

# **DISTRIBUTED DATABASE USING MONGODB**

A project report submitted for the partial fulfilment of the **Bachelor  
of Technology Degree**

in

**Computer Science & Engineering** under  
**Maulana Abul Kalam Azad University of Technology** by  
**Hilor Jain**

Roll No: 10400116173, Registration Number: 161040110074

**Subham Agarwalla**

Roll No: 10400116064, Registration Number: 161040110183

**Academic Session: 2016-2020**

Under the Supervision of  
**Prof. Ee-Kian Wong**



Department of Computer Science and Engineering Institute of Engineering & Management

Y-12, Salt Lake, Sector 5, Kolkata, Pin 700091, West Bengal, India

Affiliated To



*Maulana Abul Kalam Azad University of Technology, West Bengal*  
*formerly known as* **West Bengal University of Technology**  
*In Pursuit of Knowledge and Excellence*

Maulana Abul Kalam Azad University of Technology

BF 142, BF Block, Sector 1, Kolkata, West Bengal 700064

**MAY 2020**



**INSTITUTE  
OF ENGINEERING & MANAGEMENT**  
Salt Lake Electronics Complex, Kolkata - 700091, WB, INDIA

Phone : (033) 2357 2969/2059/2995  
(033) 2357 8189/8908/5389  
Fax : 91 33 2357 8302  
E-mail : director@iemcal.com  
Website : www.iemcal.com

## CERTIFICATE

### TO WHOM IT MAY CONCERN

This is to certify that the project report titled “**Distributed Database using mongoDB**”, submitted by **Hilor Jain, Roll No: 10400116173, Registration Number: 161040110074**, **Subham Agarwalla, Roll No: 10400116064, Registration Number: 161040110183**, students of **Institute of Engineering & Management** in partial fulfillment of requirements for the award of the degree of **Bachelor of Technology in Computer Science & Engineering**, is a bona fide work carried out under the supervision of **Prof. Ee-Kian Wong** during the final year of the academic session of 2016-2020. The content of this report has not been submitted to any other university or institute for the award of any other degree.

It is further certified that the work is entirely original and the performance has been found to be satisfactory.

**Prof. Ee-Kian Wong**

Assistant Professor

Department of Computer Science and Engineering

Institute of Engineering & Management

**Prof.(Dr.) Sourav Saha**

H.O.D.

Department of Computer Science and Engineering

Institute of Engineering & Management

**Prof.(Dr.) Amlan Kusum Nayak**

Principal

Institute of Engineering & Management

Gurukul Campus: Y-12, Salt Lake Electronics Complex, Sector-V, Kolkata 700091, Phone: (033) 2357 2969

Management House: D-1, Salt Lake Electronics Complex, Sector-V, Kolkata 700091, Phone: (033) 2357 8908

Ashram Building: GN-34/2, Salt Lake Electronics Complex, Sector-V, Kolkata 700091, Phone: (033) 2357 2059/2995

INSTITUTE OF ENGINEERING & MANAGEMENT



DECLARATION FOR NON-COMMITMENT  
OF PLAGIARISM

We, **Hilor Jain, Subham Agarwalla** students of B.Tech. in the Department of Computer Science and Engineering, Institute of Engineering & Management have submitted the project report in partial fulfillment of the requirements to obtain the above noted degree. We declare that we have not committed plagiarism in any form or violated copyright while writing the report and have acknowledged the sources and/or the credit of other authors wherever applicable. If subsequently it is found that we have committed plagiarism or violated copyright, then the authority has full right to cancel/reject/revoke our degree.

Name of the Student: HILOR JAIN

Full Signature: \_\_\_\_\_

Name of the Student: SUBHAM AGARWALLA

Full Signature: \_\_\_\_\_

Date: 09.07.2020

# TABLE OF CONTENTS

<b>CHAPTER 1. INTRODUCTION</b>	<b>6</b>
Objective	6
<b>CHAPTER 2: BACKGROUND</b>	<b>8</b>
Scaling	9
Sharded Cluster	10
Replica Set	11
Shard Keys	12
Gather and Scatter	13
<b>CHAPTER 3. PROPOSED FRAMEWORK</b>	<b>14</b>
Hardware Configuration	15
Comparing MongoDB with Sqlite	18
Testing Setup	18
<b>CHAPTER 4. EXPERIMENTAL RESULTS AND ANALYSIS</b>	<b>20</b>
BaseLine Test Analysis	20
Comparing Results	22
Load Test Analysis	23
Industry Benchmarks	25
<b>Chapter 5 Conclusion:</b>	<b>26</b>
Summary	26
Future Work	26
<b>Chapter 6 References</b>	<b>28</b>

## LIST OF FIGURES AND TABLES

Figure 1: Scaling.....	10
Figure 2: Shard cluster.....	11
Figure 3: Replica sets.....	12
Table 1: Comparison between the methodologies.....	14
Figure 4: MongoDB Configuration.....	16
Figure 5a,b: Adding shards to the mongos.....	17
Figure 7: Sqlite result 1.....	20
Figure 8: MongoDB result 1.....	21
Figure 9: Sqlite results 2.....	21
Figure 10: MongoDB result 2.....	22
Table 2: Comparison Test analysis.....	22
Table 3: Load Test Analysis.....	23
Figure 10: MongoDB load test plan.....	23
Figure 11: MongoDB Read-Write Load Test Results .....	23
Figure 12: Sqlite load test plan.....	24
Figure 13: Sqlite read load test results.....	24
Figure 14: Sqlite write load test results.....	24

# CHAPTER 1. INTRODUCTION

Distributed and parallel processing<sup>[6]</sup> on database management systems (DBMS) is an efficient way of improving performance of applications that manipulate large volumes of data. This may be accomplished by removing irrelevant data accessed during the execution of queries and by reducing the data exchange among sites, which are the two main goals of the design of distributed databases.

Also, many recent problem domains are supported by applications that are typically more complex than traditional applications, in addition to their great volume of data. Those applications require, at least in the conceptual level, a semantically richer data model which is capable of directly representing complex structures and operations in a more natural and adequate manner, such as the object data model. Therefore, in order to improve performance of those applications, it is very important to design information distribution properly, and take the application semantics into account as much as possible.

One of the main needs of health care systems is related to the ability to manage and process large volumes of data stored in heterogeneous clinical repositories. This issue is a topic of interest in the field of health informatics. This project describes a new alternative for the clinical information retrieval, stored in HL7 Clinical Document Architecture repository, using a document database, NoSQL, schema free, in a laboratory environment<sup>[2]</sup>.

## Objective

The basic objectives of our project are

- to store a clinical dataset in a mongoDB database and shard it into different nodes. We also aim to ensure that we are able to retrieve and access records in the dataset effectively.
- to evaluate the performance of the distributed mongoDB database system against a sequel database system like sqlite with the same records by using read write baseline tests and load tests.

Our project is based on a paper entitled “T1 - MongoDB: An open source alternative for HL7-CDA clinical documents management DO - 10.13140/RG.2.1.3033.7128”<sup>[1]</sup> where the author explains the importance of

using a nosql database like mongo db over typical xml file storage systems in storing clinical datasets. However here we aim to implement a sharded nosql mongoDB system to exhibit the importance of parallelism through horizontal scaling. Clinical datasets are very important because they store records of millions of patients, which can be used to find underlying patterns among patients of a disease and track everyday reports of a particular patient at miniscule level. Hence efficient storage of clinical data with high throughput and easy access can prove to be a boom to medical sciences.

## CHAPTER 2: BACKGROUND

In 2007, Eliot Horowitz and Dwight Merriman invented MongoDB<sup>[8]</sup> as a database designed to be able to scale horizontally, handle large heterogeneous data with ease and be stored at many physical locations globally.

Today, MongoDB is a hugely popular open source technology, managed by a software company of the same name.

MongoDB is used by some of the largest companies in the world, including Facebook, Google, Nokia, MTV Networks, Cisco, Forbes, and many more.

MongoDB is an open source database management system (DBMS) that uses a document-oriented data model. It is considered a NoSQL database, as it doesn't use the relational model, and therefore doesn't use SQL as its query language.

MongoDB is also a cross platform DBMS, currently supporting Windows, Mac, Solaris, and various Linux distributions at the time of writing. A MongoDB database is different to a relational database in that MongoDB uses a document-oriented model to store data. In the document-oriented model, data is stored within documents of a collection. In the relational model, data is stored within rows of a table.

In MongoDB, documents are stored as JSON documents. JSON (JavaScript Object Notation) is a standard that facilitates data interchange. By using JSON, query results can be easily parsed, with little or no transformation, directly by JavaScript and most popular programming languages. This is because JSON documents use the name/pair, and array conventions that are familiar to most popular programming languages such as C, C++, C#, Java, JavaScript, Perl, Python, and many others. This reduces the amount of business logic that needs to be built into applications that use MongoDB.

Clinical Document Architecture (CDA)<sup>[9]</sup> is a popular, flexible markup standard developed by Health Level 7 International (HL7 ) that defines the structure of certain medical records, such as discharge summaries and progress notes, as a way to better exchange this information between



providers and patients. These documents can include text, images and other types of multimedia -- all integral parts of electronic health records (EHRs).

MongoDB is the leader in a new generation of databases that are designed for scalability. With a technique called “sharding” you are able to easily distribute data and grow your deployment over inexpensive hardware or in the cloud. Scaling a database system is complex but MongoDB makes it easy. Lets understand them in dept.

## Scaling

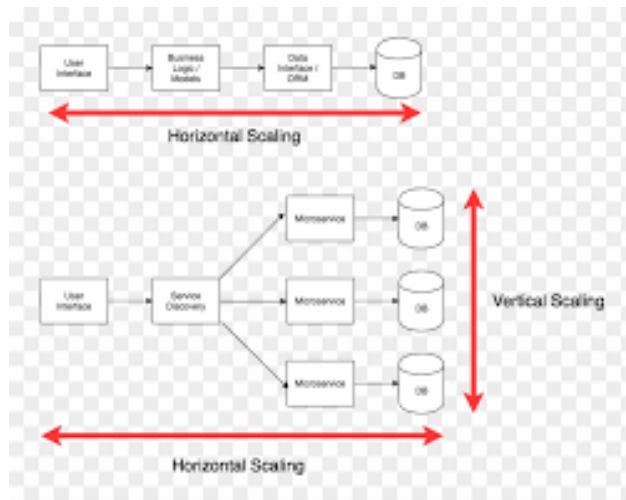
There are two methods for addressing system growth: vertical and horizontal scaling<sup>[2]</sup>.

*Vertical Scaling* involves increasing the capacity of a single server, such as using a more powerful CPU, adding more RAM, or increasing the amount of storage space. Limitations in available technology may restrict a single machine from being sufficiently powerful for a given workload. Additionally, Cloud-based providers have hard ceilings based on available hardware configurations. As a result, there is a practical maximum for vertical scaling.

*Horizontal Scaling* involves dividing the system dataset and load over multiple servers, adding additional servers to increase capacity as required. While the overall speed or capacity of a single machine may not be high, each machine handles a subset of the overall workload, potentially providing better efficiency than a single high-speed high-capacity server. Expanding the capacity of the deployment only requires adding additional servers as needed, which can be a lower overall cost than high-end hardware for a single machine. The trade off is increased complexity in infrastructure and maintenance for the deployment. MongoDB supports horizontal scaling through sharding.

The given figure briefly explains scaling:

Figure 1: Scaling



The basic structure of a distributed database also known as shard cluster comprises of three things. One are the shards, two are the mongos or the routers and thirdly the config servers or the brain of the server. Our system will have 3 shards each replicated, 1 mongos and 1 config server.

## Sharded Cluster

Sharding<sup>[2]</sup> is a method for distributing data across multiple machines. MongoDB uses sharding to support deployments with very large data sets and high throughput operations.

Database systems with large data sets or high throughput applications can challenge the capacity of a single server. For example, high query rates can exhaust the CPU capacity of the server. Working set sizes larger than the system's RAM stress the I/O capacity of disk drives.

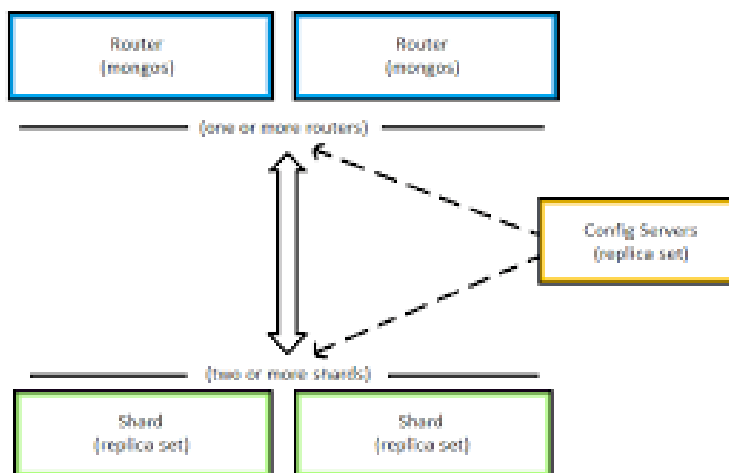
A MongoDB sharded cluster consists of the following components:

- shard: Each shard contains a subset of the sharded data. Each shard can be deployed as a replica set.
- mongos: The mongos acts as a query router, providing an interface between client applications and the sharded cluster.

- config servers: Config servers store metadata and configuration settings for the cluster. As of MongoDB 3.4, config servers must be deployed as a replica set (CSRS).

The following graphic describes the interaction of components within a sharded cluster:

Figure 2: shard cluster



As we said each shard,config server in turn can be a replica set.

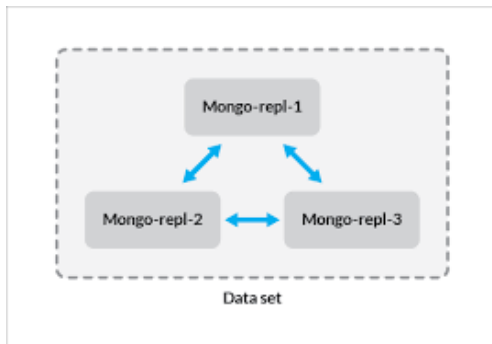
## Replica Set

A replica set<sup>[5]</sup> is a group of mongod instances that maintain the same data set. A replica set contains several data bearing nodes and optionally one arbiter node. Of the data bearing nodes, one and only one member is deemed the primary node, while the other nodes are deemed secondary nodes. The primary node receives all write operations.

A replica set can have only one primary capable of confirming writes with { w: "majority" } write concern; although in some circumstances, another mongod instance may transiently believe itself to also be primary. The primary records all changes to its data sets in its operation log,

i.e. oplog. For more information on primary node operation, see Replica Set Primary.

Figure 3: Replica Set



Like keys in a normal database each document in a distributed database is indexed. This is done using shard keys.

## Shard Keys

MongoDB uses the shard key<sup>[2]</sup> to distribute the collection's documents across shards. The shard key consists of a field or fields that exist in every document in the target collection.

You choose the shard key when sharding a collection. The choice of shard key cannot be changed after sharding. A sharded collection can have only *one* shard key. See Shard Key Specification.

To shard a non-empty collection, the collection must have an index that starts with the shard key. For empty collections, MongoDB creates the index if the collection does not already have an appropriate index for the specified shard key. See Shard Key Indexes.

## Gather and Scatter

When a query comes to a mongos, it is based on the information from the config server [scatters] the query information<sup>[4]</sup>. This process is known as scatter or scattering of queries to the shard chunks. Each shard chunk then

processes the query and generates result based in the resources it has. The generated result is then collected back to the mongos from the shards called gather of results. The mongos process these collected data and sends the result to the end user<sup>1</sup>.

Like replicated and sharded systems, the scatter/gather pattern<sup>[3]</sup> is a tree pattern with a root that distributes requests and leaves that process those requests. However, in contrast to replicated and sharded systems, scatter/gather requests are simultaneously farmed out to all of the replicas in the system. Each replica does a small amount of processing and then returns a fraction of the result to the root. The root server then combines the various partial results together to form a single complete response to the request and then sends this request back out to the client.

## CHAPTER 3. PROPOSED FRAMEWORK

Our project tries to build a distributed database using mongoDB. Dataset here used is a clinical dataset implemented in hl7cda standards. Distributed databases in mongoDB is popularly implemented using sharding. It is nothing but a way to implement horizontal scaling or parallel processing in databases.

The paper <sup>[1]</sup> we referred to has used a sample of 1,000,000 random clinical documents collected from an Italian Hospital clinical data repository (based on the standard HL7-CDA) which contains a little more than 22 million documents. We have used a hl7cda clinical depository patient.csv containing records of 1181 patients.

The following table explains the difference in methodology/materials used by them to us.

Table 1: Comparison between the methodologies

Sl no.	Methodology	Used in the paper	Used in this project
1	Data size	1000000 records of clinical data comprising 31% of lab results, 22% of clinical notes, 28% results of imaging studies, 10% of drug prescriptions and 9% of nursing sheets.	1181 records of clinical data described under 25 fields.
2	Data type	Heterogeneous data type consisting of textual, numeric data types and graphic images.	Homogeneous data type consisting of texts and numbers

3	Number of nodes/shards	1 main unsharded collection	1 collection sharded into 3 shards with replica set each (1 primary and 1 secondary); 1 mongos ; 1 config server
4	Comparisons	Comparison and evaluation of performance made between a standard nosql mongodb system to a typical xml system	Comparison and evaluation of performance made between a distributed nosql mongodb system with a sequel sqlite database system
5	Testing methodology	load test,soak test,baseline test,stress test were performed.	baseline test and load test were performed.

## Hardware Configuration

The whole setup is on a single local host built using a virtual box with the following configuration

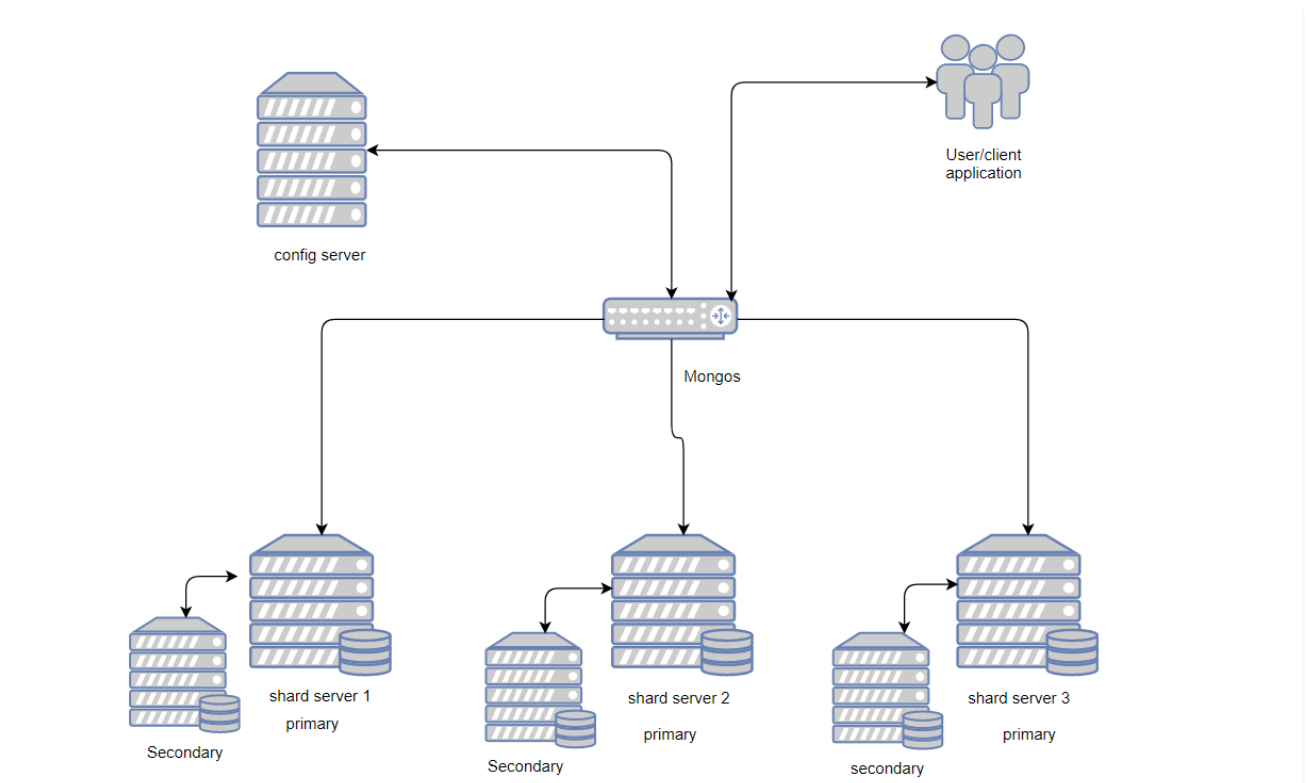
- Virtual box: Oracle VM virtual box
- Version:6.1
- Operating system: Ubuntu (64 bits) linux
- Base Memory: 1 GB
- Processor: Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz, 1800 Mhz, 4 Core(s), 8 Logical Processor(s)

## MongoDb Configuration

As mentioned in table 1 our system consists of three shard servers each with a replica set consisting of 1 primary and 1 secondary, 1 config server and 1 mongos. The replica set helps to maintain high data availability and store the exact replica of the shards. The mongos is the router which acts as an interface between the client application and the shards. The mongos collect queries from the client and scatter it to the shards. Each shard performs necessary actions and relays back to the mongos called gathering of results. The config servers store metadata of the system or the data about the data and helps the mongos to perform efficiently. More about this is explained on page 12 under the heading shard cluster.

Figure 4 describes the configuration of our distributed system.

Figure 4: MongoDB configuration Diagram



The following commands shows how shards are added to our mongos:



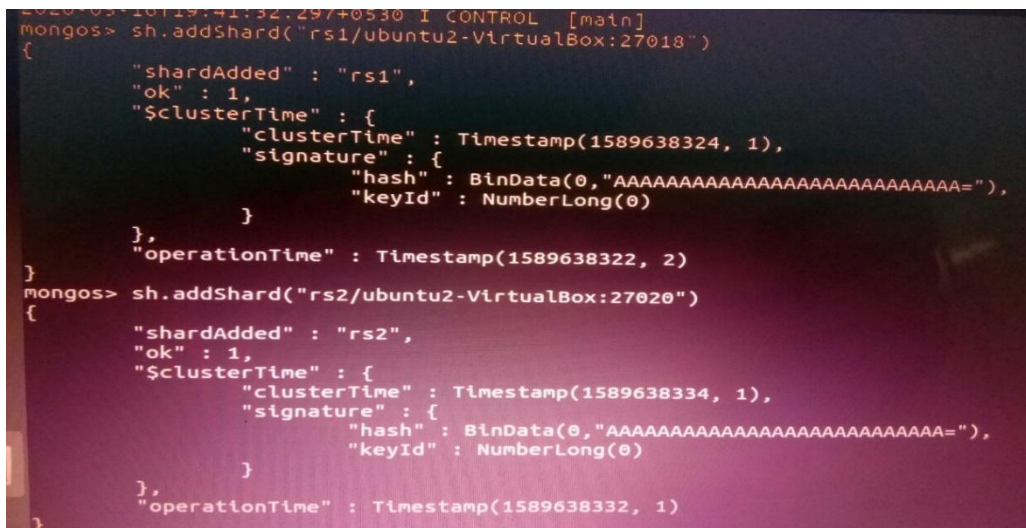
Adding shards to mongos:

```
sh.addShard("rs1/ubuntu-virtualbox:27018");
```

```
sh.addShard("rs2/ubuntu-virtualbox:27020");
```

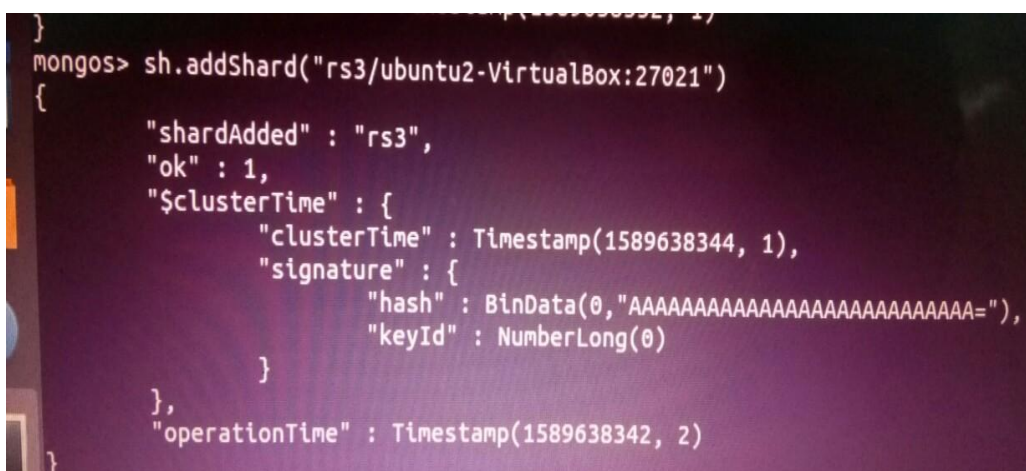
```
sh.addShard("rs3/ubuntu-virtualbox:27021")
```

Figure 5a: Adding shards to mongos



```
2020-05-10T19:41:32.297+0530 I CONTROL [main]
mongos> sh.addShard("rs1/ubuntu2-VirtualBox:27018")
{
  "shardAdded" : "rs1",
  "ok" : 1,
  "$clusterTime" : {
    "clusterTime" : Timestamp(1589638324, 1),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  },
  "operationTime" : Timestamp(1589638322, 2)
}
mongos> sh.addShard("rs2/ubuntu2-VirtualBox:27020")
{
  "shardAdded" : "rs2",
  "ok" : 1,
  "$clusterTime" : {
    "clusterTime" : Timestamp(1589638334, 1),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  },
  "operationTime" : Timestamp(1589638332, 1)
}
```

Figure 5b:



```
mongos> sh.addShard("rs3/ubuntu2-VirtualBox:27021")
{
  "shardAdded" : "rs3",
  "ok" : 1,
  "$clusterTime" : {
    "clusterTime" : Timestamp(1589638344, 1),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  },
  "operationTime" : Timestamp(1589638342, 2)
}
```

## Comparing MongoDB with Sqlite

The project tries to explain the role of distributed systems in the field of clinical data storage. We tried to shard our database in mongoDB and compare it to the same dataset when stored in sqlite and bring a comparison.

**Sqlite structure:** Sqlite used here has the same records as that of mongoDB as it is used for comparison of performance. Except unlike mongoDB ,sqlite uses tables so it has one table with cardinality 1181 and degree

25. MongoDB with same records is sharded and represents data in json like format for easy user understanding and it becomes easy to parse it into javascripts and other applications for future use.

The following mentioned queries were used to compare the performance of the same.

## Testing Setup

For the analysis of the performance some tests were used.

**Baseline Test:** This first test was run on the database created for analysis. without any optimization, consultations were carried out with a minimum number of concurrent users, to determine a baseline of performance.

**Load Test:** Once the causes of performance problems obtained from the initial test are established, necessary actions are implemented and Load Test runs simulating the concurrency of users.

For load test 3 sets of 1000 requests each were sent to mongoDB. A software tool called Apache Jmeter was used to measure the performance of the database by measuring the average response time, minimum response time, maximum response time, percentage of errors and throughput of the same. Throughput here refers to the number of requests per second the server handles.

Response time refers to the amount of time Application Server takes to return the results of a request to the user. The response time is affected by factors such as network bandwidth, number of users, number and type of requests submitted, and average think time.

**Apache JMeter** is a testing tool used for analyzing and measuring the performance of different software services and products. It is a pure Java open source software used for testing Web Application or FTP application.

### **Q1. Read Operation Test**

#### **Sqlite:**

```
mode csv patient .import file_name.csv patient select * from patient;
```

#### **MongoDB:**

```
use hospital db.patient.find().explain("executionStats");
```

### **Q2. Write operation Test**

#### **Sqlite:**

```
update patient set address="Kolkata";
```

#### **Mongoddb:**

```
db.patient.updateMany({},{$set: {"address": "Kolkata"}});
```

## CHAPTER 4. EXPERIMENTAL RESULTS AND ANALYSIS

As mentioned above here are the results of read write tests comparing mongoDB and sqlite comparing their results based on execution time.

### BaseLine Test Analysis

#### Q1 :Read operation Test:

##### Sqlite:

Figure 7: Sqlite results 1

```
e65fdd89-d593-41ab-a780-f42f3792a44f,1951-05-24,2005-08-18,999-64
",",M,white,italian,M,"Weymouth Massachusetts US","419 Bahrin
,42.12660414448737,-70.97767848117874,1290278.46,8400.719999999999
955b1b61-e049-4048-a42a-a45202060cf9,1951-05-24,"",999-70-1676,S9
alian,M,"Canton Massachusetts US","764 Beahan Dam Unit 4",Brock
3909,-70.9165736733102,1501104.56,7568.99
Run Time: real 9.100 user 0.687500 sys 1.468750
sqlite>
sqlite> .open filename to reopen on a persistent database.
sqlite> .mode csv patient
sqlite> .import C:/Users/pratham/Downloads/patients.csv patient
sqlite> select * from patient;
d171d808-1f31-4ad3-aba5-e03a2fa921c7,1983-04-03,"",999-83-6585,S99914532,X185
",M,hispanic,mexican,M,"La Paz Baja California MX","552 Rippin Port",Reve
49384661818001,-70.92858466579901,748893.99,2679.68
3603cd65-53a3-424b-bc82-76204326510d,1977-06-24,"",999-62-4132,S99987902,X3324
white,irish,M,"Southbridge Massachusetts US","1027 Satterfield Spur Unit 67"
",42.3901355236465,-73.32437332707846,870371.41,5315.999999999999
bd512b17-9e68-4b0e-8c5b-980007ecdee1,1991-10-21,"",999-65-7753,S99933589,X6000
```

Execution time: 687ms.

##### MongoDB:

Figure 8: MongoDB results1

```
mongos> db.patient.find().explain("executionStats")
{
  "queryPlanner" : {
    "mongosPlannerVersion" : 1,
    "winningPlan" : {
      "stage" : "SHARD_MERGE",
      "shards" : [
        {
          "shardName" : "rs3",
          "connectionString" : "mongodb://rs3:27021",
          "serverInfo" : {
            "host" : "rs3",
            "port" : 27021,
            "version" : "3.6.10"
          }
        }
      ]
    }
  },
  "executionStats" : {
    "nReturned" : 1181,
    "executionTimeMillis" : 67,
    "totalKeysExamined" : 0,
    "totalDocsExamined" : 1181,
    "executionStages" : [
      {
        "stage" : "SHARD_MERGE",
        "nReturned" : 1181,
        "executionTimeMillis" : 67,
        "totalKeysExamined" : 0,
        "totalDocsExamined" : 1181,
        "totalChildMillis" : NumberLong(85),
        "shards" : [
          {
            "shardName" : "rs3",
            "executionSuccess" : true,
            "executionTimeMillis" : 67,
            "totalKeysExamined" : 0,
            "totalDocsExamined" : 1181,
            "totalChildMillis" : NumberLong(85)
          }
        ]
      }
    ]
  }
}
```

**Execution time:** 67ms.

## Q2: Write Operation Test

**Sqlite:**

Figure 9: Sqlite results 2

```
Run Time: real 12.366 user 0.671875 sys 1.187500
sqlite> update patient set ADDRESS="Kolkata";
Run Time: real 0.070 user 0.031250 sys 0.015625
sqlite>
```

**Execution time:** 31ms.



## MongoDB:

Figure 10: MongoDB results 2

```
2020-05-17T10:44:43.196+0530 E QUERY [thread1] TypeError: db.patient.inse
...).explain is not a function :
@(shell):1:1
mongos> db.patient.updateMany({},{$set:{"ADDRESS":"Siliguri"}})
{ "acknowledged" : true, "matchedCount" : 1182, "modifiedCount" : 1182 }
  "executionTimeMillis" : 21,
  "totalKeysExamined" : 0,
  "totalDocsExamined" : 1182,
  "executionStages" : {
    "stage" : "SHARD_MERGE",
    "nReturned" : 1182,
    "executionTimeMillis" : 21,
    "totalKeysExamined" : 0,
    "totalDocsExamined" : 1182,
    "totalChildMillis" : NumberLong(18),
    "shards" : [
```

**Execution time:** 21ms.

## Comparing Results

Table 2: Comparison Results Analysis

	MongoDB	Sqlite
Read Test execution time	67ms.	687ms
Write Test execution time	21ms	31ms.

Inference: The above results suggest mongoDB is 8.15 times faster than sqlite.

## Load Test Analysis

**Table 3:** Load test analysis

Load Test	Measures	MongoDB	Sqlite
Read test	No. of requests	1000	1000
	Throughput	194.3/s	62.8/s
Write Test	No. of requests	1000	1000
	Throughput	843/s	71.5/s

Inference: The above results in load test suggest MongoDB is 7.79 times faster than sqlite

### Test Plan and test results

Figure 11: MongoDB Load Test plan

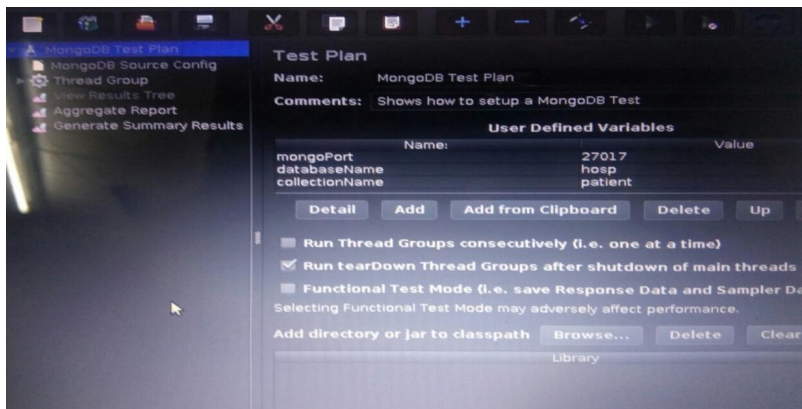


Figure 12: MongoDB Read-Write Load Test Results

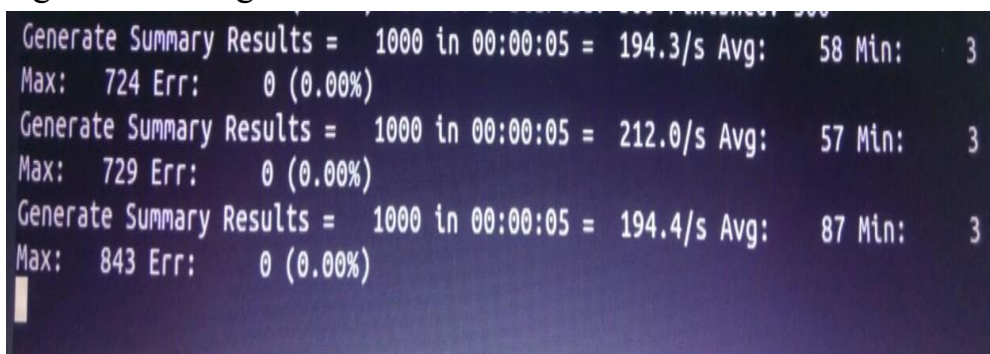


Figure 13 : Sqlite load test plan:

Name: JDBC Connection Configuration

Comments:

Variable Name Bound to Pool:

Variable Name for created pool: demo test

Connection Pool Configuration: Name of the JMeter variable that the pool will be bound to.

Max Number of Connections: 10

Max Wait (ms): 10000

Time Between Eviction Runs (ms): 60000

Auto Commit: True

Transaction Isolation: DEFAULT

Preinit Pool: False

Init SQL statements separated by new line:

1

Connection Validation by Pool:

Test While Idle: True

Soft Min Evictable Idle Time(ms): 5000

Validation Query: select 1 from dual

Database Connection Configuration:

Database URL: https://sqliteonline.com/#idde=22c986445b0707c946995cc0f6e1d7225976f6b42164cfafb5a5da60ee386fc

JDBC Driver class: oracle.jdbc.OracleDriver

Username: hilore05121999@gmail.com

Password: \*\*\*\*\*

Figure 14: Sqlite read load test results

Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename: Browse... Log/Display Only: ☐ Errors ☐ Successes Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/s...	Sent KB/sec	Avg. Bytes
JDBC Request	1000	1	0	11	1.55	0.00%	62.8/sec	3.43	0.00	56.0
TOTAL	1000	1	0	11	1.55	0.00%	62.8/sec	3.43	0.00	56.0

Figure 15:Sqlite write load test results

Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename: Browse... Log/Display Only: ☐ Errors ☐ Successes Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/s...	Sent KB/sec	Avg. Bytes
JDBC Request	1000	1	0	12	1.55	0.00%	71.5/sec	3.91	0.00	56.0
TOTAL	1000	1	0	12	1.55	0.00%	71.5/sec	3.91	0.00	56.0



## Industry Benchmarks

A comparison between MySQL and MongoDB was made by Simform Solutions Pvt limited, a software development company in USA<sup>[7]</sup>.

Measurements were performed in the following cases:

MySQL 5.7.9

MongoDB 3.2.0

Each of these has been tested on a separate m4.xlarge Amazon instance with the ubuntu 14.4 x64 and default configurations, all tests were performed for 1000000 records.

The study was performed for both read (select) and write (update) commands.

It was observed that mongoDB was 9.07 times faster for read commands and 55.52 times faster for write commands.

Our results show that mongoDB was about 8.15 (read write baseline tests) and 7.79 (read write load tests) times faster than sqlite. Although read results seem persistent write results show a variation from industrial bench marks.

## Chapter 5 Conclusion:

### Summary

We were able to build a distributed mongoDB system and store hl7cda clinical dataset into it. We were able to access records and perform all basic database dml operations. Our system consisted of three shards, one mongos and one config server and the whole setup was built on a single local host.

As for the evaluation part, we compared MongoDB against a sql database sqlite. Both read write baseline tests and load tests suggested MongoDB was approximately 8 times faster compared to sqlite. However our results showed slight deviation from industrial results in terms of write tests.

These variations in write operation tests could be due to

- a) The industrial test report <sup>[7]</sup> depicts results of comparison of mongoDB against mysql while we used sqlite for comparison, though both are sql databases in nature sqlite is faster than mysql.
- b) difference in the number of records for performance evaluation as larger test case result in better results
- c) difference in the type of records, mongoDB works better with heterogeneous data
- d) difference in the type of database commands used for evaluation, join and cross product operations need more time to compute.

### Future Work

In this project we tried to bring out the importance of using distributed nosql databases like mongoDB for clinical datasets. Although the results clearly showed mongoDB was faster than the sequel database, a more precise study can be made with a larger clinical record.

Tests can be done using heterogeneous data consisting of graphic images, textual data and numbers to bring out the real power of mongoDB.

Since we were only able to work with few read and write commands, better comparison can be drawn using other dml commands for a detailed study.

A complete distributed system with more physical hosts can be built, this will not only help us to improve the throughput of the system but will also give an idea how important is the role of parallelism in the clinical world.

## Chapter 6 References

- [1] T1 - MongoDB: An open source alternative for HL7-CDA clinical documents management DO - 10.13140/RG.2.1.3033.7128
- [2] <https://docs.mongodb.com/manual/core/distributed-queries/>
- [3] <https://www.oreilly.com/library/view/designing-distributed-systems/9781491983638/ch07.html>
- [4] <https://books.google.co.in/books?id=3V7BDwAAQBAJ&pg=PA497&lpg=PA497&dq=Gathering+and+scattering+in+distributed+database&source=bl&ots=1v97tejELj&sig=ACfU3U3S6SfN5kPqmPz5sGkvNbpz>
- [5] <https://docs.mongodb.com/manual/replication/>
- [6] <https://dataconomy.com/2014/08/distributed-nosql-mongodb/>
- [7] MongoDB vs MySQL: A Comparative Study on Databases by simform
- [8] <https://medium.com/s-c-a-l-e/mongodb-co-creator-explains-why-nosql-came-to-be-and-why-open-source-mastery-is-an-elusive-goal-3a138480b9cd>
- [9] <https://searchhealthit.techtarget.com/definition/Clinical-Document-Architecture-CDA>