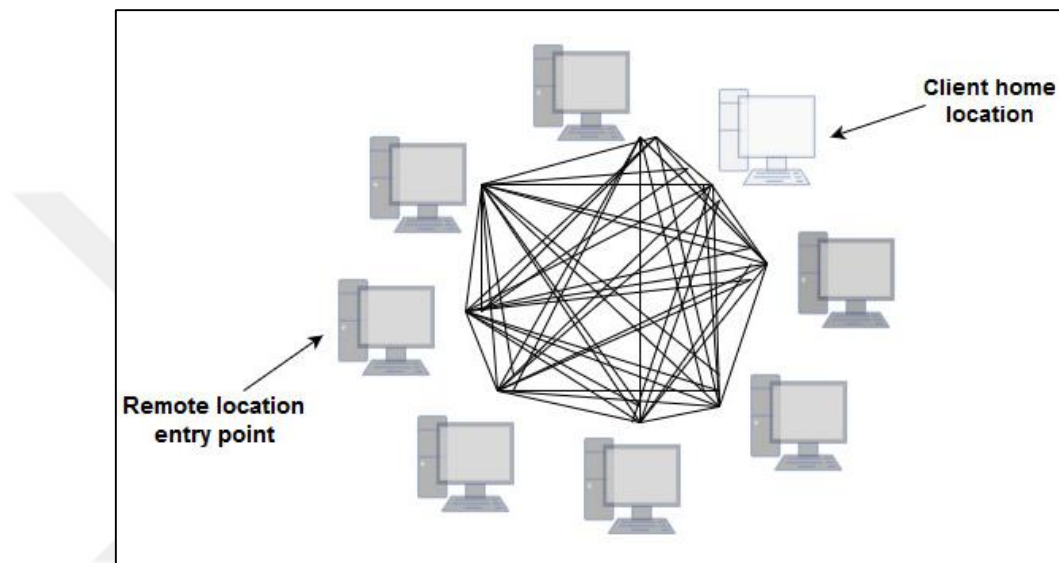


Figure 2.9. Security Threats with Distributed environments

Also, the maintenance here is expensive in term of computation with a higher chance for error to appear and a high ability for the data to get stolen and all that related to the fact that there are no central management systems [52].

2.6.2 Authentication

The authentication in NoSQL databases is provided at the local node unlike the servers, and that makes the NoSQL databases widely exposed to different attacks which lead to losing in the data to the attackers' accounts [16]. As an example of this kind of threats, we have the NoSQL databases Kerberos, which gives the authentication to the client nodes, but the attackers can get unauthorized access by duplication of the Kerberos ticket. The Kerberos ticket is an authentication mechanism in network developed by MIT. It works while the user is sending a

request for login by asking for a ticket from the Key Distribution Center (KDC). As an acknowledgment, the KDC will send a Ticket-Granting Ticket (TGT) for the user and ticket encrypted with the user password. The user will receive the TGT, and if the user decrypts it with the right password, then the ticket will provide the user the ability to log into the system [53]. It is an excellent way to authenticate, but still it is critical, and this is due to the reason that one side carries it (KDC), and the attackers can crash through the network perimeter, hijack credentials and use them to move through the network, taking additional credentials and escalating privileges along the way to accomplish their goals [52].

2.6.3 Safeguarding Integrity

The safeguarding integrity is a requirement declaration in which the database does not give unauthorized accessibility to a user to modify the content of the data in the database, and those operations can be deleting, updating and insertion. In NoSQL databases, there is a lack in both of integrity and confidentiality. All that is related to the reason that we do not have a scheme or tables to determine the permission and the updating on that table or row. In NoSQL databases, we might face a situation when many users modify a data, and that will give different copies of the same data. Also, that makes the transaction in NoSQL databases hard [34]. That is why the financial sectors do not depend entirely on the NoSQL databases. It lacks a central controlling for that kind of transaction [54].

2.6.4 Fine-Grained Authorization and Access control

To protect data, we need first to authenticate the right use of that data. Moreover, when the user passes the authorization, he will become a part of the database system. However, in NoSQL databases, the situation is different because it lacks the schema and it cannot give the authorization on tables of part of the data. There is object level security which means that we do not have a fine-grained in NoSQL databases. Access control in databases depends on the classification of the authorization of the database. Access control and both disk and memory accounting are performed at the column-family level. Families Column keys are grouped into sets called column families, which form the basic unit of access control. The whole data stored in a

column family is usually of the same type. A column family must be created before data can be stored under any column key in that family. After a family has been created, any column key within the family can be used. However, the access control differs from one database to another depending on the security level that has been promoted to the users. Moreover, in NoSQL, we do not have a schematic model which makes it hard to implement [54].

2.6.5 Protection of Data at Rest and in Motion

When we finish working on the data and flush it, and then we put it in the storage unit, we call this data at Rest. In Data Motion, when we have communicated with data, there are transition swaps. It is classified into two types: Inter-node communication and client-node communication. Most of NoSQL databases do not serve with the data at rest protection method. It might provide some encryption level on the data to protect it inside them, and that makes the data ambiguous [16]. Some NoSQL databases provide a partial level of data at rest and data in motion. In MongoDB, we have two encryption method for the data at rest. While Cassandra uses an encryption method called TDE (Transparent Data Encryption) for data at rest protection which consists of 6 encryption algorithms. However, the weak point here is to commit log in Cassandra because it is not encrypted which will lead to a severe gap in security. In data in motion, neither Cassandra nor MongoDB provides any Client-node communication. In Inter-node communication, Cassandra does not support any encryption, but we can modify the Inter-node communication by reconfiguring the server encryption options in the file `Cassandra.yaml`. MongoDB does not provide any inter-node communication.

2.6.6 Privacy of User Data

It is one of the main challenges that face the NoSQL databases due to its relationship with the dependency of the web-based application. While the NoSQL databases handle a significant amount of data with fast responses, the privacy is the most active threats in the big data field. Any NoSQL database system can deal with many nodes of users, but the problem starts when data is injected from an attacker to the system,

and it can spread out in the entire system due to the fact of nonexistent of central management [55]. Many challenges confront the user data privacy and to mitigate those problems the developers try to overcome them by providing a more secure mechanism and methods to protect the data. There are many types of those risks such as the cryptography of data. This is the first defensive line against the attackers which requires powerful algorithms to protect the data. Another challenge is the data mining analysis. Data analysis is good to provide more information about users to help in giving the right reply for a user request. At the same time, however, the privacy is at risk of getting uncovered by the analyzer itself. The last example given by us is the divided access control in the big data. When there is no main controller for the users to flow through the network, there is a high probability for the attackers to access through those networks as a normal user [56].



CHAPTER 3

NOSQL DATABASE MANAGEMENT SYSTEMS USED IN THE STUDY

This chapter presents the NoSQL Databases Systems that are used in this study: MongoDB (3.6.3) and Apache Cassandra (3.11.1) with detailed information about each system. There are two significant parts to be explained: the first part is the architecture of each database system and the different aspects between them, besides how each one of them handles the flow of the data and how it manages them. The second part will be about the security of the NoSQL databases and the security architecture of each databases system and for both open source and enterprise editions of databases.

3.1 MongoDB

MongoDB is one of the most known NoSQL Database systems in the current days. It is considered one of the open source software under the GNU license and developed by 10gen Software Company in 2007 before it changed to the name MongoDB Inc. [57]. It is developed by the C++ programming language with scheme-less and document orientation. It also has a structured mechanism for the document. It is suitable for storing the large document as JSON File (JavaScript Object Notation) and many other files such as videos and images.

3.1.1 MongoDB Architecture

In the beginning, the developers of MongoDB kept in mind two main features to focus on as NoSQL database system: Scalability and High Performance. They designed it in a way to keep up with the development of the applications and

accelerating evolution of the technologies. It stores the data with high capability and real-time, and that makes MongoDB suitable for the online applications. MongoDB has the ability to mapping the memory files, and this helps the operating system to manage where to store the file in the case on the storage memory or in the virtual memory. Concerning this fact, the MongoDB lost the ability to control the data when written to the storage memory. MongoDB has different package when it comes with *Mongod*, *Mongo*, and *Mongos* which are the significant parts in the MongoDB package [58]. The main daemon process for the MongoDB system among them is the ***Mongod***. It handles the request for the data and provides all the information for managing the data and the accessibility to them. The ***Mongo*** part of the package is responsible about the client information and representation. It gives the user the ability to update and query the data and administrative operations. ***Mongos*** part handles the data routing among the shards or what is known as (MongoDB shard). It controls the flow of the queries to the shards and determines the locations of the data within the shards [59].

3.1.2 MongoDB Data Model

As we mentioned before, the primary data representation in the MongoDB is the JSON Files (BSON), an example of this kind of data is shown in the Figure 3.1 below. There is a unique identifier called "empId" in a document inside the collection of the JSON file to prevent the replication of the information. Those identifiers should have specific features [59]:

- They should be unique (not repeated).
- It is not possible to insert an empty key unless to determine the end of the key.

```
[{
  "empld":1001,
  "firstName": "jonh",
  "lastName": "springer",
  "title": "Engineer",
  "city": "Mumbai",
  "street": "Fadke Street",
  "ZipCode": "426357",
  "privateMobile": "2564875421",
  "privateLandline": "251201546",
  "salary": 150000
},
{
  "empld":1002,
  "firstName": "rafa",
  "lastName": "springer",
  "title": "Engineer",
  "city": "Mumbai",
  "street": "Fadke Street",
  "ZipCode": "426357",
  "privateMobile": "2564875422",
  "privateLandline": "251201542",
  "salary": 350000
}
]
```

Figure 3.1. MongoDB JSON Document

The MongoDB is a sensitive case when it comes to documents information. The example below is for this case, and it shows two different documents. MongoDB can store many shapes of documents within a collection.

```
{"Name": "mustafa"}

{"name": "mustafa"}
```

3.1.3 MongoDB Query Model

There are several types of the query within the MongoDB package. MongoDB Query Model allows executing different queries over all the parts (documents) of a collection such as arrays or any embedding objects. There are many features within the query model used for a response of query depending on the parameters used in the query. Those features include [60]:

- Key-value query: This is often used as the primary key for a document.
- Comparators: Those are used to compare the parts of queries within a range. From those operations, we have the higher than (>), greater than or equal (>=), less than (<) and less than or equal to (<=).
- Logic operations: All the conditions are supported by specific representations such as: and, or, equal, not equal).
- Aggregation queries: The well-known operations are from any modern programming languages (*count*, *minimum*, *maximum*).
- Sorting queries.
- Group by: It is the same as the group by a command in SQL programming languages.
- Map Reduce: This is the main query used in NoSQL databases to handles the large volumes datasets.

3.1.4 CRUD in MongoDB

This section concentrates on the performance of MongoDB in the light of the most critical four operations which are used in every database: create, read, update and delete. The operations that are regarded as modification operations are create, update and delete, whereas the read operation is considered a query. In MongoDB, it is a query which tackles a particular document within a collection. A specific field from a document, a MongoDB query, may use a projection to specify that field. When reading from a particular field in a document, a projection may be used by a MongoDB query to limit that field. MongoDB is bundled with several kinds of queries. The query model of MongoDB permits queries over all documents inside a collection including embedded object and array [61].

MongoDB clarifies several query behaviors as follows:

1. Mongo DB queries address one collection.
2. Queries in Mongo DB can be changed to put limits, sort orders, and skips.
3. A sort method should be adopted to get the order of documents by a query.

3.1.5 Mongo DB Aggregation

The aggregation process means the processing of data returning of the computed results of that data. From several documents we collect values, any data similar to other several operations are applied, and we get one result. MongoDB gives a broad set of aggregation operations. The documents within collections are the input to the aggregation operations, and the results are returned.

3.1.6 MongoDB Features

MongoDB is very powerful, and it is considered as a high-performance computing database system, and it has more flexibility in document orientation. Moreover, it is an open source software supported by a big community. It stores the data in a way different from the relational database system, in addition to tables, it uses collections and documents which are used for complex queries and fast retrieving. Besides, MongoDB is a scalable database that has additional indexing mechanism which includes TTL indexes and geospatial indexes which are used in geographic applications [57]. The configuration is easy in MongoDB, and the storing mechanism can handle data with large sizes easily. It is scheme less which is right in the failure cases that make the administration in such cases easy to handle and overcome. The new version of MongoDB contains a storage engine. The new features in this version are [62]:

- **Connector for Business Intelligence and Visualization Tools:** This feature gives the ability to MongoDB to visualize the data to the user in business intelligence tools such as Tableau. For doing that there is a need to give MongoDB the possibility to connect to a data center in tabular form. Making this with MongoDB is hard. Because of that, they made it in the new versions. The main purpose behind making it is to connect MongoDB servers with the BI tools without the need to store any data. To have a good understanding look at the Figure 3.2 below.



Figure 3.2. The BI connector in MongoDB [63]

- **Encryption at rest:** This means the ability of encryption the data when it is in the storage, and also it is one of the main features related to the security in MongoDB enterprise and NoSQL databases with powerful encryption algorithms, whereas we do not have it in the open source version.
- **Document validation:** One of the latest version updates is supporting document validation in MongoDB, which helps in decreasing the overload and effort for both users and companies during the verification process for data.
- **Dynamic Lookups:** It is part of the aggregation of the MongoDB, it helps in modeling the data with more flexibility.
- **Schema Visualization:** The new ACID graphical interface for MongoDB helps the users to analyze and manipulate the flow of the data and comparing the values among them with additional feature related to the performance and the security of the MongoDB databases.

3.1.7 Replication in MongoDB

The replication is a process related to the *Mongod* in the MongoDB package, and it is used to maintain the same data by replicating it to many sources [64]. MongoDB supports replication similar to the master-Slave replication configuration, but not the same. The good point here is that there is a recovery step in case a failover which appears in the primary node or the connection lost between the primary node and secondary node. This procedure is done automatically. The MongoDB recovers the

connection by converting one of the secondary nodes to a primary node. By doing that, we will assure the availability and redundancy to our database system. The replica set can extend to N number of nodes in a cluster, but it cannot have more than primary node. And by that the writing operations will be limited to the primary node only. In general, the replication means when a primary node receives data, it copies the data and replicates it to the secondary nodes [64]. Figure 3.3 below presents the replication.

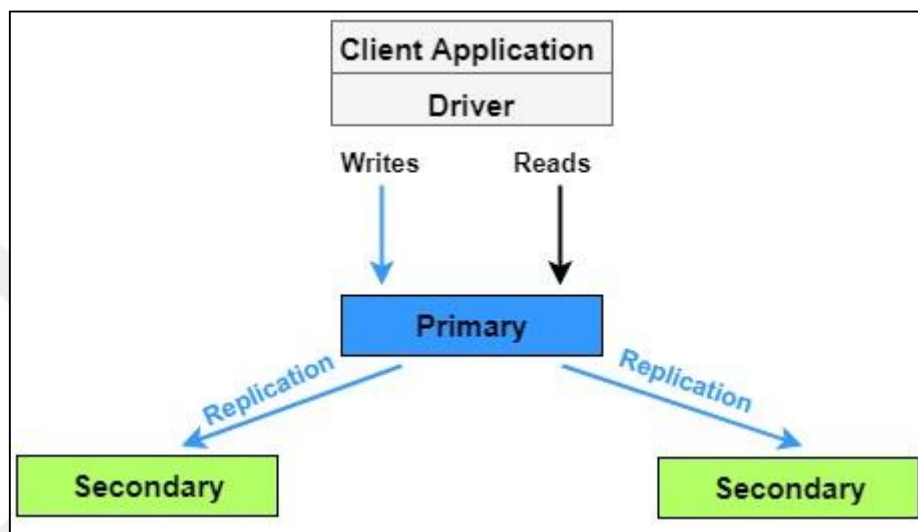


Figure 3.3. Replication in MongoDB

3.1.8 Sharding in MongoDB

MongoDB presents a sharding procedure. It can be characterized as a way of putting away data on a few machines. And it will provide a good position with rapid incrementing of data. A separate machine which is utilized to store data may not be appropriate because of low throughput between the read/write processes. Sharding overcomes this issue by making use of even scaling. The MongoDB bunch includes three parts:

- Mongos (routing server)
- Shards
- Configuration server

To support sharding, MongoDB designs the Sharding which means the process of breaking a massive volume of data into smaller volumes to overcome the restriction of hardware. Bottlenecks in RAM or disk I/O are regarded instances of hardware restrictions. MongoDB consequently modifies the data in the sharded cluster as the data increases to the extent of the group increments or abatements. Sharding, unlike in social databases, is programmed and incorporated into the database, it minimizes the weight for designers and operations groups. Each shard consists of a replica set, and we make use of the shards to store real information. Expanding the number of nodes inside each shard prompts will expand repetition and accessibility. Configuration servers (mongod) are used to hold metadata that includes mappings for the original data in the shards. Directing servers use metadata to course operations to particular shards. MongoDB disperses data or shards at the collection level. Sharding specifies a collection's data with the shard key [65]. The first step to shard a collection can be achieved with a Sharding key. A shard key is not different from ordering. A shard key is partitioned into chunks equitably over the shard by MongoDB. Every chunk consists of a few numbers of records altogether. The essential benefit of the pieces lies in modifying the shards. In the case that a shard estimate becomes more substantial than the other shards, a few substances of the piece will be relocated to other smaller shards to rebalance the sizes of the shards. In the event in which another node is contained or expelled from the cluster, the pieces will also redistribute the information over the group. In Figure 3.4, the sharding architecture in MongoDB is presented.

The configuration (mongod) holds the metadata, and it is responsible about routing the data to the shards. MongoDB distributes the data to the shards, and the shard will generate a shard key for the data. The shard key is like an index to this data. Then the shard key shares across the shard using MongoDB, the shard contains many chunks. Each chunk contains many ordered documents. This procedure offers to balance to the shards by migrating the shards with smaller size when a shard grows more substantial than the other shards. When a node is removed or added to the cluster, the chunk will resend the data among all the other nodes [65].

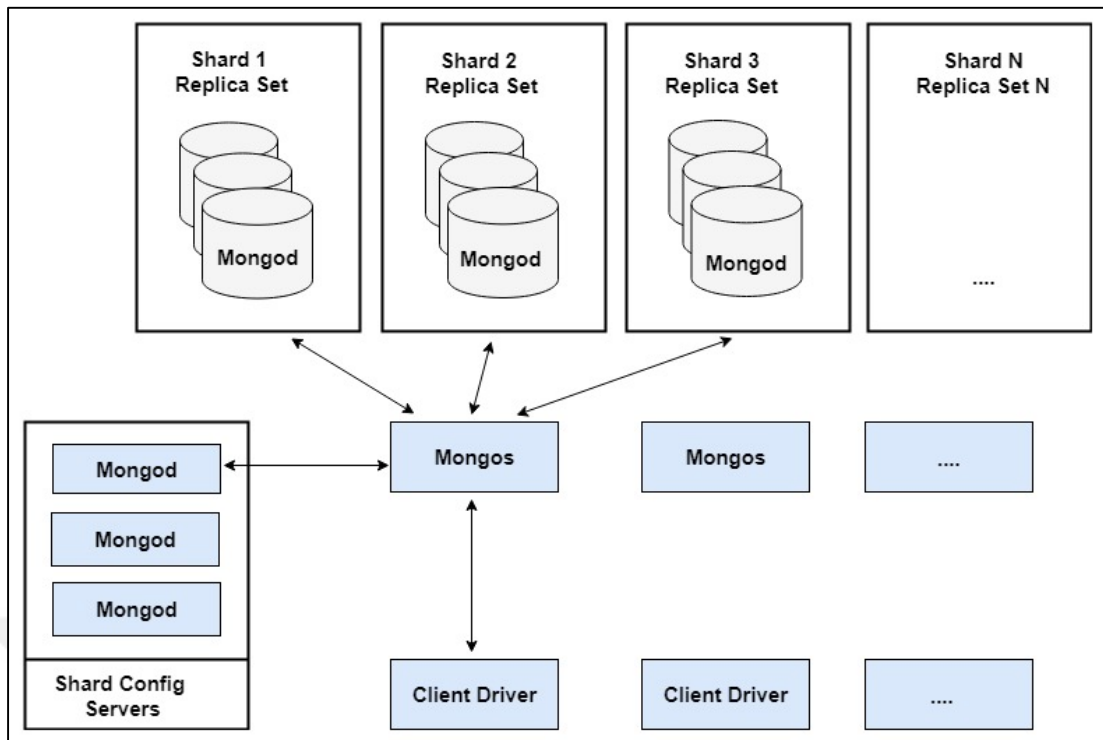


Figure 3.4. Sharding mechanism in MongoDB

3.1.9 Failure Handling in MongoDB

The excellent feature in MongoDB is the automatic failover recovery. If a node crashes in MongoDB cluster, we will face a state of losing the data. Even if the node return to work later, still we can get corrupted data. This problem is related to many reasons such as the hardware failure or problems in connection. When a node fails, the replication will recover the last update to that node and make it online again. Still, we have the terrible cases which cannot fixe with the failure handling. One of these cases is that when all the nodes fail within a shard. In this scenario, the MongoDB cannot make any operation in that shard. The second scenario is when the configuration servers fail. Then we lose the update of the data between shards.

3.2 Apache Cassandra

Apache Cassandra is NoSQL databases with high scalability and the data store of the Cassandra in from the fault-tolerant type. It is developed in Java programming language by Facebook to handle the search operation inside the Facebook inbox [66]. The first release was in 2008, and it took two years to be the head of Apache

company projects in 2010. The primary reason to build Cassandra was to give the ability to help in the solving the Index search problem [67].

3.2.1 Apache Cassandra Architecture

Unlike the other NoSQL database systems, Apache Cassandra has peer-to-peer architecture. There is a need to know master-slave architecture to understand the peer-to-peer architecture. In the master-slave architecture, when the master distributes data and controls the flow of the data to the other slave nodes, the slave needs to synchronize the data with the master to keep up-to-date data. However, when the connection is corrupted, or the master-slave fails the system start to lose data, and in the best cases, it will recover after a while which means loss of data. In Cassandra, the peer-to-peer architecture helps the node to control itself and work as similar to a master node at the same time. That gives the Cassandra the ability not to fail since there is no master node in it. Moreover, because of the peer-to-peer, the performance improves more than any other database in the previous versions [67].

3.2.1.1 Key Space

Apache Cassandra keeps all the information about the Metadata internally in a key space to help in providing the cluster with the needed information for all the operations in it [68]. Each node contains a Metadata. This Metadata is stored there by the aid of the key space, and the Metadata contains other information which includes [68]:

- the name of the cluster
- the token of the node
- the information about the transferred data
- the information about the schema and the key-space definitions
- the node's bootstrapping

3.2.1.2 Commit Logs, Memtables and SSTables

The commit logs in Cassandra help to improve it. The commit logs are responsible for the writing operation when a writing operation appears the commit logs record it in case a failure happens to recover it later. Other than recording it in the commit log, the writing operations are considered unsuccessful. Moreover, they can be recovered by the MemTable. After the commit log records data as successful, it will be written to the MemTable which will redirect the information to the SSTable after the MemTable reaches its limit. The MemTable will be created after that and stores the data from the SSTable, after that the SSTable will flush itself [69]. See Figure 3.5 below.

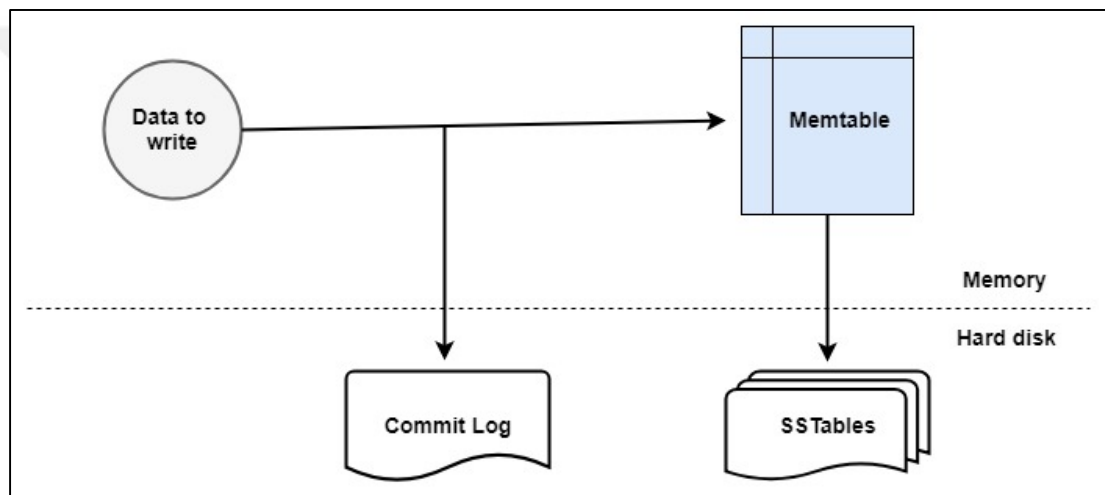


Figure 3.5. Writing operation in Apache Cassandra

Each memTable has a threshold value to decide the need of flushing when it reaches it. To control the limit amount copy of data in Cassandra, we need to set a value to the replication factor, and this factor is not associated with the number of cluster nodes. This procedure helps Cassandra to achieve durability and scalability. For the consistency, there are many levels of it in Cassandra. The part responsible about the levels is called *a quorum*. The consistency controls the amount of data replication before sending a response to a user request [69]. There are 5 levels (0, 1, ANY, QUORUM and ALL) [68]. The quorum value can be decided by [69]:

$$Quorum\ Value = Replica\ Number / 2 + 1$$

3.2.1.3 Hinted Handoff

This feature in Cassandra Architecture provides availability. When a node fails in the cluster, the other nodes around it starts to acquire data temporarily. When the corrupted node returns to work, it starts collecting and reallocate the temporary data inside it. This helps to avoid the loss of the data. [70].

3.2.1.4 Compaction

This operation is responsible for freeing the space in the SSTable by shrinking the data to shift this space to the end of the memory. This procedure is merging the SSTable and moving the keys, combining the columns and neglect the tombstones. The tombstone is the value associated with the delete operation and is responsible about recording the last delete operation which has been done on a record [68]. The SSTable also helps not to delete any record easily. Rather than deleting it after a delete operation appears, Cassandra will consider this record as an update and keep it in the ram within a tombstone. The whole procedure can be controlled by the value of the flag for this operation. This operation helps in reducing the space on the storage to receive a new data [68].

3.2.1.5 Bloom Filter

In 1970 Burton bloom invent the bloom filter. The bloom filter is a probabilistic data model. This filter searches for a record existing in the SSTable in Apache Cassandra by using a fast non-deterministic algorithm. It works like a cache memory for fast search operations, and this boosts the operation with the large volume size data [71].

3.2.1.6 Staged Event-Driven Architecture (SEDA)

With the help of SEDA model, the Cassandra can divide any set of operations into many stages to execute each stage. SEDA is suitable for simplifying the construction of the big data and can handle a massive amount of data quickly. "SEDA stages are composed of three components: event queue, an event handler, and an associated thread pool "[68].

3.2.2 Data Model

The representation of the data model in Cassandra is distributed into multi-dimensional tables. The tables contain keys as indexes and rows with a fixed size. The shape of a cluster in Apache Cassandra is a ring. Moreover, each cluster has its key space. Each key space should have a name and attribute to do the key space work against the other keys just as similar as the keys in relational databases. The key space consists of the features below [72]:

- **Replication Factor**

Replication factor is responsible for choosing the number of nodes to store data at the same time. If we have a replication factor of 4, then it means that four nodes will receive the rows of data and store them. The replication factor is vital to achieving high consistency.

- **Replica Statement Strategy**

There are two strategies for replication in Apache Cassandra: the *Simple Strategy* also known as (Rack Unaware Strategy) which is used for one rack and one data center, *Network Topology Strategy* known as (Rack Aware Strategy) which is used when one needs more than one data center. Also, it is recommended for a future extending plan for data centers [73].

- **Column Family**

The column family contains the rows of data as similar to the tables in the relational database systems. The row consists of ordered columns. The whole structure of the data is a container (Column Family). A column family can be associated with more than one key space. The column family has three main features: *name*, *value* and *time stamp*. The row value is named as a validator, and the column value is named comparator. There are many data types in Apache Cassandra. See Figure 3.6 below.

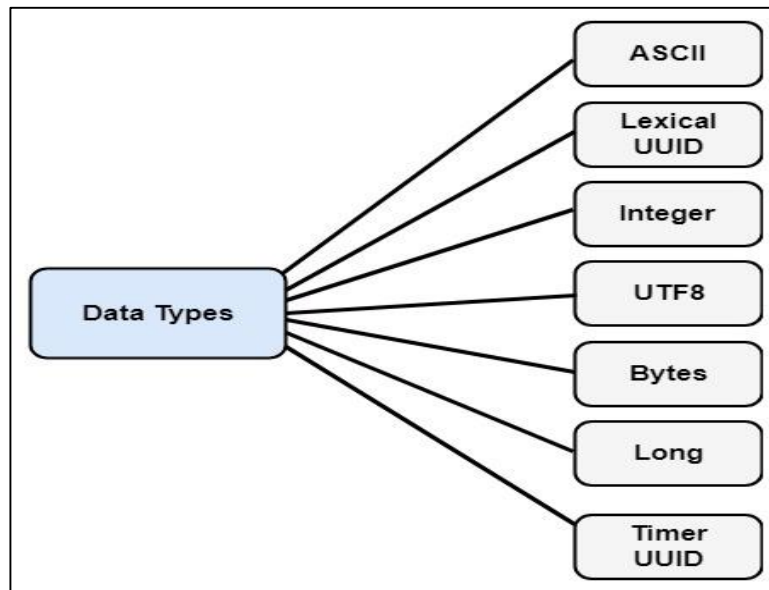


Figure 3.6. Data Types in Cassandra

3.2.3 Apache Cassandra Features

Just similar to the hash table data structure, Cassandra is considered as a key-value store. Cassandra has a decentralized design which means any node can do read and write operation separately. This mechanism is good to avoid the node failure. See Figure 3.7. The Read/Repair method provides Cassandra with high availability. And there is no need to match column with a row like the relational database systems.

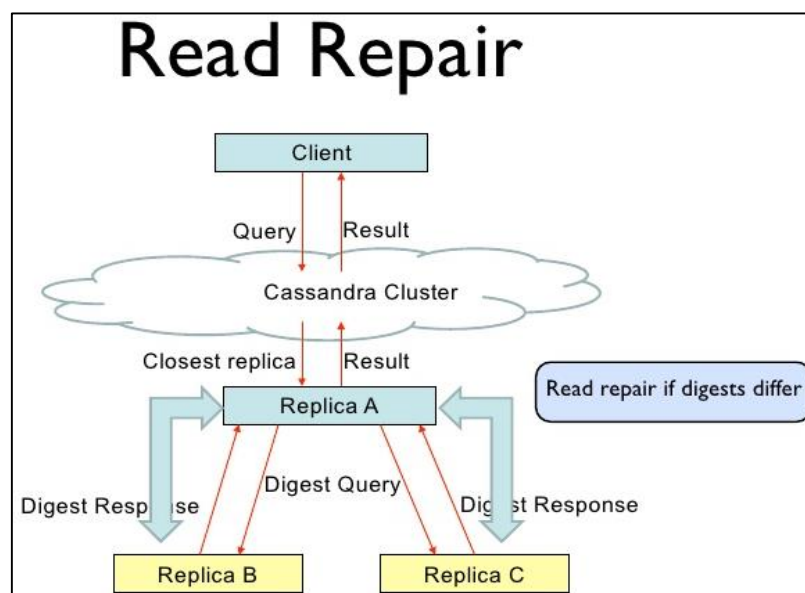


Figure 3.7. Read/Repair node procedures in Cassandra

The nodes can communicate with the gossip protocol [74]. The gossip protocol is responsible about to controlling the sending and receiving the data from and to the cluster. Cassandra is an AP software according to the CAP Theorem. However, it has an amount of consistency that can unbalance the relation between availability against data accuracy. Since Cassandra is part of the Apache Family it can integrate easily with Apache *Hive* [75], *Map Reduce* and Apache *Pig* [76] to make a complete set for the CAP Theorem.

3.2.4 Fault Tolerance in Cassandra

As we mentioned before, Cassandra is a peer-to-peer architecture with ring connection. Every node can read and write the data. Depending on the replication factor value, the nodes with the same number of the value will receive the same data. However, even with that, there is a node without a key called the zookeeper. The system selects the zookeeper. The nodes that will receive the keys are equal to $(N - 1)$ where N is the number of total nodes. If a node fails, the other node will receive the data until it returns to work again. Then it will start to collect the data from the nodes. In case a leader node fails then another node will be selected as a leader. The fail tolerance is limited to hardware fails or natural accidents [77].

3.3 Security in NoSQL Databases

The founders of the NoSQL focused on the performance rather than the security as clarified in the previous chapter. Moreover, when the users start increasing their feedback about the weakness of the big data some providers start considering the security features. Unlike the RDBMS, the security is interconnected with the performance of the NoSQL databases. However, at the same time, it is weak because NoSQL databases miss the centralized controller by the user.

3.3.1 Cluster Security in NoSQL

The security in NoSQL model is similar in design to the RDBMS but different in work. Some security tools can be found as a built-in feature in the NoSQL databases. We have the secure communication connector SSL/TLS, authentication with Kerberos,

data at rest and data at motion transparent encryption. The main problem in the security of NoSQL databases is the cost which is expensive. See Figure 3.8 [78].

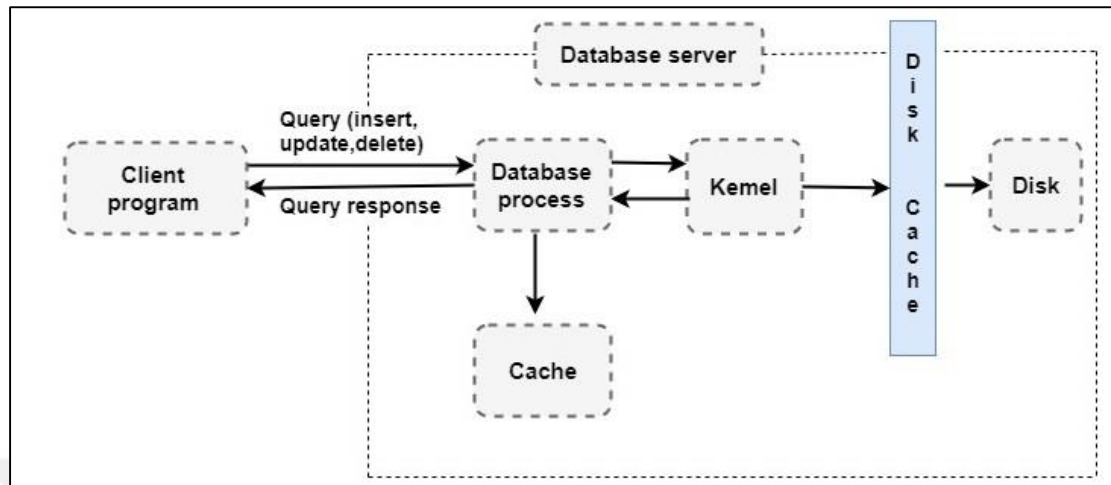


Figure 3.8. NoSQL Security Model

3.4 Authentication

In the beginning, the NoSQL databases did not contain proper support for authentication [79], but nowadays most of the NoSQL databases contain the authentication method. This functionality is responsible of specifying the user data through logging process or when we all know the username and password. The principal activities of this part are the following [80].

1. Providing the user with the information about the availability of the username that he picked to avoid creating the same login access information to the different user.
2. Creating centralized access to the users on the database itself. And prevent the unauthorized user from logging the system.
3. Giving the constraints about creating the password and what it need to create a suitable secure password. Like the minimum characters for the password and the level of complexity.

3.4.1 Authentication in MongoDB

There are several algorithms for authentication in MongoDB. In the open source edition, we have the SCRAM which stands for (Salted Challenge Response Authentication Mechanism). The SCRAM is the current default authentication Mechanism in MongoDB, and it uses the SHA-1 Encryption Algorithm. It is used to encrypt names, passwords and database authentication by comparing the user and its password that are associated with a specific database [81]. The second one is the MongoDB-CR (Challenge Response). It was the previous default Mechanism until the version 3.0. It was used to identify the users by their passwords [82]. After the 2.6 versions, the MongoDB made another authenticate mechanism called the x.509 Certificates. It is used to authenticate the users to login the database by using the certificates through the TLS/SSL connection rather than users names and passwords. To use this method one needs to generate a valid certificate with a single authority. The certificate should contain two fields (`keyUsage`, `extendedKeyUsage`) which contain the digital user signature and its authority [83]. In the enterprise, we have additional mechanisms to the previous methods. We have FIPS which stands for (Federal Information Processing Standard) which is a security standard to the US computer security criteria to all the software that encrypts and decrypts the data securely. There is a need to have the OpenSSL library to use the FIPS 140-2 [84]. Another mechanism is the Kerberos which is offered to *mongod* and *mongos*. It is a protocol for industry authentication. It is used for large size client-server systems. In this mechanism, each user has to use authenticated communication, and this is called the *principal*, and the principal should have a unique name. This procedure is done by the KDC (Kerberos Key Distribution Center) which contains the principals, and each principal is related to a secret key. When a user sends a request to obtain the *ticket* to access, the KDC checks the client's secret key and builds a connection after checking the server's secret key, and it keeps the secrets hidden for both sides (client/server) [85]. The enterprise edition also provides another mechanism for proxy authentication. Administrators use this feature. They manipulate the user's requests to the cluster and proxy their authentication to LDAP (Lightweight Directory Access Protocol) [86].

3.4.2 Authentication in Apache Cassandra

In Cassandra, we have fewer mechanisms than in the MongoDB. We have in the open source version the standard authentication only by using the usernames and passwords login which are called the Internal Authentication [87]. In the enterprise edition, we have three mechanisms: Internal Authentication, LDAP and Kerberos [87]. Also, those are the same mechanisms explained above in the MongoDB authentication.

3.5 Authorization

The authorization is the process of giving the users the accessibility to a specific part of the database which means giving the user the limit access to the objects in the database by making constraints and roles to the users for those objects. To let them reach only the guaranteed data for those users without showing them unnecessary or private data for other users [88].

3.5.1 Authorization in MongoDB

The authorization in MongoDB is done by the RBAC (Role-Based Access Control). The RBAC gives a role or more to the user to limit his access to the resources of the database system and anything outside the roles is specified to the user, the user cannot access. The same mechanism is used by the enterprise version of MongoDB [89].

3.5.2 Authorization in Apache Cassandra

In Cassandra, we have the same as the MongoDB RBAC with the same functionality. Also, there is RBAC in the enterprise. In the Cassandra, this feature is not enabled by default. The Cassandra.yaml file can easily be configured and uncommented the two lines:

Authenticator: `com.datastax.bdp.cassandra.auth.PasswordAuthenticator`

Authorizer: `com.datastax.bdp.cassandra.auth.CassandraAuthorizer`

Cassandra provides more flexibility, we can easily enable both the authentication and authorization, and we can use authentication without authorization to give free space to the user to work with no constraints on them [90].

3.6 Auditing

The auditing can be recognized as the system mechanism recording the user activity in the system and monitoring them when needed [91]. It also helps in the security to detect the possible cracking in the user's passwords [92]. Auditing is the ability to detect any attempt to insert unauthorized data to the database and the way of detecting that. The auditing securing can be done by recording all the request to the database and saving them in logs and trails to detect them and filter them. The Auditing can also be done through the changes in the configuration files for the databases not only in the data [88].

3.6.1 Auditing in MongoDB

Unfortunately, there is no auditing in the open source version of MongoDB, whereas in the enterprise edition we have many features for auditing [93]:

- **Schema (DDL):** In this feature, the MongoDB records the Data Definition Language queries such as creating and deleting the database objects and databases.
- **Replica set and sharded cluster:** This means recording the replication and sharding operations and configurations.
- **Authentication and Authorization:** Recording the authentication and authorization information and logs within the clusters and databases are needed.
- **CRUD operations:** Recording all the operations we make on the database information (create, read, update and delete) and records.
- Moreover, the activities between the clients and databases.

3.6.2 Auditing in Apache Cassandra

Just like the MongoDB, Cassandra open source version does not contain auditing property. In the enterprise, we have the following features [94]:

- **AUTH:** Recording the login operation to the database.
- **DDL:** The logs files of the Data Definition Language operations.
- **DML:** The logs files of the Data Manipulation Language operations.
- **DCL:** The logs files for the Role-Based (RBAC) operations.
- **QUERY:** Records all the queries that have been done in the database.

3.7 Transport Encryption

The data always transports from the user to the database. However, here the transportation is encrypted to save the data from unauthorized users to see its content. Encryption, in general, has several ways. The Transport Encryption and Encryption data at rest. The transport encryption is associated with the connection while the encryption in data at rest is when the data in the storage [88]. We will cover the transport encryption only.

3.7.1 Transport Encryption in MongoDB

In the open source version, MongoDB supports TLS/SSL (Transport Layer Security) and (Secure Sockets Layer). Those algorithms use the OpenSSL libraries. Also, they make sure that the connection can be readable by only the intended user. They encrypt the connection key of 128-bit of length. In the enterprise edition, we have the same method with an additional one, the FIPS with the minimum of 128-bit length key [95].

3.7.2 Transport Encryption in Apache Cassandra

In Cassandra, in the open source edition, we have an encryption method, encryption with SSL. It can work in hardware and software. It can be a node or cluster or peers. The endpoints need to transport the information between them to build the trust in

the connection, and then the keys need to be generated. The public key will be shown to the entities of the connection, and the private keys will be stored on the server until it receives a certificate from the user. Then it will check the private key that is associated with, and it will provide the permission for the connection [96]. Among all the other version of MongoDB and Cassandra, the enterprise has a various number of JCE cipher algorithms [97]:

- AES/CBC/PKCS5Padding. With different keys: 128, 192, and 256.
- AES/EBC/PKCS5Padding. With different keys: 128, 192, and 256.
- DES/CBC/PKCS5Padding. With 56-bit key.
- DESede/CBC/PKCS5Padding. With different keys: 112, 168.
- Blowfish/CBC/PKCS5Padding. With keys: 32, 40, 56, 112, 128, 168, 192, 256, and 448.
- RC2/CBC/PKCS5Padding. With different keys: 40, 56, 112, and 128.

The default algorithm is the AES/CBC/PKCS5Padding with a key length of 128-bit.

3.8 Encryption at Rest

We mean by this to protect the data which is not moving through the traffic in database associated Network. We can conclude that any data is either a computer or any terminal machine that is not moving to another entity like node or cluster which is called Data at Rest. Moreover, it also means that the data is not active or it has any process on it [98].

3.8.1 Encryption at Rest in MongoDB

There is no encryption at rest in the open source version. In the enterprise edition MongoDB supports many algorithms to encrypt the data:

- AES256-CBC (Advanced Encryption Standard in Cipher Block Chaining mode) via OpenSSL

- AES256-GCM (Advanced Encryption Standard in Galois/Counter Mode)

Both algorithms work with 256-bit keys only. We can run them either locally on our machine or remotely through KMIP (Key Management Interoperability Protocol) server [99].

3.8.2 Encryption at Rest in Apache Cassandra

We do not have it in the open source version. In the enterprise, we have six algorithms which work with the different length of keys and can run locally and remotely through KMIP server [100]:

- AES/CBC/PKCS5Padding with keys: 128, 192 and 256.
- AES/ECB/PKCS5Padding with keys: 128, 192 and 256.
- DES/CBC/PKCS5Padding with 65-bit key only.
- DESede/CBC/PKCS5Padding with keys: 112 or 168.
- Blowfish/CBC/PKCS5Padding with keys: from 32 to 448.
- RC2/CBC/PKCS5Padding with keys: from 40 to 128.

3.9 JMX Authentication

JMX is a term stands for (Java Monitoring Extension). It is a technology for monitoring the checks of the progress of the resources depending on the JVM (Java Virtual Machine). This technology provides a deep controlling for the resources that have been built by Java programming language. It uses the MBeans (Managed Beans) to collect all the information, and those beans are Java objects that are laid in MBean server. There is no support for JMX in MongoDB at all. Datastax provides this tool in Apache Cassandra [101].

3.10 Authentication Caching

It is another technology that is supported in Apache Cassandra. The cache is temporary storage to store data for quick response with the limited size of storage for only the important information. The caching in Cassandra is storing the roles of the user login and related information. The caching helps to not authorize the user many times during work in the database. It limits many logins at a time without the need to do the authorize procedures many times. The default value for a user role to stay in the cache is measured with milliseconds, and the roles stay in the cache for 120000 milliseconds (2 minutes) as a default. The same time for updating is needed. For the permission to a user to do these procedures is 120000 milliseconds and to update it 2000 milliseconds. As the maximum number of entries allowed in the cache depends on RLAC (Row-Level Access Control) and the formula below gives the right size to the data to be cached [102]:

$$\mathit{numRlacUsers} * \mathit{numRlacTables} + 100$$

3.11 Proxy Roles

Just like the authenticating caching, the proxy rules are another security feature counted to Apache Cassandra except it is in the enterprise version only. The proxy rules allow the user to log to a server and execute the CQL (Cassandra Query Language) commands in the application layer. This technique means that this server is responsible about transferring the commands to the database server and replying to the user requests without the need to log them to the database server. As an outcome, it helps to secure the data and limit the authorization to the user in the command [103]. See Figure 3.9 below.

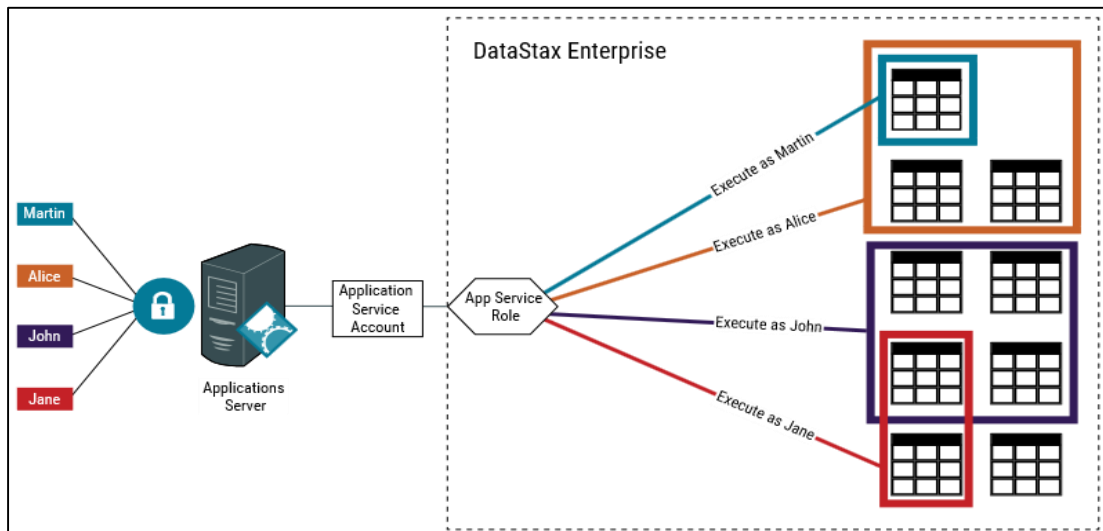


Figure 3.9. Proxy Roles in Cassandra

3.12 Node-to-Node encryption

Using SSL (Secure Sockets Layer), we can secure the data that transfers between nodes in a cluster. There is no such mechanism in MongoDB open source version, but the situation is different with the enterprise copy. In MongoDB enterprise, we use the same techniques and algorithms we have mentioned earlier in the Transport encryption part with a minimum of 128-bit length for the key [104]. In Cassandra, all the nodes should have an SSL certificate, and they should modify the `cassandra.yaml` file by enabling the `server_encryption_options` line for each node and there is a need to set the key store and the trust store. After that, we restart the Cassandra to update the changes [105].

3.13 Client-to-Node encryption

This mechanism is to encrypt the data which can be found in all the versions of MongoDB and Cassandra. This technique is about securing the data between the user application machine and the cluster of the database. This technique depends on the SSL by securing the connection channel between the wanted node in the cluster and the user (client). MongoDB deals with this technique in the same way as the node-to-node encryption with a change in the 3.2.6 version. By checking for a certificate for a user identity with the data allocated in a `.pem` file that contains the root certificate chain to check whether the user is the right user by checking his identity that already

has been set by the CA (Certificate Authority). This certificate should be valid in the system the moment user used it to log the cluster [104]. In Cassandra, it is less complicated. All we need to do is to enable `client_encryption_options` on each node, and, and then we restart the Cassandra [106].



CHAPTER 4

TEST ENVIRONMENT

In this chapter, the details of MongoDB and Cassandra testing environments will be given. From the installation of databases, configuration parameters are chosen to set up single and cluster nodes test, and network setup.

4.1 Hardware 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
- 200 GB of ephemeral storage in each unit
- Ubuntu 16.04 LTS (64-bit)

4.2 Software Features

Several software components were used to produce the results presented in this study. They are as follows:

4.2.1 Yahoo Cloud Serving Benchmark (YCSB)

YCSB is an open source tool developed by Yahoo labs [107] used in benchmarking the performance of several databases, such as HBase, OrientDB, Redis, MongoDB,

Cassandra, and Hypertable. The tool was first introduced in 2010 by Brian F. Cooper [108] as a tool to test cloud base services, and then furtherly developed by the team to include several SQL and NoSQL databases. Figure 4.1 below shows YCSB tool architecture. The tool is typically used to stress test Read, Write, Update, Scan, Insert operations on the given database. Although there are six different workloads predefined and set by default, each workload can be redesigned to fit any test required by the user. For example, the read and write percentage performed by YCSB tool can be reconfigured with an ability to choose the size of record and operation counts.

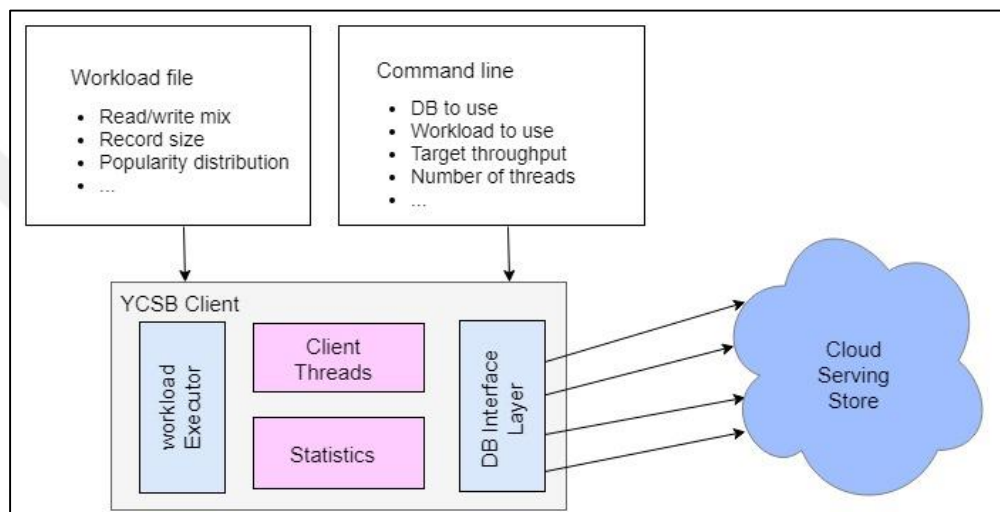


Figure 4.1. YCSB Architecture

Installation of latest version YCSB tool (0.12.0) is a straightforward process, which is explained as follows:

```
terminal> curl -O --location
https://github.com/brianfrankcooper/YCSB/releases/download/0.12.0/ycsb-0.12.0.tar.gz

terminal> tar xfvz ycsb-0.12.0.tar.gz
terminal> cd ycsb/bin
```

The following should be defined and passed as arguments to YCSB tool to run a workload:

- 1 – Load/Run: Load is used to load the data into the designated databases, whereas Run is used afterward to start the defined test.

- 2 – The name of the database is like mongodb for Mongo database, and cassandra-cql for Cassandra database.
- 3 – Workload name is such as workloada, workloadb.
- 4 – Remote host IP address where the database is hosted.

For example, to run a stress test using workload B against a Cassandra database, we used the following:

```
terminal> ycsb load cassandra-cql -s -P workloads/workloadb -p
hosts="192.168.56.101"

terminal> ycsb run cassandra-cql -s -P workloads/workloadb -p
hosts="192.168.56.101"
```

It is important to note that `-s` sets YCSB to statistically reports its output on the terminal, `-P` to define a workload, and `-p` to pass the remote IP address.

4.2.2 Mongo Database (Open Source)

Mongo database is one of the two databases tested in this study. Single node installation and configuration is a simple procedure. However, cluster configuration requires more attention to network and building cluster entities. For this study, we used MongoDB 3.6.3 installed in a machine hosting Ubuntu 16.04 LTS (long-term supported version). According to Mongo website [109]. The required steps to install MongoDB are elaborated below.

Step 1 – Adding MongoDB keyserver and package source list:

```
terminal> sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv
2930ADAE8CAF5059EE73BB4B58712A2291FA4AD5

terminal> echo "deb [ arch=amd64 ] http://repo.mongodb.org/apt/ubuntu "$(lsb_release -
sc)"/mongodb-org/3.6 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-3.6.list
```

Step 2 – Run package update:

```
terminal> sudo apt-get update
```

Step 3 – Install MongoDB:

```
terminal> sudo apt-get install mongodb-org=3.6.3 mongodb-org-server=3.6.3 mongodb-  
org-shell=3.6.3 mongodb-org-mongos=3.6.3 mongodb-org-tools=3.6.3
```

Step 4 – Start MongoDB server:

```
terminal> sudo service mongod start
```

After installation, we must prepare the database to receive YCSB benchmark data by creating a database named "ycsb," while YCSB tool will create the required collection inside ycsb database named "usertable."

```
terminal> mongo --host 95.183.187.144  
> use ycsb;
```

4.2.2.1 MongoDB Cluster Installation

Throughout documentation site [110], MongoDB cluster is divided into three main entities, see Figure 4.2 below.

- 1 – ConfigSvr: Config servers store metadata and configuration settings of the cluster are used to map the requested data by client's applications to the shard node which contains it. Such data can be retrieved reliably and consistently.
- 2 – Database Router (referred to as Mongos): The DB query routers are the MongoDB instances responsible for carrying applications data queries and building the right responses from the appropriate shards nodes through consulting cluster's Config servers. In other words, DB routers are the interface between client applications and data stored among shards nodes.
- 3 – Shards (i.e. Nodes): Shards are where all data stored, only containing a subset of the total sharded data. MongoDB is designed shards to hold data in what is called replica sets instead of a single machine which inherently allows data to be replicated among several secondary nodes to ensure data availability in the events that a primary shard node goes offline.

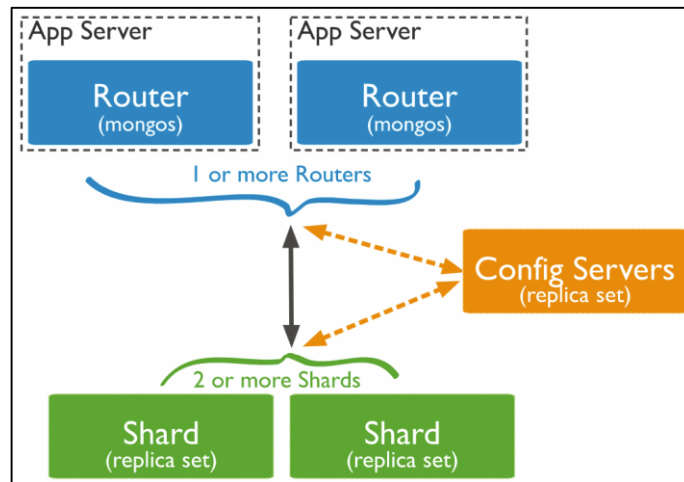


Figure 4.2. MongoDB Cluster Design

One should note that these entities are not different software packages designed to perform the duties mentioned above, but it is the MongoDB software package performing different roles defined in the database configuration file; found under `/etc/mongo/mongo.conf`. For example, to configure Shards, the parameter "ClusterRole:" should hold the value of `shardsvr`, whereas the same parameter for the config server should be set to `configsvr`.

For to test the performance of MongoDB cluster, we configured a cluster containing six MongoDB nodes; four shards, one DB router, and one ConfigSvr. Figure 4.3 gives an overview of cluster layout used in this study.

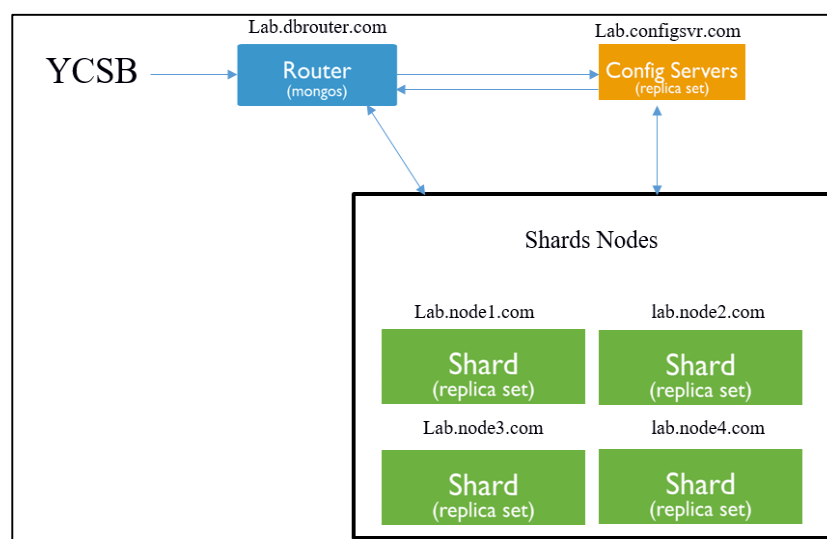


Figure 4.3. MongoDB Cluster Design Overview

The installation of Mongo databases on each node was done according to the steps explained in Section 4.2.1 above, however, starting the cluster was accomplished as follows:

Step 1 – Start the ConfigSvr:

```
Terminal> mongod --config /etc/mongod.conf --fork
```

Step 2 – Start every Mongo database in every Shard node:

```
Terminal> mongod --config /etc/mongod.conf --fork
```

Step 3 – Start and connect the DB router to cluster ConfigSvr:

```
Terminal> mongos --configdb "replConfig/95.183.187.142:27017" --bind_ip  
95.183.187.140
```

Step 4 – Login to DB router from another terminal and start adding Shard nodes using sh command:

```
mongos> sh.addShard ("shardrep1/95.183.187.143:27017")  
mongos> sh.addShard ("shardrep2/95.183.187.145:27017")  
mongos> sh.addShard ("shardrep3/95.183.187.144:27017")  
mongos> sh.addShard ("shardrep4/95.183.187.146:27017")
```

Step 5 – Confirm the cluster status:

```
mongos> sh.status ()
```

Step 6 – Create Sharded database named “ycsb” and add shard collection “usertable”:

```
mongos> use ycsb;  
mongos> sh.enableSharding ("ycsb");  
mongos> sh.shardCollection ("ycsb.usertable", {"name":1});
```

4.2.3 Cassandra Database (Open Source)

Cassandra is the second database tested in this study licensed and maintained by Apache [111]. We used the latest version of Apache Cassandra (3.11.1) along with Java 8 installed on Ubuntu 16.04 LTS according to the following instructions [112]:

Step 1 – Install Java 8:

```
terminal> sudo add-apt-repository ppa:webupd8team/java
terminal> sudo apt-get update
terminal> sudo apt-get install oracle-java8-set-default
```

Step 2 – Add Cassandra keyserver and package source list:

```
terminal> gpg --keyserver pgp.mit.edu --recv-keys F758CE318D77295D
terminal> gpg --export --armor F758CE318D77295D | sudo apt-key add -
terminal> gpg --keyserver pgp.mit.edu --recv-keys 2B5C1B00
terminal> gpg --export --armor 2B5C1B00 | sudo apt-key add -
terminal> gpg --keyserver pgp.mit.edu --recv-keys 0353B12C
terminal> gpg --export --armor 0353B12C | sudo apt-key add -
terminal> echo "deb http://www.apache.org/dist/cassandra/debian 311x main" | sudo tee -
a /etc/apt/sources.list.d/cassandra.sources.list
```

Step 3 – Run system update:

```
terminal> sudo apt-get update
```

Step 4 – Install Cassandra:

```
terminal> sudo apt-get install cassandra
```

Step 5 – Check Cassandra server status:

```
terminal> sudo service mongod status
```


Now, “ycsb” keyspace and “usertable” table should be created accordingly:

```
terminal> cqlsh 95.183.187.144
cqlsh> 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,
```

4.2.3.1 Cassandra Cluster Installation

Cassandra cluster designed by Apache provides a quick and easy way to build and configure database cluster if compared to MongoDB approach. However, MongoDB approach of clustering databases guarantees horizontal and vertical scalability of the cluster whereas Cassandra approach was designed to scale only horizontally. We followed the procedure found in Cassandra documentation website [113] and changed several configuration parameters in Cassandra.yaml file is (found under etc/cassandra/cassandra.yaml) to accommodate our cluster design. In every node, we performed the following:

Step 1 – Set the cluster name to “TestCluster”

```
cluster_name: 'TestCluster'
```

Step 2 – Set tokens_num to 256:

```
num_tokens: 256
```

Step 3 – Set rpc_address to host ip address:

```
rpc_address: 95.183.187.140
```

Step 4 – Set seed to remote seeder address:

```
- Seed: "95.183.187.143"
```

Step 5 – Configure the listening address to host ip address:

```
Listen_address: 95.183.187.140
```

Step 6 – Set data snitch to RackInferringSnitch

```
endpoint_snitch: RackInferringSnitch
```

After that, the seeder node should be started first followed by starting the remaining three nodes. It is important to note that Cassandra Seeder Node functions as ConfigSvr and Mongos (DB Router) when compared to MongoDB cluster. Figure 4.4 gives an overview of Cassandra cluster layout used in this study.

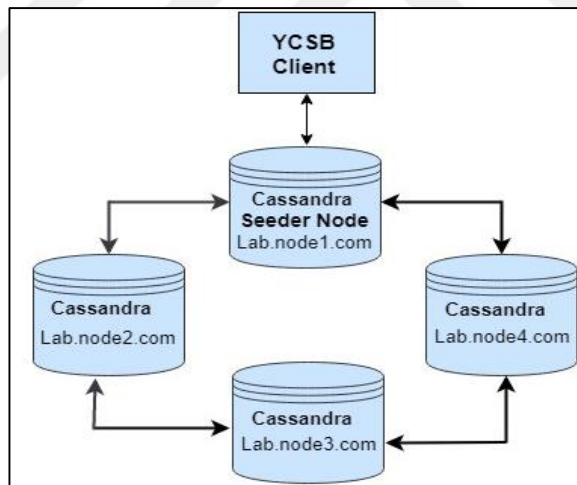


Figure 4.4. Cassandra Cluster Design Overview

4.3 NoSQL Databases Security

The second aim of this study is to compare and test the security of both Cassandra and MongoDB databases. Through reading security features of both databases from [114-116], we have found that Cassandra and MongoDB open source versions only support few security features such as authorization, authentication, data transport

encryption. Whereas the Enterprise edition of both databases furtherly supports an extensive list of security features including data encryption at rest, JMX authentication, Proxy roles, node-to-node encryption.

In this study, the security features of open source and an enterprise edition for both databases will be evaluated against ten security factors identified from [114-116], then, a general conclusion will be drawn concerning the database which provides better security features and protection of its data. The encryption benchmark (i.e., overhead) of both databases will be demonstrated using different workloads of YCSB.

4.4 MongoDB Enterprise

MongoDB enterprise edition is also available through Ubuntu package manager where the same procedure depicted in Section 4.2.2 above can be used to install all the required packages. However, the apt-get repository list and installation command are different:

Step 1 – Add the packet source list:

```
terminal> sudo echo "deb [ arch=amd64,arm64,ppc64el,s390x ]  
http://repo.mongodb.com/apt/ubuntu xenial/mongodb-enterprise/3.6 multiverse" | sudo  
tee /etc/apt/sources.list.d/mongodb-enterprise.list
```

Step 2 – Install the Enterprise edition:

```
terminal> sudo apt-get install mongodb-enterprise
```

4.5 MongoDB Enterprise Encryption

As discussed in Section 4.3, it is essential to test encryption speed of both databases, in other words, the cost of encrypting data at rest. To prepare MongoDB for such test, firstly, the database encryption keys must be generated and saved to be accessible by the database itself. Secondly, data encryption is not active by default, and therefore, an argument must be passed to enable the encryption. We used OpenSSL to randomly generate the 256-bit key, and the key is then stored in a file named "system_keys" which is then passed to the mongod command.

The following must be performed in order to enable MongoDB encryption engine.

```
terminal> openssl rand -base64 32 > system_keys  
terminal> mongod --enableEncryption --encryptionKeyFile system_keys
```

4.6 Cassandra Enterprise

Cassandra enterprise edition is maintained and developed by Datastax, and therefore, it is no longer available under Apache open source license. The installation is done merely by downloading the related package from Datastax website provided at [117] and following the installation instructions.

4.7 Cassandra Enterprise Encryption

In general, enabling encryption in Cassandra is similar to MongoDB in that keys must be generated first, and then encryption will take place. However, Cassandra enterprise provides its tool to generate encryption keys, and the tool accepts key length and name of encryption algorithm used as parameters:

```
terminal> dsetool createsystemkey 'AES/ECB/PKCS5Padding' 256 system_key
```

By default, the generated key(s) will be stored under “dse/resources/dse/system_keys/system_key”. Another advantage of Cassandra enterprise encryption engine is that it allows per-table encryption, which is a benefit over MongoDB encryption style which only allows per-database encryption.

After creating ycsb keyspace along with usertable collection, we alter the table to add encryption accordingly:

```
cqlsh> ALTER TABLE usertable WITH compression = {'class':  
'EncryptingLZ4Compressor', 'cipher_algorithm': 'AES/ECB/PKCS5Padding',  
'secret_key_strength': 256, 'system_key_file': 'system_key'};
```

CHAPTER 5

RESULTS

In this chapter, the results of the two tested databases, MongoDB and Cassandra, are thoroughly explained in tables and illustrated figures. As we discussed in the previous chapter, YCSB benchmark tool will be used to emphasize testing the two databases according to the following four workloads (according to YCSB documentation site):

- 1 – **Workload A:** This workload has a mix of 50% reads and 50% writes. An application example is a session store recording recent actions.
- 2 – **Workload B:** This workload has 95% reads and 5% writes. We use application examples such as photo tagging and adding a tag as an update, but most operations are to read tags.
- 3 – **Workload E:** This workload has 95% scans and 5% inserts. In this workload, short ranges of records are queried instead of individual records. An application example is threaded conversations.
- 4 – **Workload F:** This workload has 50% reads and 50% read-modify-writes. In this workload, the client will read a record, modify it, and write back the changes.

Three testing scenarios are proposed in this study for both databases, (1) Single node server test, (2) Multi-node test (Cluster), and (3) encryption overhead. At the end of this chapter, a comprehensive evaluation of four databases security (including enterprise edition) features is given.

5.1 Single Node Test

Several previous studies have considered testing of MongoDB and Cassandra performances in a single node scenario; however, there have been numerous architectural database updates since the newer versions of MongoDB 3.6.3, and Cassandra 3.11.1. Therefore, we are opted to repeat this test accordingly and to record any performance improvements/differences.

For each of the workload mentioned above, we have tested each database with record counts from 20K to 100K and operation counts from 20K to 100K.

5.1.1 Workload A

Workload A is divided equally into 50% read, and 50% update, the tables below depict the performance of both databases along with throughput and average latencies.

Table 5.1. Test Results of MongoDB (Workload A)

	Run Time	Throughput	Read Operations	Average Read Latency	Update Operations	Average Update Latency
1	16435.333	1836.577	9931	715.426	10069	853.327
2	34237.333	1178.981	20039	770.210	19961	897.791
3	46760.666	1308.195	29931.666	694.376	30068.333	829.756
4	68618.666	1176.480	39913.666	782.349	40086.333	903.535
5	90840	1100.874	49947.666	826.171	50052.333	964.233

Table 5.2. Test Results of Cassandra (Workload A)

	Run Time	Throughput	Read Operations	Average Read Latency	Update Operations	Average Update Latency
1	39930	501.584	9946	1929.983	10054	1709.200
2	72121.333	554.783	20021.666	1782.545	19978.333	1639.367
3	114176.666	525.575	30002	1906.669	29998	1768.465
4	153660.666	521.470	39891.333	1956.324	40108.666	1783.894
5	180697.333	553.602	49926.666	1841.026	50073.333	1689.928

Notice also Figure 5.1 which gives the more precise idea of the performance results.



Figure 5.1. Workload A Test Performances Graph

5.1.1.1 Workload A Findings

Based on the illustrated results in Figure 5.1, we can conclude the following:

- 1 – Both databases performance was closely similar in Read and Update operations (Figure 5.1 – C, E).
- 2 – MongoDB has lower Read and Update latency (Figure 5.1 – D, F).
- 3 – MongoDB is faster in terms of Runtime and Throughput (Figure 5.1 – A, B).

5.1.2 Workload B

Workload B consists of heavy Read at 95% with only 5% of update operation, the tables below depict the performance of both databases along with throughput and average latencies.

Table 5.3. Test Results of MongoDB (Workload B)

	Run Time	Throughput	Read Operations	Average Read Latency	Update Operations	Average Update Latency
1	18073	1106.901	18999.666	847.681	1000.333	1255.566
2	35382	1130.569	37992.333	847.934	2007.666	1153.773
3	50223.333	1197.029	56997.333	806.477	3002.666	1087.200
4	69030.333	1158.954	75938.666	833.732	4061.333	1114.189
5	84827	1178.901	94944	823.008	5056	1079.744

Table 5.4. Test Results of Cassandra (Workload B)

	Run Time	Throughput	Read Operations	Average Read Latency	Update Operations	Average Update Latency
1	39458.333	507.13	18956.333	1786.055	1043.666	1948.521
2	73361	545.277	37985.333	1744.496	2014.666	1705.923
3	106671	562.504	57023.666	1714.173	2976.333	1713.62
4	142182.333	562.662	75979	1729.820	4021	1704.219
5	174442.333	573.276	94991	1705.425	5009	1678.946

We can easily conclude that MongoDB is still better than Cassandra because MongoDB has also performed better than Cassandra. Although read and update operations baseline remains close, MongoDB has demonstrated low latency in both. Figure 5.2 below illustrates the performances results for workload B.

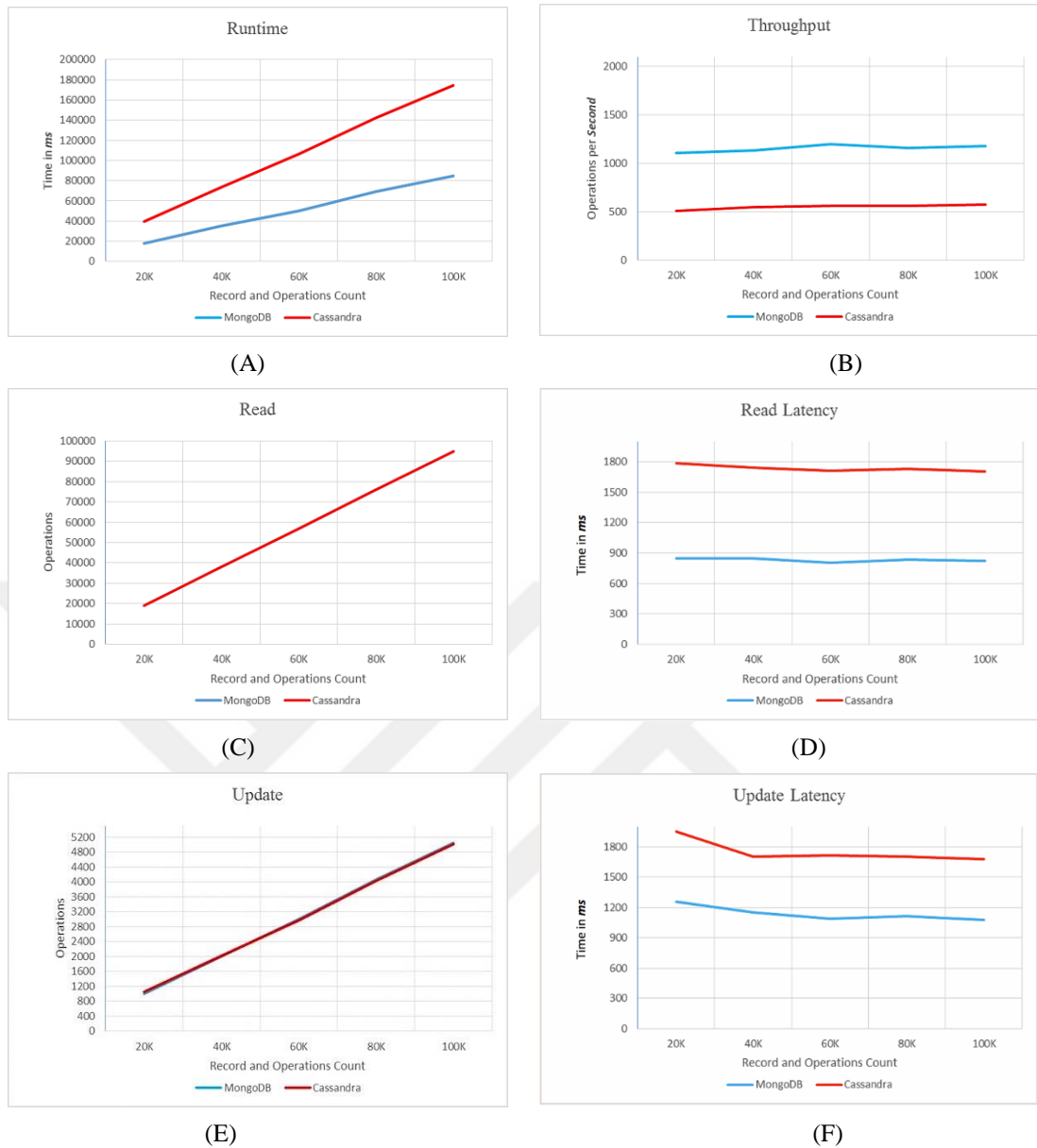


Figure 5.2. Workload B Test Performances Graph

5.1.2.1 Workload B Findings

Based on the illustrated results in Figure 5.2, we can conclude:

- 1 – Both databases performance is closely similar in Read and Update, but MongoDB has performed slightly better in Update operations (Figure 5.2 – C, E).
- 2 – MongoDB has much lower Read and Update latencies (Figure 5.2 – D, F).
- 3 – MongoDB has also performed faster in terms of Runtime and Throughput (Figure 5.2 – A, B).

5.1.3 Workload E

Workload E consists of massive Scans at 95%, and 5% inserts, the tables along with figures below illustrate the observed performances of both databases along with throughput and average latencies.

Table 5.5. Test Results of MongoDB (Workload E)

	Run Time	Throughput	Scan Operations	Average Scan Latency	Insert Operations	Average Insert Latency
1	61140	327.248	18982	3116.253	1018	1215.029
2	122496.333	326.551	37998.333	3132.648	2001.666	1248.305
3	182979.333	327.956	57039.333	3121.077	2960.666	1281.497
4	247197	323.638	75945.333	3166.320	4054.666	1342.127
5	312054.666	320.487	94990.666	3199.101	5009.333	1354.309

Table 5.6. Test Results of Cassandra (Workload E)

	Run Time	Throughput	Scan Operations	Average Scan Latency	Insert Operations	Average Insert Latency
1	80894	247.681	19019.333	3939.912	980.666	2408.27
2	169119	236.662	37998	4244.616	2002	2058.631
3	251757	238.74	57008	4237.062	2992	2090.074
4	341142.333	234.517	75967.666	4330.378	4032.333	2021.399
5	429940	232.742	94996.333	4373.914	5003.666	2057.852

See Figure 5.3 below which also concerns workload E.

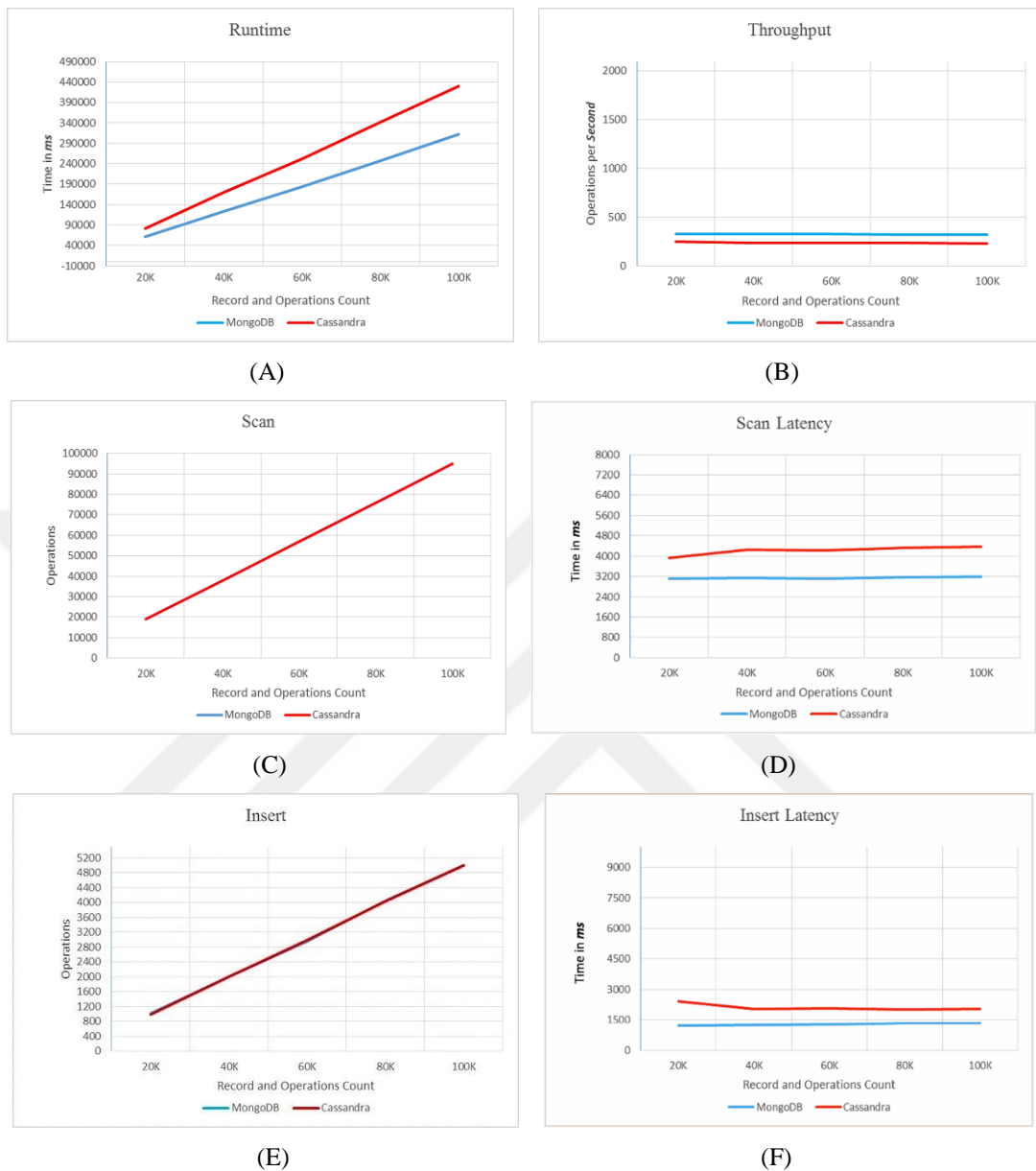


Figure 5.3. Workload E Test Performances Graph

5.1.3.1 Workload E Findings

In this workload, we are laboriously scanning for the queries with a partial percentage of inserting operations. Based on the illustrated results in Figure 5.3, we can conclude:

- 1 – Both databases performance is closely similar in Scans and Inserts (Figure 5.3 – C, E).

- 2 – MongoDB has lower Scan and Insert latencies, although Cassandra performance starts getting closer to MongoDB (Figure 5.3 – D, F).
- 3 – Although, MongoDB has the faster Runtime, MongoDB and Cassandra Throughput baseliners are not far away from each other (Figure 5.3 – A, B).

5.1.4 Workload F

Throughout our test, we have found the workload F to be the highest workload concerning performance and latency. It consists of 50% Reads and 50% Read-Modify-Writes operations. The tables below depict the measured performances of both databases along with throughput and average latencies, while Figure 5.4 illustrates performances comparison.

Table 5.7. Test Results of MongoDB (Workload F)

	Run Time	Throughput	Read Operations	Average Read Latency	RMW Operations	Average RMW Latency
1	27351	731.251	20000	830.059	9986.333	1840.798
2	50711.666	794.682	40000	786.985	19889.666	1707.952
3	79389.666	755.824	60000	824.531	29932	1791.495
4	101342	792.430	80000	785.904	40127	1709.865
5	134400.333	744.292	100000	836.038	49863.666	1827.968

Table 5.8. Test Results of Cassandra (Workload F)

	Run Time	Throughput	Read Operations	Average Read Latency	RMW Operations	Average RMW Latency
1	53806.666	371.71	20000.0	1689.699	10026	3333.299
2	104141	384.100	40000.0	1713.053	19997.333	3323.215
3	153152	391.768	60000.0	1689.571	29937.666	3302.937
4	205605.666	389.098	80000.0	1723.413	40042.333	3321.504
5	250886.333	398.586	100000.0	1677.215	50029	3256.287

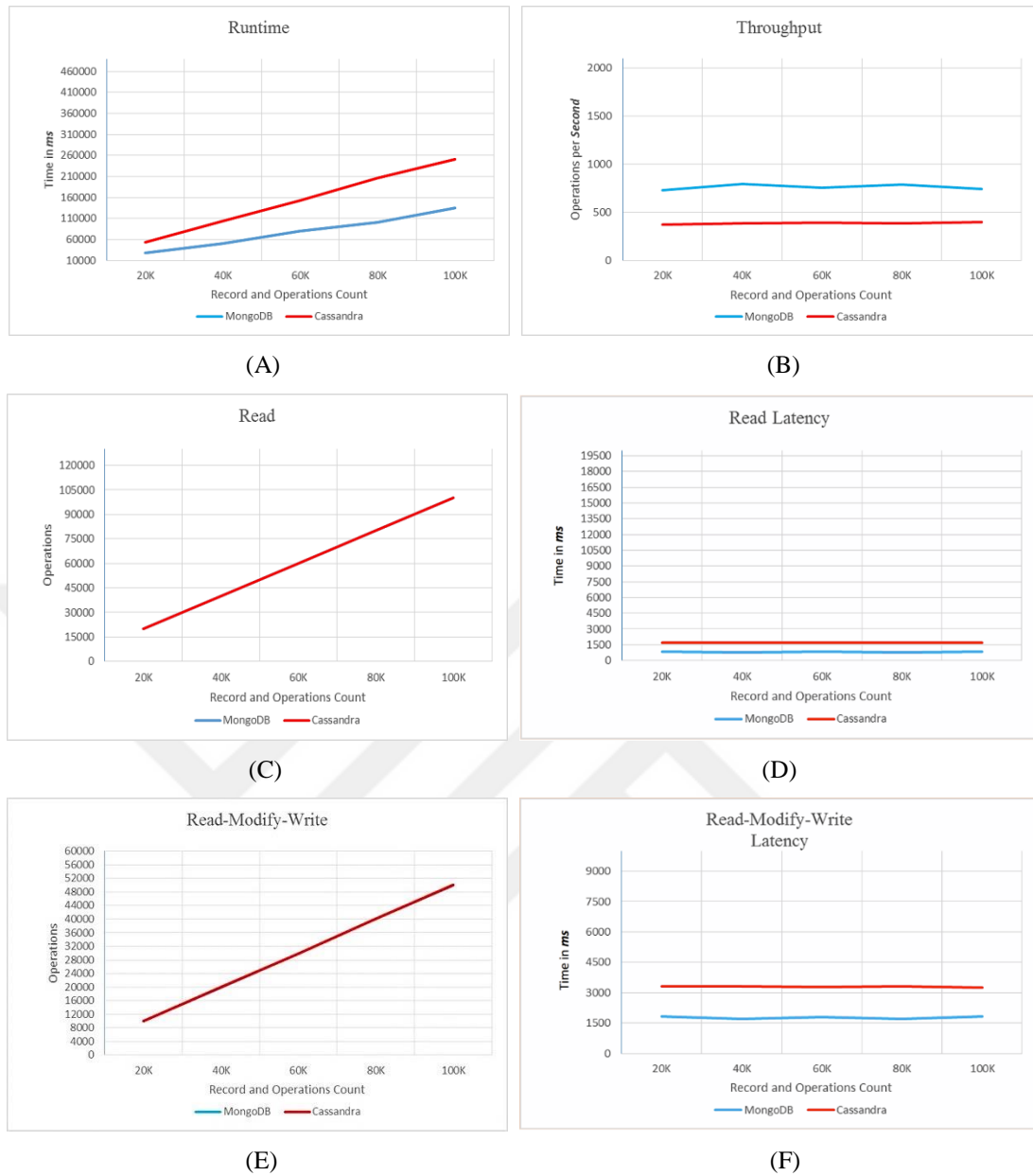


Figure 5.4. Workload F Test Performances Graph

5.1.4.1 Workload F Findings

In this workload, we have equally performed Reading and Read-Modify-Write operations. MongoDB still outperforms Cassandra in every aspect, runtime, throughput, and latencies, see Figure 5.4.

5.2 Multi Node Test (Cluster)

Real world production environment always relies on clustering to deploy databases. Such that is to gain better performance, to provide data safety and services high availability during failovers. Single node test of MongoDB and Cassandra gives a thorough understanding of where both systems stand in term of performance and latencies; however, such test cannot be generalized to include the performance of the same databases in a cluster configuration.

Therefore, the second part of the test work presented in this chapter will focus on testing both databases in a cluster configuration, each cluster with four nodes. The test is conducted with different record counts and operation counts, each of which starts from 200K to 1000K, respectively. The details of each configuration have been mentioned in the previous chapter, see Sections 4.2.2.1 and 4.2.3.1.

5.2.1 Workload A

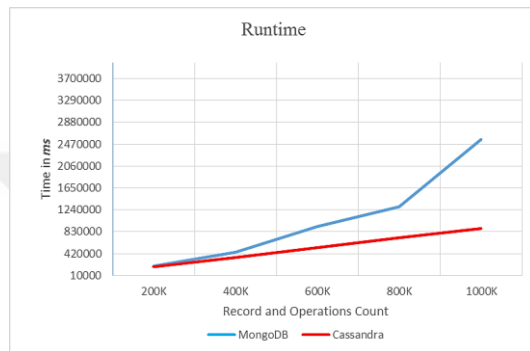
The same procedure is followed as in single node test. Similarly, the tables below and figures illustrate the observed performance.

Table 5.9. Test Results of MongoDB (Workload A)

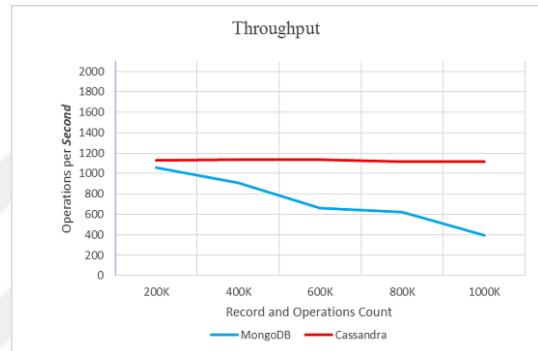
	Run Time	Throughput	Read Operations	Average Read Latency	Update Operations	Average Update Latency
1	189325.666	1056.521	100008	802.976	99992	1074.406
2	442939.666	911.039	199921	988.431	200079	1213.205
3	922747	658.34	300038.333	1380.849	299961.666	1682.892
4	1303404.666	619.233	399872.333	1460.166	400127.666	1787.272
5	2555220	392.490	500067.333	2450.367	499932.666	2649.419

Table 5.10. Test Results of Cassandra (Workload A)

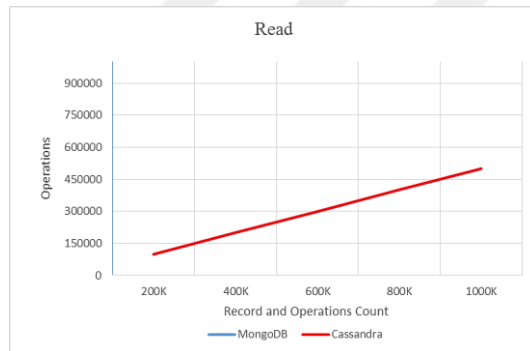
	Run Time	Throughput	Read Operations	Average Read Latency	Update Operations	Average Update Latency
1	177161.333	1129.016	100001	1012.823	99999	709.259
2	351265.333	1138.748	200067	1019.912	199933	702.684
3	528788	1134.67	299697.333	1033.142	300302.666	2104.688
4	716027.333	1117.376	400081.3333	1056.752	399918.666	707.610
5	896105.666	1116.104	499666.666	1063.785	500333.333	704.772



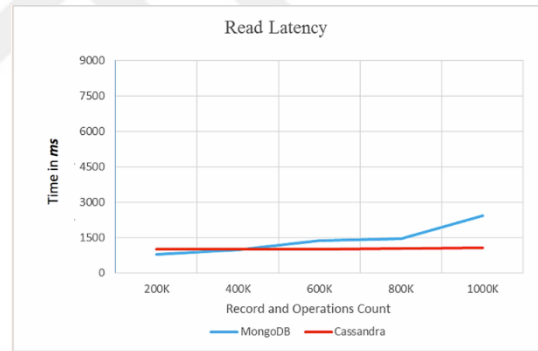
(A)



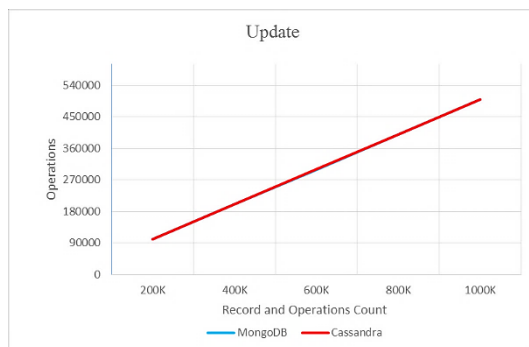
(B)



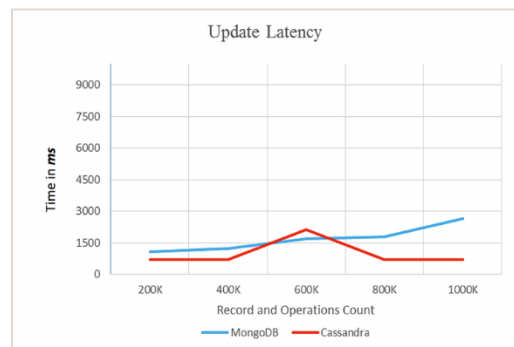
(C)



(D)



(E)



(F)

Figure 5.5. Workload A Test Performances Graph

5.2.1.1 Workload A Findings

Cassandra cluster has impressively outperformed MongoDB in both Runtime and throughput (see Figure 5.5 – A, B). Although MongoDB outperformed Cassandra in Read latency at 200K, and for Update latency at 600K, MongoDB much worse in all other record sizes for both Read and Update latencies (see Figure 5.5 – D, F).

5.2.2 Workload B

With 95% Read and 5% update, the tables and figure given below show the obtained results.

Table 5.11. Test Results of MongoDB (Workload B)

	Run Time	Throughput	Read Operations	Average Read Latency	Update Operations	Average Update Latency
1	160311.333	1247.590	190027.666	784.821	9972.333	985.462
2	325884.333	1227.465	380013.333	789.11	19986.666	1198.377
3	502878.333	1193.212	570099.333	806.697	29900.666	1417.208
4	858295	932.615	760093.333	1052.147	39906.666	1380.099
5	1784495	560.897	950099.333	1758.093	49900.666	2201.564

Table 5.12. Test Results of Cassandra (Workload B)

	Run Time	Throughput	Read Operations	Average Read Latency	Update Operations	Average Update Latency
1	207330	964.645	189958.666	1023.69	10041.333	820.578
2	415484.666	962.738	380012	1034.872	19988	808.048
3	656912.666	913.377	570160	1096.912	29840	811.614
4	903624	885.419	759964.333	1134.951	40035.666	809.569
5	1097332	911.306	950123	1102.226	49877	806.6163

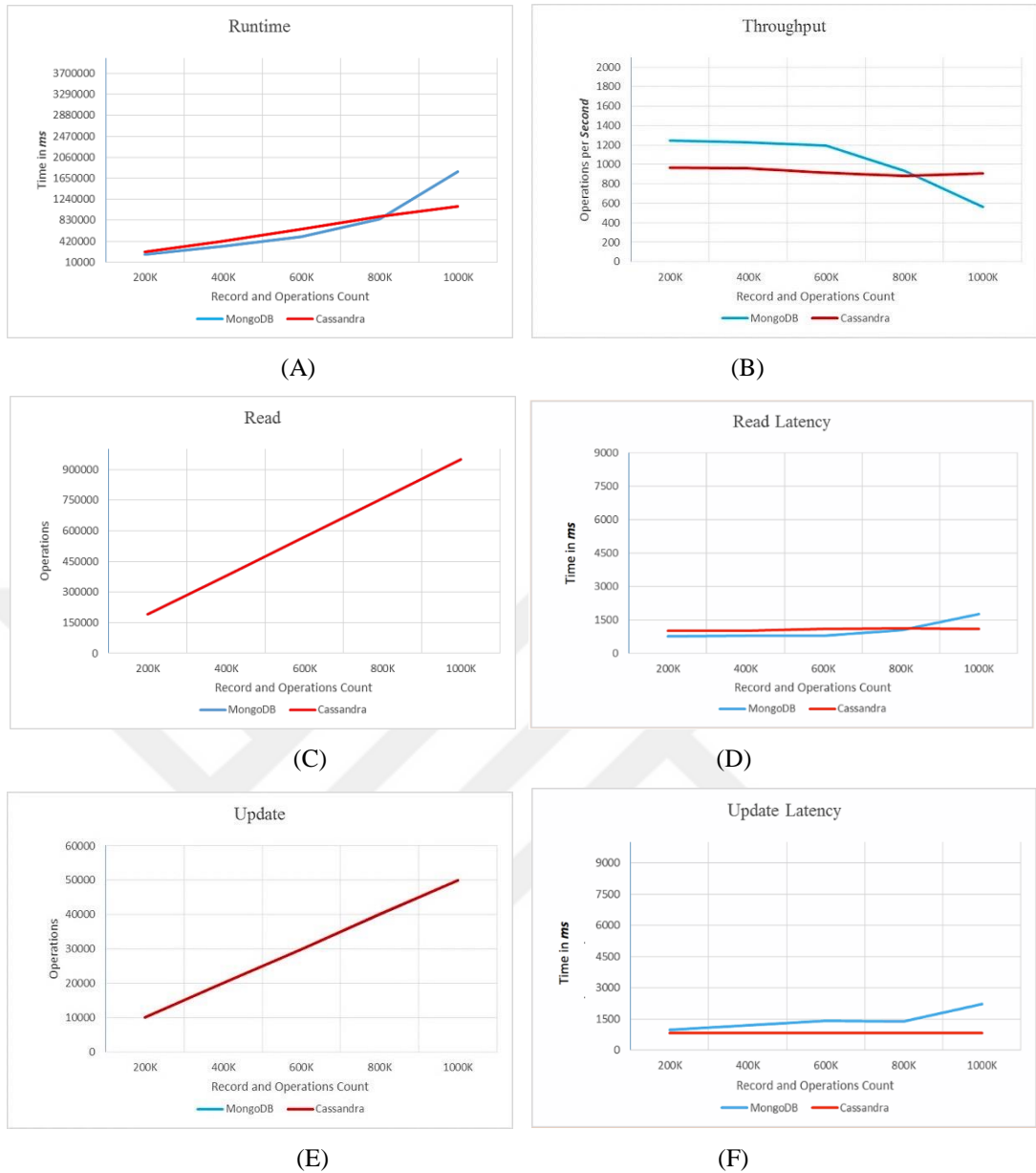


Figure 5.6. Workload B Test Performances Graph

5.2.2.1 Workload B Findings

Workload B shows that MongoDB has outperformed Cassandra in both time and throughput (see Figure 5.6 – A, B). However, it fails at big record sizes, precisely at 800K and 1000K. For Read latency (see Figure 5.6 – D), MongoDB performance was better at small record sizes, 200K, 400K, and 600K, but it fails at bigger records of 800K and 1000K. On the other hand, Cassandra outperformed MongoDB in all workloads for Update latency (see Figure 5.6 – F).

5.2.3 Workload E

Workload E has Scan and Insert operations. Performance results are entirely different from single node test, as shown in the tables and figures below.

Table 5.13. Test Results of MongoDB (Workload E)

	Run Time	Throughput	Scan Operations	Average Scan Latency	Insert Operations	Average Insert Latency
1	489738.666	408.397	189961	2508.358	10039	1170.669
2	983812.333	406.583	380000	2522.627	20000	1144.206
3	1559814.666	384.734	569946.333	2671.215	30053.666	1137.814
4	2305628.666	347.120	760287.666	2967.738	39712.333	1143.912
5	2992651.333	334.343	949899.666	3084.228	50100.333	1157.343

Table 5.14. Test Results of Cassandra (Workload E)

	Run Time	Throughput	Scan Operations	Average Scan Latency	Insert Operations	Average Insert Latency
1	669796	298.608	189996	3444.24	10004	1089.534
2	1407478.333	284.201	379963.333	3631.700	20036.666	1090.711
3	2143035	279.988	570193.333	3690.331	29806.666	1070.743
4	2905027	275.384	760025.333	3755.047	39974.666	1075.058
5	3728085.333	268.382	950125.666	3855.991	49874.333	1098.939

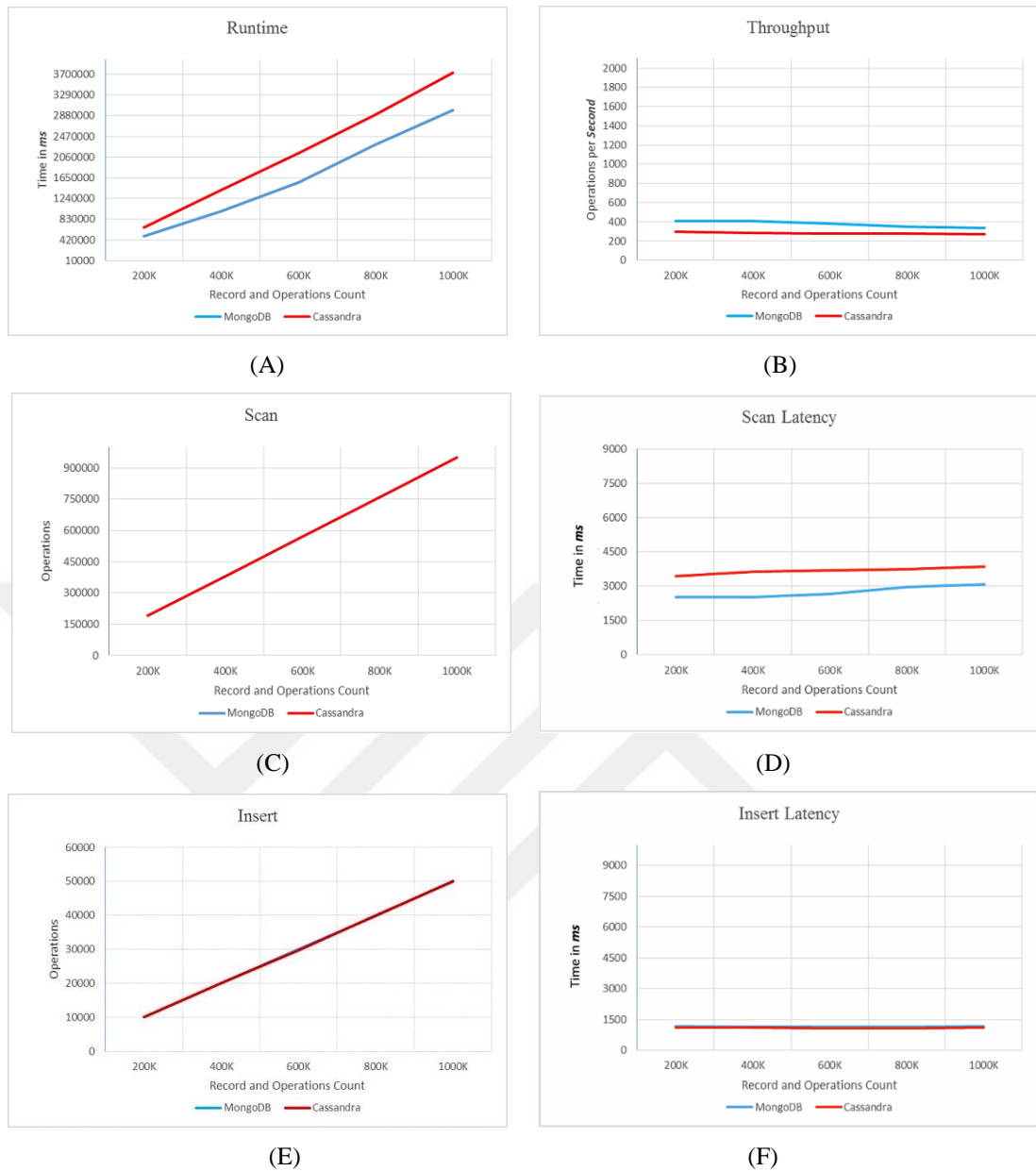


Figure 5.7. Workload E Test Performances Graph

5.2.3.1 Workload E Findings

MongoDB cluster has observed less running time and more throughput compared to Cassandra (see Figure 5.7 – A, B). Insert and Scan operations' baselines are closely similar. However, MongoDB cluster has outperformed Cassandra with Scan latency while the baseline remained close in Insert latency (see Figure 5.7 – D, F).

5.2.4 Workload F

Similarly, workload F is the slowest among other tested workloads. It has taken 45 minutes when loading 1000K data in cluster test. The figure and tables are shown below summarize the obtained results.

Table 5.15. Test Results of MongoDB (Workload F)

	Run Time	Throughput	Read Operations	Average Read Latency	RMW Operations	Average RMW Latency
1	263087.333	760.204	200000.0	788.969	100032.333	1827.800
2	548234.333	729.658	400000.0	799.145	199982	1932.037
3	878288.666	683.342	600000.0	846.958	300256.333	2075.620
4	1517119.333	527.782	800000.0	1347.153	399828	2426.343
5	2589082.666	389.412	1000000.0	2042.227	500663.666	3126.639

Table 5.16. Test Results of Cassandra (Workload F)

	Run Time	Throughput	Read Operations	Average Read Latency	RMW Operations	Average RMW Latency
1	281281	711.032	200000	1006.159	99915.666	1760.635
2	564644.666	708.411	400000	1020.211	199815.333	1773.740
3	885190.333	677.827	600000	1084.707	299888.333	1840.972
4	1214971.333	658.554	800000	1125.522	400062	1888.033
5	1492033.333	670.283	1000000	1102.238	500035	1861.163

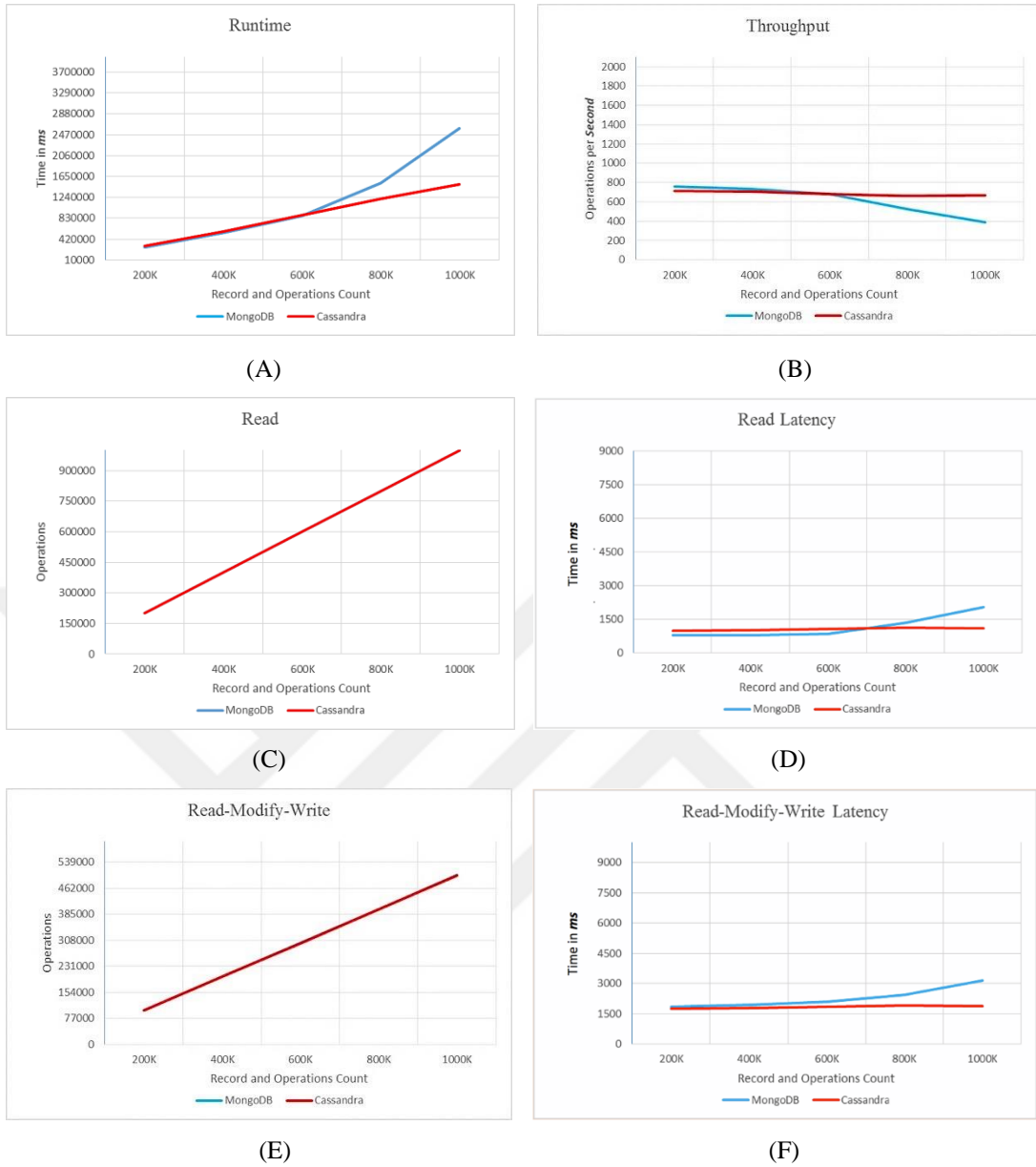


Figure 5.8. Workload F Test Performances Graph

5.2.4.1 Workload F Findings

MongoDB cluster has performed slightly similar to Cassandra during 200K, 400K, and 600K. However, time and throughput performances behavior increasingly become worse during 800K, and 1000K, respectively. Similarly, the performance baseline grew up in both Read (at 800K) and Read-Modify-Update (at 600K) latencies.

5.3 NoSQL Security

After reviewing the selected set of features defined in Section 4.3, we have found that open source version of both databases supports only a few features when compared to the enterprise edition. We have also found that Cassandra supports more algorithms and security features when compared to MongoDB enterprise edition. The table below summarizes the comparison of selected security features among MongoDB and Cassandra (Open Source and Enterprise).

5.3.1 Security Features Findings

From the above table, we can conclude the points below:

- 1 – MongoDB (open source and enterprise edition) supports more authentication algorithms while Cassandra supports taking the lead in the authorization.
- 2 - Cassandra supports a wide selection of encryption algorithms to encrypt data over the network and at rest (i.e., stored on disk), with an ability to choose various key lengths, such as 40, 56, 112, 128, 168, 192, 256, and 448 bit.
- 3 – Cassandra (open source and enterprise edition) supports JMX authentication and authentication caching. However, Proxy roles are supported by enterprise edition only, while MongoDB has no native support for these features.
- 4 – MongoDB provides better auditing features which are essential in database administration.
- 5 – Node to node and client to node encryption are supported by Cassandra (open source and enterprise edition) and MongoDB enterprise. However, MongoDB open source version only supports the client to node encryption.
- 6 – Last and from [16, 18, 20], we can add the following:

Table 5.17. Security Features Comparison

Features	MongoDB (open Source)	Cassandra (open Source)	MongoDB (Enterprise)	Cassandra (Enterprise)
Authentication	Yes supported: SCRAM, MongoDB-CR, x.509	Yes supported: Internal Authentication	Yes supported: SCRAM, MONGODB-CR, x.509, FIPS, Kerberos, SASL binding to LDAP servers.	Yes supported: Internal, LDAP, Kerberos
Authorization	Yes supported: RBAC	Yes supported: Cassandra Role Manager	Yes supported: RBAC	Yes supported: RBAC, LDAP, Kerberos
Transport Encryption	Yes supported: TLS/SSL	Yes Supported: TLS, SSL	Yes supported: FIPS, TLS/SSL	Yes supported: AES/CBC/PKCS5Padding, AES/ECB/PKCS5Padding, DES/CBC/PKCS5Padding, DESede/CBC/PKCS5Padding, Blowfish/CBC/PKCS5Padding, RC2/CBC/PKCS5Padding
Encryption At Rest	No	No	Yes supported: AES256-CBC via OpenSSL. AES256-GCM	Yes supported: AES/CBC/PKCS5Padding, AES/ECB/PKCS5Padding, DES/CBC/PKCS5Padding, DESede/CBC/PKCS5Padding, Blowfish/CBC/PKCS5Padding, RC2/CBC/PKCS5Padding
Auditing	No	No	Yes supported: schema (DDL), replica set and sharded cluster, authentication and authorization, CRUD operations, system activities from users to databases applications	Yes supported: AUTH, DML, DDL, DCL, Query
JMX Authentication	No	Yes Supported: File based username and password authentication	No	Yes Supported: File based username and password authentication, Cassandra-controlled roles and passwords.
Authentication Caching	No	Yes	No	Yes Supported: Internal (using given username and password), LDAP
Proxy Rules	No	No	No	Yes Supported: Internal (using given username and password), LDAP
Node-to-Node encryption	No	Yes Supported: TLS, SSL, FIPS	Yes Supported: TLS, SSL, FIPS	Yes Supported: TLS_RSA with AES 128 CBC_SHA, TLS_DHE_RSA with AES 256 CBC_SHA
Client-to-Node encryption	Yes Supported: TLS, SSL, FIPS	Yes Supported: TLS, SSL, FIPS	Yes Supported: TLS, SSL, FIPS	Yes Supported: TLS_RSA with AES 128 CBC_SHA, TLS_DHE_RSA with AES 256 CBC_SHA

5.3.2 Cassandra:

- Default does not encrypt Cassandra Data Files meaning that an attacker with access to the file-system can directly extract information database files.
- Default does not encrypt Inter-Cluster and Client communication interfaces. Encryption and clients-certificates should be enabled by setting the `.keystore` and `.truststore` with their related paths.
- Cassandra Query Language is a parsed language. Thus it is vulnerable to injection-attack.
- Cassandra uses Thread-Per-Client model which means a new connection requires Cassandra server to set new thread. Therefore, Cassandra is more vulnerable to DOS attacks where an attacker can initiate numerous fake connections to Cassandra server by only knowing remote server IP address. Cassandra project recommends utilizing connection pool to protect system resources better.
- Authentication in Cassandra is provided through *IAuthenticate* interface and through *SimpleAuthenticator* class, both are not enabled by default. It is also important to note that client interface defined in Apache Thrift Framework [118] will always transmit passwords in plain-text, such that if an attacker that is capable of sniffing network packets between any client and databases can easily discover the password. All passwords are stored in unsalted MD5 hashed format, however, MD5 is not cryptographically secure and it is easy to recover exact plain-text using pre-calculated lists and rainbow tables found online.
- Authorization in Cassandra is available through *IAuthority* interface, which comes with a set of security concerns. The first concern is the pass-through implementation, which gives full access permissions to all database files regardless of the users. The second one, weakness is the set of permissions given to any user which is stored on a flat file and not on a synchronized file across the cluster nodes. Therefore, the individual user must re-give the same set of permissions at each node of the cluster.

5.3.3 MongoDB:

- Default does not encrypt MongoDB Data Files meaning that an attacker with access to the file-system can directly extract information database files.
- MongoDB uses the TCP port 27017 by default for clients' connections and TCP port 28017 for management statistics; default does not encrypt both of these ports.
- MongoDB heavily utilize JavaScript in its MongoDB Query Language which makes it more prone to script-injection and XSS attacks.
- MongoDB does not support SSL client-node communication by default.
- MongoDB authentication is not supported when running in Shard mode, only in standalone and replica-set mode. However, it is not enabled by default. Passwords and user authentications details are stored in a flat file in the following format: `<user name>: mongo: < password>`. Such file can be easily accessed by an attacker from the admin data-files and all the defined passwords are recovered. Although passwords are MD5 hashed, MD5 is still cryptographically insecure.
- Authorization is also not supported in Shard mode, and only in replica-set and standalone mode.

5.4 Encryption at Rest Benchmark

Encryption of data at a network or disk level comes at the prices of speed, or what is known as encryption overhead. Overhead is defined as the time spent by encryption engine to encrypt all the incoming data, and generally, it can be measured according to following formula:

$$\text{Overhead Time} = (\text{Run time (Encryption)} - \text{Run time (normal)}) / \text{Run time (normal)}$$

From Table 5.17, we can see that encryption at rest is only available in enterprise edition of both databases. Therefore, to calculate the overhead time, we have used YCSB tool with the same workloads used in this chapter and measured time and throughput for each database without encryption, then after, we have enabled encryption according to Sections 4.5 and 4.7 and re-performed the same test. We

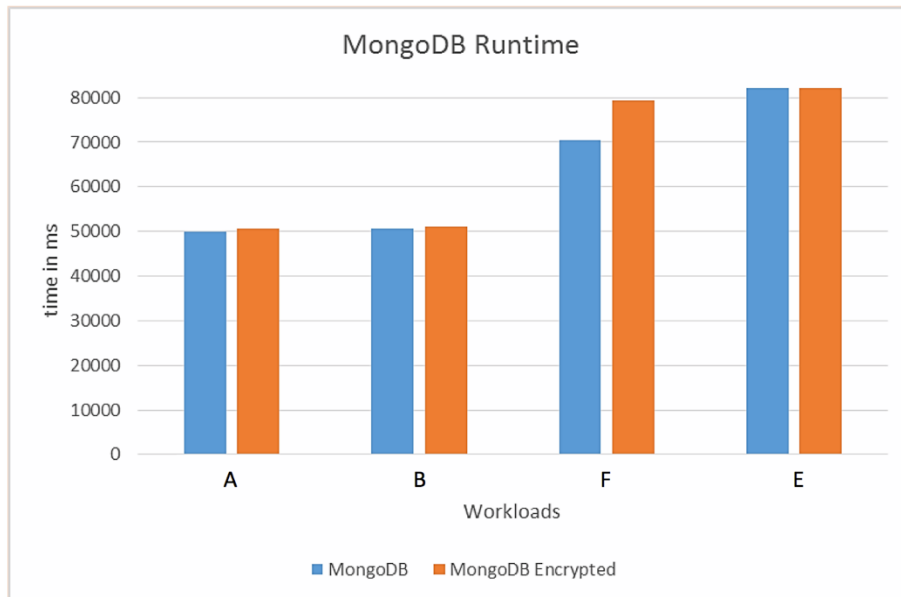
have used AES256-CBC encryption algorithm with 256-bit of key length in MongoDB and Cassandra, however, in Cassandra, the algorithm is found under a different name, which is AES/CBC/PKCS5Padding. The results are shown in Tables 5.18, 5.19, and Figure 5.9.

Table 5.18. MongoDB Overhead Results

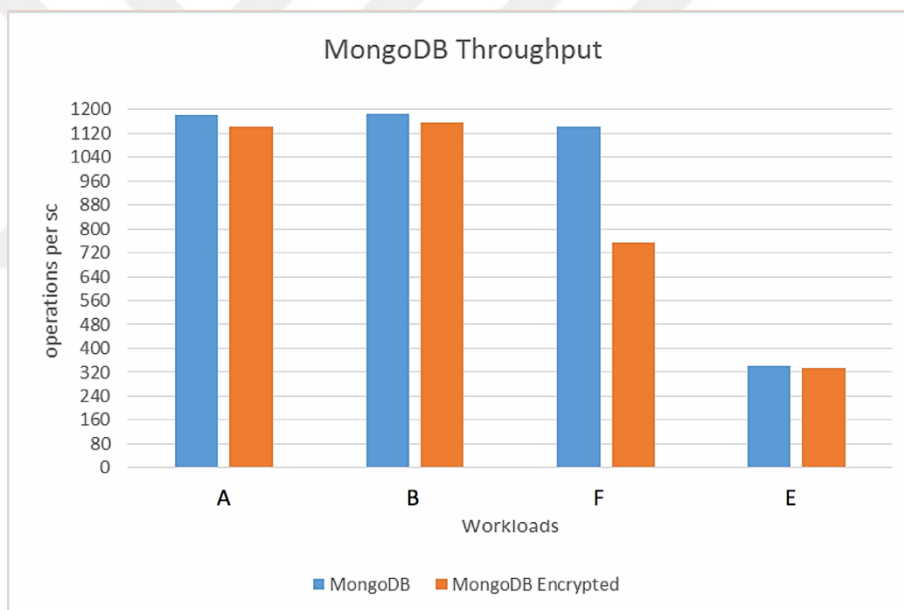
	Workloads	MongoDB Enterprise (non-Encrypted)		MongoDB Enterprise (Encrypted)		Overhead	
		Avg Run Time	Avg Throughput	Avg Run Time	Avg Throughput	Time	Throughput
1	A	50027.666	1180.041	50643.013	1141.687	1.23 %	- 3.25 %
2	B	50555.266	1185.747	51196.533	1157.266	1.26 %	- 2.40 %
3	F	70562.466	1140.109	79285.199	753.203	12.36 %	- 33.93 %
4	E	181008.199	340.082	183667.200	334.518	1.46 %	- 1.63 %

Table 5.19. Cassandra Overhead Results

	Workloads	Cassandra Enterprise (non-Encrypted)		Cassandra Enterprise (Encrypted)		Overhead	
		Avg Run Time	Avg Throughput	Avg Run Time	Avg Throughput	Time	Throughput
1	A	111273.399	530.385	112484.333	518.962	1.08 %	- 2.15 %
2	B	111650.399	532.081	112734.133	527.080	0.97 %	- 0.93 %
3	F	166336.732	359.051	169537.999	353.918	1.92 %	- 1.42 %
4	E	275627.999	223.216	288188.733	217.885	4.55 %	- 2.38 %



(A) MongoDB Runtime

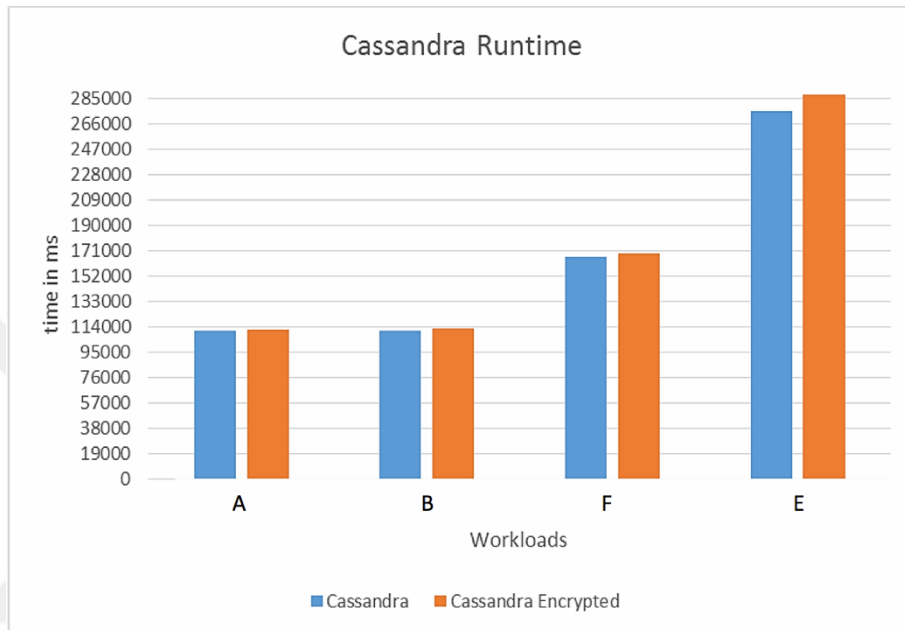


(B) MongoDB Throughput

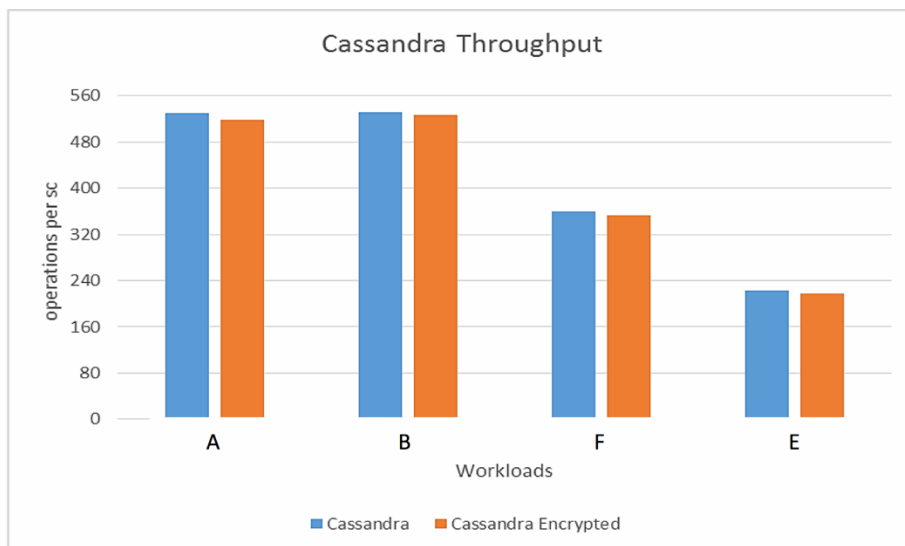
Figure 5.9. MongoDB Overhead Performances Graph

From the Figure 5.9.A above, we can see that MongoDB performed well regarding runtime overhead. The overall runtime overhead for workload A, B, and E is below 2%. However, overhead time of workload F was the worse at 12%.

Throughput also decreased by 3.25%, 2.40%, and 1.63% for workloads A, B, and E respectively (see Figure 5.9.B). We also found that workload F gave the worse throughput overhead at 33.93% decrease. Therefore, we can conclude that Read-Modify-Write is the costliest operations to perform when considering encryption at rest in MongoDB.



(A) Cassandra Runtime



(B) Cassandra Throughput

Figure 5.10. Cassandra Overhead Performances Graph

As it can be seen from Figure 5.10.A and 5.10.B Cassandra enterprise performance was much worse in comparison with MongoDB. From Tables 5.18 and 5.19, Cassandra is between ~ 2x to 2.5x slower than MongoDB in all workloads. Concerning encryption runtime and throughput overhead, we observed less than 2% overhead of the total runtime and 3% throughput decrease for all workloads with ~ 5% runtime overhead for workload E.

5.4.1 Encryption Overhead Test Findings

According to the results presented, we have observed the following:

- 1 – Time and throughput overhead: although both databases had closely the same overhead percentage for runtime and throughput, MongoDB performance is 2x to 2.5x faster than Cassandra. We found Read-Modify-Write to be the costliest operation in MongoDB while Scan and Insert operations are the costliest for Cassandra.
- 2 – MongoDB performed considerably slow workload F while Cassandra in workload E - with and without encryption - since we expected steady performance overhead, i.e., different operations should have the same cost.

CHAPTER 6

CONCLUSION

This study presents the performance and security comparison results of two NoSQL databases (MongoDB and Apache Cassandra). To answer the questions of the study presented in Section 1.3, we have prepared a test environment detailed in Chapter 4.

To answer the first question, MongoDB outperformed Cassandra in all workloads (A, B, E, and F) regarding runtime and throughput. It is safe to say that Cassandra performance was close to MongoDB only in Scan and Insert operations with comparable performance latencies, whereas we observed substantial performance differences in Read, Update, Read-Modify-Write operations with noticeable worse latencies.

Concerning, for the second question, in our multi-node test (i.e., cluster configuration), Cassandra outperformed MongoDB concerning runtime and throughput in workload A (MongoDB dropped Read latency from 400K to 1000K and much worse performance in Update latency), and workload B only at 800K and 1000K. While MongoDB took the lead in workload E (runtime, throughput, Scan latency, and closely similar Insert latency), both Cassandra and MongoDB performance was similar at 200K, 400K, and 600K in workload F with MongoDB falling behind in terms of runtime, throughput (at 800K, and 1000K), Read latency (from 800K), and Read-Modify-Write latency from records size 600K.

As regards the third question mentioned in section 1.3 above, besides the performance behavior in a cluster configuration, we should mention that MongoDB architecture supports horizontal and vertical scalability meaning that more nodes can

be configured to work as a single DB router or as multiple DB routers and the same thing is correct for config and shards server. In connection with this question, it is concluded that MongoDB is better suited for production environments.

Regarding the fourth research question, MongoDB enterprise edition came short concerning security features. Cassandra enterprise edition supports a wide range of authentication, authorization, and encryption algorithms, which makes it more capable of integrating into a heterogeneous environment.

As concerns, the fifth research question, Tables 5.18 and 5.19 reveal that we found Cassandra enterprise performance was worse by $\sim 2x$ to $2.5x$ for all workloads in comparison with MongoDB. Therefore, it is concluded that MongoDB is better suited to use when data encryption is a requirement.

Regarding the literature reviewed in Section 1.4, our results show that over the past few years MongoDB performance became better when compared with the previously published works. According to the single node test we found that MongoDB performed better concerning runtime and throughput when compared to Cassandra, in contrast with the studies published by John Klein in [12], Abramova in [13], and Jay [14].

We expected Cassandra performance to be better for all workloads in our proposed multi-node test due to its in-memory data structure. However, MongoDB recent edition 3.6.3 new architecture seems to handle performance and scalability much better than the previous versions when compared to Kumar and Roseline work published in [15]. Therefore, it shows that MongoDB cluster performance has also improved. To summarize, Cassandra has better Read and Update performance in cluster test due to its in-memory structure, while MongoDB is better at Insert and Scan operations.

Both databases showed improved security support and implementation of several features when compared to previously published studies in [16-19]. However, the comparison carried out in this study revealed that Cassandra enterprise edition took

the lead in the security features comparison because DataStax has acquired it and it is no longer available as an open source.

Lastly, we did not find any previously published work which studied encryption overhead of NoSQL databases making this work significant.

6.1 Suggestions for Future Study

The following topics are deemed worthy of future studies:

- 1 – Testing performance of vertically scaled MongoDB cluster with other NoSQL database.
- 2 – Checking the overhead method in the multi-node cluster and observing the performance of the new versions of MongoDB and Cassandra and other NoSQL database systems.
- 3 - Testing the performance of NoSQL database management systems with the relational database management system (RDBMS) to present a comparative study in term of performance and security.
- 4 – Testing the new security features that will be added in MongoDB, Cassandra, and checking whether is an enhancement in the previous features.

REFERENCES

- [1] Halper F., Kaufman M., Hurwitz J., "Big data for dummies," John Wiley & Sons Inc., Hoboken, New Jersey, 2013.
- [2] Agrawal D., Bernstein P., Bertino E., Davidson S., Dayal U., Franklin M., Widom J. (2012), *Challenges and Opportunities with Big Data* [Online]. Available: CRA.org.
- [3] Andlinger P. (2013), *RDBMS dominate the database market, but NoSQL systems are catching up* [Online]. Available: <https://engines.com>.
- [4] *NoSQL databases* [Online]. Available: <https://nosql-database.org>.
- [5] Li et al., "A performance comparison of SQL and NoSQL databases," *Proc. 2013 IEEE Pacific Rim Conf.*
- [6] Parker et al., "Comparing nosql mongodb to an sql db.", *Proc. 2013 ACM Southeast Conf.*
- [7] Hadjigeorgiou C., "Rdbms vs nosql: Performance and scaling comparison.", M.S. thesis, High Performance Computing, The University of Edinburgh, 2013.
- [8] Tudorica B. et al., "A comparison between several NoSQL databases with comments and notes," *Roedunet International Conference, IEEE, 2011.*
- [9] Hammes D., "Comparison of NoSQL and SQL Databases in the Cloud," *Proc. 2014 Southern Association for Information Systems Conf.*
- [10] Jatana N., Puri S., Ahuja M., Kathuria I., Gosain D., "A survey and comparison of relational and non-relational database," *International Journal of Engineering Research & Technology*, 2012.

- [11] Abubakar Y., Adeyi T., Auta I., "Performance evaluation of NoSQL systems using YCSB in a resource austere environment," *Performance Evaluation*, pp. 23-27, 2014.
- [12] Klein J. et al., "Performance evaluation of nosql databases: A case study," *Proc. 2015 1st Workshop on Performance Analysis of Big Data Systems, ACM*.
- [13] Veronika A. and Bernardino J., "NoSQL databases: MongoDB vs Cassandra," *Proc. 2013 international computer science and software engineering Conf., ACM*.
- [14] Choi C., "A study and comparison of NoSQL databases". M.S. thesis California State University, Northridge, USA, 2014.
- [15] Rajith K., Roseline R. Mary., "Comparative Performance Analysis of various NoSQL Databases: MongoDB, Cassandra and HBase on Yahoo Cloud Server," *Imperial Journal of Interdisciplinary Research*, Vol. 3, No. 4 2017.
- [16] Lior O. et al., "Security issues in nosql databases," in *Trust, Security and Privacy in Computing and Communications Conf.*, 2011.
- [17] Anam Z et al., "Security of sharded NoSQL databases: A comparative analysis," in *Information Assurance and Cyber Security Conf.*, 2014.
- [18] Preecha N. and Chomsiri T., "A comparison the level of security on top 5 open source NoSQL databases," in *9th International Information Technology and Applications Conf.*, 2014.
- [19] Sethuraman S. and Nair A., "Security maturity in NoSQL databases-are they secure enough to haul the modern it applications?" in *Advances in Computing, Communications and Informatics Conf.*, 2015.
- [20] Cuzzocrea, Alfredo, and Hossain Shahriar. "Data masking techniques for NoSQL database security: A systematic review." in *IEEE International Big Data Conf.*, 2017.
- [21] Cattell R., "Scalable SQL and NoSQL data stores," *Acm Sigmod Record*, Vol. 39, No. 4, pp.12-27, 2011.

- [22] Mike B., "The Chubby lock service for loosely-coupled distributed systems," *Proc. 2006 7th symposium on Operating systems design and implementation*.
- [23] Hector G. and Salem K., "Main memory database systems: An overview," *IEEE Transactions on knowledge and data engineering*, Vol. 4, No.6, pp. 509-516, 1992.
- [34] Antonios M. et al. "A classification of NoSQL data stores based on key design characteristics," in *Procedia Computer Science*, pp. 94-103, 2016.
- [25] Tiwari S., "CHAPTER 1: NOSQL: WHAT IT IS AND WHY YOU NEED IT," in *Professional NoSQL*, John Wiley & Sons Inc., pp. 9, 2011.
- [26] Ahmed O. et al., "Comparison and classification of nosql databases for big data," *Proc. 2015 International Big Data Conf.*
- [27] Mayur M. P. et al. "A qualitative analysis of the performance of MongoDB vs MySQL database based on insertion and retrieval operations using a web/android application to explore load balancing—Sharding in MongoDB and its advantages." *Proc. 2017 International IoT in Social, Mobile, Analytics and Cloud Conf.*
- [28] Sadalage P., Fowler M., "NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence," Adison-Wesley, pp. 39, 2013.
- [29] Sadalage P., Fowler M., "NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence," Adison-Wesley, pp. 40, 2013.
- [30] Rasha O. and Piazzolla P., "Modelling replication in nosql datastores," in *International Quantitative Evaluation of Systems Conf.*, 2014.
- [31] Chao-Hsien L. and Zheng Y., "Automatic SQL-to-NoSQL schema transformation over the MySQL and HBase databases," in *IEEE International Consumer Electronics, Taiwan*, 2015.
- [32] Sadalage P., Fowler M., "NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence," Adison-Wesley, pp. 49, 2013.
- [33] Sadalage P., Fowler M., "NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence," Adison-Wesley, pp. 50, 2013.

- [34] Sadalage P., Fowler M., "NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence," Addison-Wesley, pp. 51, 2013.
- [35] Dean J. and Ghemawat S., "MapReduce: Simplified Data Processing on Large Clusters," *Proc. 6th Symposium on Operating Systems Design & Implementation Conf.*, Vol. 6, 2004.
- [36] Wang G. and Tang J., "The NoSQL Principles and Basic Application of Cassandra Model," *Proc. International Computer Science and Service System Conf.*, Washington, DC, USA, pp. 1332-1335. 2012.
- [37] Brewer E. A., "Towards Robust Distributed Systems (Abstract)," *Proc. 19th Annual ACM Symposium on Principles of Distributed Computing*, Portland, Oregon, USA. pp. 7, 2000.
- [38] Gilbert S., and Lynch N. A., "Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services," *ACM SIGACT News*, Vol. 33, No. 2, pp. 51-59, 2002.
- [39] Abadi D., "Consistency Tradeoffs in Modern Distributed Database System Design: CAP is Only Part of the Story," *Computer*, Vol. 2, pp. 37-42, 2012.
- [40] Gilbert S., Lynch N., "Brewer's conjecture and the feasibility of consistent, available, partition tolerant web services," *ACM SIGACT*, Vol. 33, No. 2, pp. 51-59, 2012.
- [41] Gilbert S., Lynch N., "Perspectives on the CAP Theorem," *Computer*, Vol 45, No. 2, pp. 30-6, 2012.
- [42] Gessert F., Wingerath W., Friedrich S., Ritter N., "NoSQL Database Systems: A Survey and Decision Guidance," *Computer Science-Research and Development*, Vol 32, No. 3-4, pp. 353-356, 2017.
- [43] *Call me maybe: MongoDB* (18/5/2013) [Online]. Available:<https://aphyr.com>
- [44] Ganesh D., "BASE analysis of NoSQL databases," *Future Generation Computer Systems*", Vol. 52, pp. 13-21, 2015.
- [45] Vogels W., "Eventually Consistent," *Communications of the ACM*, Vol. 52, No. 1, pp. 40-44, 2009.

- [46] Deka G., "NoSQL Database for Storage and Retrieval of Data in Cloud," CRC Press. pp. 252, 2017.
- [47] Mohamed A., Obay G., Ismail M., "Relational vs. NoSQL databases: A survey," International Journal of Computer and Information Technology, Vol. 3, No. 3, pp. 598–601, 2014.
- [48] Ahmed J., Gulmeher R., "NoSQL databases: New trend of databases, emerging reasons, classification and security issues," International Journal of Engineering Sciences & Research Technology, Vol. 4, No. 6, pp. 176–184, 2014.
- [49] Factor P., *NoSQL: Are You Ready to Compromise with Security* (1/4/2013) [Online]. Available: <http://www.sqlservercentral.com/>
- [50] Ebrahim S., Mohammad A. N., "Survey on security issues in Big Data and NoSQL," ACSIJ Advances in Computer Science: An International Journal, Vol. 4, No. 4, pp. 68–72, 2015.
- [51] Patil H. S., Mukhtar Y. S., "Distributed database: An relevance to business organization," Journal of Information and Operations Management, Vol. 2, No. 1, pp. 21–24, 2011.
- [52] Deka G., "NoSQL Database for Storage and Retrieval of Data in Cloud," CRC Press. pp. 257, 2017.
- [53] Carlo B., Basso A., Giusto C., "Kerberos protocol: an overview," Distributed Systems, Italy, (2002).
- [54] Deka G., "NoSQL Database for Storage and Retrieval of Data in Cloud," CRC Press. pp. 258, 2017.
- [55] Deka G., "NoSQL Database for Storage and Retrieval of Data in Cloud," CRC Press. pp. 260, 2017.
- [56] Jose M., Serrão C., "Security and privacy issues of big data, Handbook of research on trends and future directions in big data and web intelligence," IGI Global, pp. 20-52, 2015.
- [57] MongoDB Official Website, [Online]. Available: <http://mongodb.com/>.

- [58] Chodorow K., Dirolf M., “MongoDB - The Definitive Guide: Powerful and Scalable Data Storage,” O'Reilly, 2010.
- [59] *MongoDB Documentation*, [PDF]. Available: <http://docs.mongodb.org/v3.6/MongoDB-manual.pdf/>
- [60] *MongoDB Query Manual* [Online]. Available: <https://docs.mongodb.com/manual/tutorial/query-documents/>
- [61] *MongoDB Crud Operations* [Online]. Available: <https://docs.mongodb.com/manual/crud/>
- [62] *MongoDB Previews New Features at Global User Conference*, [Online]. Available: <https://www.mongodb.com/press/new-features-at-global-user-conference>
- [63] *MongoDB Connector for BI*, [Online]. Available: [https:// docs.mongodb.org /bi-connector/](https://docs.mongodb.org/bi-connector/)
- [64] *MongoDB Replication*, [Online]. Available: <https:// docs.mongodb.com /manual/replication/>
- [65] *MongoDB Sharding*, [Online]. Available: <https://docs.mongodb.com /manual /sharding/>
- [66] *Facebook's Cassandra paper, annotated and compared to Apache Cassandra 3.0*, [Online]. Available: <https:// docs.datastax.com /en/articles /Cassandra /cassandrathenandnow.html>
- [67] Lakshman, A., Malik P., “Cassandra-A Decentralized Structured Storage System,” ACM SIGOPS Operating Systems Review, Vol. 44, No. 2, pp. 35-40, 2010.
- [68] Hewitt E., “Cassandra - The Definitive Guide: Distributed Data at Web Scale,” O'Reilly Media Inc., 2016.
- [69] *Turnable Consistency*, [Online]. Available: https:// teddyma.gitbooks.io /learncassandra/content/replication/turnable_consistency.html

- [70] *Cassandra DataStax Documentation*, [Online]. Available: <https://docs.datastax.com/en/cassandra/3.0/cassandra/operations/opsRepairNodesHintedHandoff.html>
- [71] *Cassandra Documentation - Bloom Filters*, [Online]. Available: http://cassandra.apache.org/doc/latest/operating/bloom_filters.html
- [72] Artem C., Kashlev A., Lu S., "A big data modeling methodology for apache Cassandra," in *IEEE International Congress on Big Data*, 2015.
- [73] *Cassandra DataStax Documentation – Data Distributed Replication*, [Online]. Available: https://docs.datastax.com/en/Cassandra/2.1/Cassandra/architecture/architectureDataDistributeReplication_c.html
- [74] Ganesh A. J., Kermarrec A., Massoulie L., "Peer-to-Peer Membership Management for Gossip-Based Protocols," *IEEE Trans. Computers*, Vol. 52, No. 2, pp. 139-149, 2003.
- [75] Apache Hive, [Online]. Available: <https://hive.apache.org/>
- [76] Apache Pig, [Online]. Available: <https://pig.apache.org/>
- [77] Dietrich F., "Cassandra: Principles and application," Department of Computer Science, University of Illinois, Urbana-Champaign, unpublished, 2010.
- [78] Deka G., "NoSQL Database for Storage and Retrieval of Data in Cloud," CRC Press. pp. 260, 2017.
- [79] Factor, Michael, et al. "Secure Logical Isolation for Multi-tenancy in cloud storage." *Mass Storage Systems and Technologies (MSST)*, 2013 IEEE 29th Symposium on. IEEE, 2013
- [80] Deka G., "NoSQL Database for Storage and Retrieval of Data in Cloud," CRC Press. pp. 261, 2017.
- [81] *MongoDB SCRAM Protocol*, [Online]. Available: <https://docs.mongodb.com/manual/core/security-scram/>
- [82] *MongoDB Security*, [Online]. Available: <https://docs.mongodb.com/manual/core/security-mongodb-ct/>

- [83] *Use x.509 Certificates to Authenticate Clients*, [Online]. Available: <https://docs.mongodb.com/manual/tutorial/configure-x509-client-authentication/>
- [84] *Configure MongoDB for FIP*, [Online]. Available: <https://docs.mongodb.com/manual/tutorial/configure-fips/>
- [85] *Kerberos Authentication*, [Online]. Available: <https://docs.mongodb.com/manual/core/kerberos/>
- [86] *Authenticate Using SASL and LDAP with OpenLDAP*, [Online]. Available: <https://docs.mongodb.com/manual/tutorial/configure-ldap-sasl-openldap/>
- [87] *Cassandra DataStax Documentation*, [Online]. Available: <https://docs.datastax.com/en/cassandra/3.0/cassandra>
- [88] Deka G., "NoSQL Database for Storage and Retrieval of Data in Cloud," CRC Press. pp. 262, 2017.
- [89] *Role-Based Access Control*, [Online]. Available: <https://docs.mongodb.com/manual/core/authorization/>
- [90] Schumacher R., *A Quick Tour of Internal Authentication and Authorization Security in DataStax Enterprise and Apache Cassandra* (26/2/2013), [Online]. Available: <https://www.datastax.com/dev/blog/a-quick-tour-of-internal-authentication-and-authorization-security-in-datastax-enterprise-and-apache-cassandra>
- [91] Badkar P., *Oracle Database Security Guide 11g Release 1*, [PDF]. Available: http://docs.oracle.com/cd/B28359_01/network.111/b28531.pdf
- [92] Meier J., Mackman A., Dunner M., Vasireddy S., Escamilla R., Murukan A., "Chapter 2: Threats and Countermeasures," [Online]. Available: msdn.microsoft.com/en-us/library/ff648641.aspx
- [93] *MongoDB Auditing*, [Online]. Available: [https:// docs.mongodb.com /manual /core/auditing/](https://docs.mongodb.com/manual/core/auditing/)
- [94] *Enabling Data Auditing in DataStax Enterprise*, [Online]. Available: https://docs.datastax.com/en/datastax_enterprise/5.0/datastax_enterprise/sec/auditEnabling.html/

- [95] *Transport Encryption*, [Online]. Available: <https://docs.mongodb.com/manual/core/security-transport-encryption/>
- [96] *Cassandra DataStax Documentation – SSL Authentication*, [Online]. Available: <https://docs.datastax.com/en/Cassandra/3.0/Cassandra/configuration/secureSSLIntro.html/>
- [97] *Cassandra DataStax Documentation*, [Online]. Available: https://docs.datastax.com/en/dse/5.1/dse-admin/datastax_enterprise/security/secEncryptLocalKeys.html/
- [98] Deka G., “NoSQL Database for Storage and Retrieval of Data in Cloud,” CRC Press. pp. 259, 2017.
- [99] *Cassandra DataStax Documentation - Encryption at Rest*, [Online]. Available: <https://docs.mongodb.com/manual/core/security-encryption-at-rest/>
- [100] *Cassandra DataStax Documentation – Encryption Configuration*, [Online]. Available: https://docs.datastax.com/en/datastax_enterprise/5.0/datastax_enterprise/sec/encryptConfig.html/
- [101] *Cassandra DataStax Documentation – JMX Authentication*, [Online]. Available: <https://docs.datastax.com/en/cassandra/2.1/cassandra/security/secureJmxAuthentication.html>
- [102] *Cassandra DataStax Documentation – Authentication Caching*, [Online]. Available: https://docs.datastax.com/en/dse/6.0/dse-admin/datastax_enterprise/security/secAuthCacheSettings.html
- [103] *Configuring proxy roles for applications*, [Online]. Available: https://docs.datastax.com/en/dse/5.1/dse-admin/datastax_enterprise/security/secProxy.html
- [104] *TLS/SSL Configuration for Clients*, [Online]. Available: <https://docs.mongodb.com/manual/tutorial/configure-ssl-clients>
- [105] *Cassandra Node to Node Encryption*, [Online]. Available: https://docs.datastax.com/en/cassandra/2.1/cassandra/security/secureSSLNodeToNode_t.html

- [106] *Cassandra Client to Node Encryption*, [Online]. Available: <https://docs.datastax.com/en/cassandra/3.0/cassandra/configuration/secureSSLClientToNode.html>
- [107] *Yahoo Cloud Service Benchmark tool (YCSB)*, [Online]. Available: <https://research.yahoo.com/news/yahoo-cloud-serving-benchmark/?guccounter=1>
- [108] Brian F. C. et al., "Benchmarking cloud serving systems with YCSB," *Proc. 2010 ACM symposium on Cloud computing Conf.*
- [109] *MongoDB Official Website: Installation*, [Online]. Available: <https://docs.mongodb.com/manual/installation/>
- [110] *MognDB Official Website: Shard Installation*, [Online]. Available: <https://docs.mongodb.com/manual/tutorial/deploy-shard-cluster/>
- [111] *Cassandra Official Website*, [Online]. Available: <http://cassandra.apache.org>
- [112] *Cassandra Official Website: Installation*, [Online]. Available: http://cassandra.apache.org/doc/latest/getting_started/installing.html
- [113] *Cassandra Official Website: Configuration file*, [Online]. Available: http://cassandra.apache.org/doc/latest/configuration/cassandra_config_file.html?highlight=cluster
- [114] *MongoDB official website: Security*, [Online]. Available: <https://docs.mongodb.com/manual/security/>
- [115] *Cassandra Official Website: Security*, [Online]. Available: <http://cassandra.apache.org/doc/latest/operating/security.html>
- [116] *Cassandra Enterprise Official Website: Security*, [Online]. Available: [https:// docs.datastax.com/en/dse/5.1/dse-admin /datastax_enterprise /security / securityTOC.html](https://docs.datastax.com/en/dse/5.1/dse-admin/datastax_enterprise/security/securityTOC.html)
- [117] *Datastax Official Website*, [Online]. Available: <https://www.datastax.com>.
- [118] *Apache Thrift Framework*, [Online]. Available: <https://thrift.apache.org/>