COMPREHENSIVE COMPARISON OF RDBMS AND NOSQL DATABASES FOR HEALTHCARE APPLICATIONS

TERM PROJECT

Submitted by:

GROUP 1

Swetha Subramanian

Revathi Boopathi

Madhura Pandit

Instructor

Prof. Ming-Hwa Wang, Ph.D

San Jose State University

ACKNOWLEDGMENTS:

We would like to thank Professor Ming-Hwa Wang, Ph.D for providing us with this opportunity to work on a Database project and choose the topic of our interest. He also guided us in doing the related research work which enhanced our knowledge in this field. We would also like to thank him for the support that he has provided throughout the project. This gave us the opportunity to learn about various databases both theoretically and practically.

Additionally, we would like to thank our families for their support.

Table of Contents

List of Tables

Page#

List of Figures

Page#

● Abbreviations Used in the Project

Relational Database Management System	**RDBMS**
Database	**DB**
Mongo Query Language	**MQL**
Electronic Health Records	**EHR**
Electronic Medical Records	**EMR**

Table 1: List of Abbreviations

1. Abstract

The selection of a better performing database for a particular application is the most important task for organizations, especially in case of a Healthcare domain considering the variety of data and sources contributing. For a Medical Insurance Plan recommendation system, the read operation performance of a database needs to be fast and optimal. As NoSQL Databases have dominated the datastore collection now, RDBMS databases are still vastly used and relied on for many applications. In this project, we analyze the performance of a Document NoSQL store - MongoDB and RDBMS store - MySQL using Patient Records and Healthcare Plans data. The cloud version of both databases have been chosen - MongoDB Atlas and AWS - RDS. The performance evaluation of the databases are done based on time of execution for the most used operations - CRUD, and for these two methods - Python and JMeter- are used to record time. Apart from individual databases, this project will also evaluate performance in case of a Polyglot Persistence, a hybrid approach, where we make use of best of both databases based on the business needs it's supposed to satisfy for efficiently leveraging both the databases. The goal of this project is to create a prototype to evaluate the performance of the data stores - MongoDB, AWS RDS & Polyglot in case of an optimal Healthcare Insurance plan recommendation system.

2. Introduction

2.1. Objective:

The objective of this project is to explore performance testing tools (Python, JMeter etc.) for the databases in order to compare the performance of RDBMS with MongoDB considering the requirements of Healthcare data application. This comparison will help us identify a superior database with respect to the type of data to be handled and typical use cases of our application. We will be able to recommend the same for Healthcare organizations which are eager to change their DB structure for better performance and to enhance the customer experience for any specific application.

2.2. What is the problem?

We are considering a particular use case in the Healthcare domain which is Healthcare Insurance Recommendation Application. Whenever a person wants to buy a new medical insurance policy from Healthcare Insurance providers, they need to provide some personal details like name, age, address etc. and based on some predefined criteria, the recommendation application provides few best suited medical insurance policies. Such applications gather data from many insurers (Healthcare providers) and many people can access this application to identify insurance plans as per their requirements. Along with accessing large data, this application needs to be updated constantly for which database should have faster accessibility and performance. Most frequently used operations are- read and insert/create as users keep entering their details and read the results provided by the application.

Both NoSQL & RDBMS have their advantages and disadvantages, but for a given application/ domain it is required to identify which is best suited to ensure good performance. The complex technology landscapes with multiple competing products, organizations must balance their cost and speed of technology against the fidelity of the decision. There is rarely a single 'Right' answer in selecting a complex component for an application. But a wrong choice will be costly & will have an adverse effect on productivity. Hence the question arises to choose between these databases as per different application needs. This situation demands comprehensive study of databases and its performance evaluation to answer the technology selection for organizations.

2.3. Why is this project related to this class?

This project aims to compare the performance of a NoSQL database with that of a Traditional Relational Database and identify the best suited database as per the requirements of the Healthcare application we have chosen. For doing the same, we require detailed understanding of types of Healthcare data, their general use cases and database structures, their fundamental differences and strengths to handle such data. This project will help us better understand the concepts learnt in class, about Relational Databases and NoSQL Databases and further demonstrate how it can be applied in the Real-World Applications.

2.4. Why other approach is no good?

Many organizations still use local servers to store data which are expensive when it comes to scaling up. Use of local servers also has slow adaptability and needs regular maintenance. Storage and power needs change as business grows or shrinks. Updates are inevitable and tricky, especially for bigger companies. Large organizations looking to scale up or down often need to go through some amount of research, budget meetings and sign-offs for even a simple RAM upgrade or hardware replacement. This approach is tedious, time consuming and complicated. Also, use of traditional RDBMS has its own disadvantages when it comes to handling large unstructured data which needs to be updated and accessed continuously.

2.5. Why Our Approach is Better?

To come up with a better performing Database for the Healthcare Insurance Application, we chose Cloud environment with RDS (AWS) and MongoDB for our experiment. Following are some highlights:

- Cloud is better for large data handling with faster accessibility
- Cloud is inexpensive for scaling up
- Cloud allows data to be accessed from anywhere with internet connection
- Maintenance and upgrades are handled by Cloud providers
- Polyglot persistence - Hybrid approach with cloud RDS and MongoDB enables to use the best part of both as per application needs.

Our aim is to verify and compare the performances of both with respect to the Healthcare Insurance Application which is a recommendation system. The best choice depends largely on a company's needs. Healthcare organizations need not worry about data Maintenance & upgrades as it is handled by cloud providers. Our solution includes identifying better

performing DB by comparison analysis and also the use of Polyglot persistence, i.e. a hybrid solution using cloud RDS and MongoDB Atlas, that enables us to use the best part of both as per application needs.

2.6. Statement of the problem:

Selection of an ideal database for a given application is a challenge. In the Healthcare domain, especially with respect to Healthcare Insurance, it is important to choose a database system that balances the cost and speed of processing large unstructured data. A patient requires an ideal Healthcare Insurance Plan for optimal coverage cost. With this use case, we propose a project to analyze and compare the performance of RDBMS, MongoDB in Cloud environments and a hybrid setup (MongoDB + RDS) to identify the superior database system for handling such Healthcare Applications effectively.

2.7. Area or Scope of Investigation:

1. Generate pseudo Healthcare Data
    - Generating Healthcare data similar to real-world applications using python code.
    - Creating tables according to application needs.
2. Understand the structural differences of RDBMS and MongoDB databases in Cloud environments.
    - SQL as a relational database system and MongoDB as a non-relational (NoSQL) database system vary in many aspects- structure, language, properties, etc. It is important to gain knowledge of both database systems for better understanding.

3. Implementation of Polyglot Persistence, a Hybrid approach where RDBMS and MongoDB both are used for suitable parts of the Application each.

4. Use of suitable Performance comparison tools such as JMeter, python in-built modules.
   - Perform CRUD operations based on Healthcare Insurance Recommendation Application use cases and record the performance with respect to time.

5. Visualize our results using Visualization tools like Tableau or Excel.

6. Analyze and identify the superior database with respect to performance.

3. Theoretical Bases and Literature Review

3.1. Definition of the Problem - Mathematical Formulation:

As per our Healthcare application use case, the weightage to various chosen attributes is considered.

For example, take CRUD operations, if the application is performing only Retrieve & Update, with more Retrieval and less Update operations, we can set appropriate weights for these 2 attributes accordingly.

Databases:	A) MongoDB	B) RDS
Attributes:	1) Create (30%)	2) Read (40%)
	3) Update (20%)	4) Delete (10%)

i.e., 0.4 for Read and 0.2 for Update.

We define **Performance core** as,

Sum of all [(Weightage assigned to attribute x)*(execution time for attribute x)]

For **Database A** (MongoDB), the executional performance is calculated as

= [0.4*(exec time for Reads in db**A**)] + [0.2*(exec time for Updates in db**A**)]

= 0.4*(550ms) + 0.2*(700ms) = 220 + 140 = **360ms**

For **Database B** (CouchDB), the executional performance is calculated as

= [0.4*(exec time for Reads in db**B**)] + [0.2*(exec time for Updates in db**B**)]

= 0.4*(750ms) + 0.2*(1000ms) = 300 + 200 = **500ms**

By Comparing the **Performance Scores** of databases, we can consider the dataset with the least execution time. This can be scaled up with the increase in attributes, different databases as well as various data sizes.

3.2. Theoretical Background of the Problem:

Our problem of choosing the best suited and best performing database is demonstrated for a Healthcare Insurance Recommendation Application. Healthcare insurance companies keep providing data continuously from various sources and it is collected by Recommendation Application. Handling a variety of sources, large data which constantly needs to be updated and accessed is main challenge. Based on person's medical history, age group, tobacco preference and number of dependents, application filters out some values and returns few medical insurance plans as recommendations for that person. Best suited plan is then

displayed along with the premium that needs to be paid and copay percentages with estimated deductibles.

- Over the period of time, Healthcare systems have improved in performance- from using transaction processing systems to RDBMS. The usage of RDBMS was beneficial but with the onset of NoSQL, it was required to transition to the new technology to encompass the large amount of data that needed to be handled. In an industry like healthcare, there is a requirement for interoperability and easy access of health records by the authorized personnel. MongoDB provides these features in addition to security of the data.
- In this project, healthcare data is to be used in a structured (RDBMS) and unstructured (NoSQL) format. As mentioned, many applications switched from RDBMS to NoSQL and picked it as an information stockpiling framework. However, in order to leverage the pros of RDBMS, we need to have easier conversion mechanisms between relational databases and NoSQL. The main challenge of this project is data conversions before loading it to the respective databases. The document format in MongoDB and its respective tabular structure in relational database is shown in figure below.

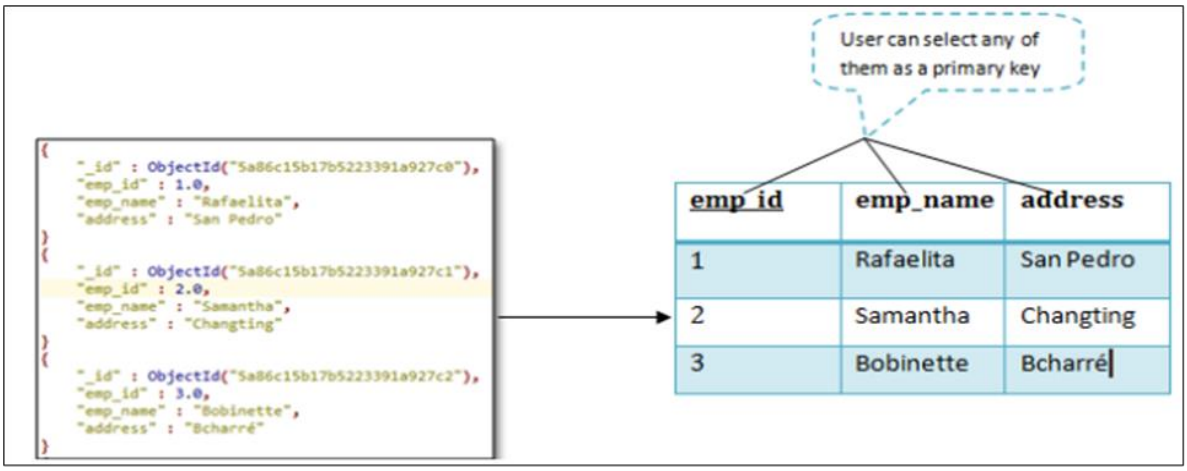


Fig 1: The above example of Document NoSQL to relational conversion can also be implemented vice versa

- Relational databases use a method of normalization to get the data in the standard structure of tables. Normalization up to 3NF or more is used resulting in more tables. Whereas inMongoDB we can have entire data into one single collection of documents, or we can have a separate collection of documents for each table in SQL. The complex queries require retrieval or accessing data from multiple tables. This is typically done by join query in SQL and manual query in MongoDB within collection of documents.
- Another aspect we are exploring in this project is the [benchmarking tool. These play an important role in the selection of a database for a given application. Benchmark tools are a reproducible framework for comparing the performance in terms of time, memory and quality of any database or operations specific to the database.

3.3. Related Research to solve the Problem:

In this section we will refer to some of the related works.

In [1] with the use of JDBDT framework, authors compared performance of CRUD operations as well as Aggregate functions with complex queries. By creating separate collections for each table in RDBMS, the performance was recorded in which MongoDB performed far better than RDBMS except for Aggregate functions.

In [2] the authors performed testing in C# console with insert operations in four different levels for three tables with three different types of updates, simple and complex queries. MongoDB performed better than SQL but in case of indexed and non-indexed updates results were reversed.

[3] talks about Migration of healthcare relational database to NoSQL cloud database for healthcare analytics and Management

[4] suggests how to integrate MongoDB and MySQL by a middleware layer between application and DB layers.

[5] The current epoch is perceived as the Big Data era due to the fact that massive, complex and growing data that derives from different sources is generated at an exponential rate

In [6] we consider NoSQL to RDBMS conversion as a research challenge, and we proceed further to develop a generic framework for NoSQL to RDBMS data conversion

[7] Investigate the execution of MongoDB (a document-oriented database) versus MySQL database in view of insertion and retrieval tasks utilizing a web/android application to investigate load balancing

In [8], they have compared executive time of 10 NoSQLs in stand-alone mode and analyzed the respective operation performance.

Five NoSQL databases (Redis, MongoDB, Couchbase, Cassandra, HBase), each on a 4-node cluster are compared in terms of their performance, using Yahoo! Cloud Service Benchmarking- comparison of loading time, execution time & throughput [9].

In [10], relational databases and non-relational databases were evaluated for performance based on CRUD operations, but the comparison was performed with simple structured data and only one schema of MongoDB was examined which wasn't beneficial for MongoDB.

In healthcare database systems, MongoDB is wildly popular and is one of the fastest ways to build FHIR (Fast Healthcare Interoperability Resources) applications. Interoperability in Healthcare enables seamless and secured health information exchange across all hospitals & patients [11].

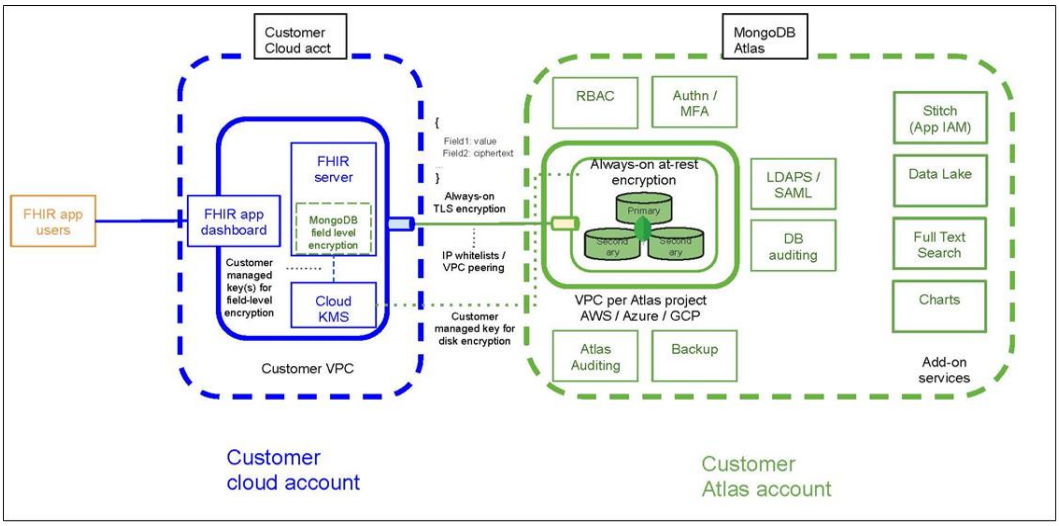


Fig 2: FHIR Application Architecture with MongoDB

3.4. Advantage/ Disadvantage of Previous Research (final)

The following are the advantages and disadvantages that we found from the abovementioned related work:

Advantages:

- The advent of NoSQL was a remarkable contribution for the healthcare system as data was made easily available and was also interoperable.
- Simulation of a real-world scenario of accessing databases from a single application by using python drivers and accessing applications.
- RDBMS proved to be performing better in case of indexed, non-indexed queries as well as Aggregate function than NoSQL databases.

Disadvantage:

- Most papers are only using CRUD operations for execution time comparison.
- Experimented on a fixed set of 3 document NoSQL databases rather than taking an exhaustive approach and analyzing other NoSQL types

- While using MongoDB, it is required to configure it with respect to the application/ data for which it is used or rather the data needs to be converted.
- Polyglot persistent infrastructures are not considered in most papers whereas it is needed in most real-world applications.

3.5. Our Solution to solve this problem

In this project, we plan to use MongoDB Atlas and RDS AWS as our databases for performance comparison. The challenge we face here is in using healthcare data, which is extremely sensitive.

Hence, we would need to create synthetic data, for which we aim to use python. Further, we aim to evaluate performance via comparison between the databases using Python code (timestamp) and Jmeter (Java-Groovy), to find if there is any difference in performance by using two different tools.

We want to create a recommendation system based on some predefined variables such as age-group, tobacco usage, dependents count, etc.

3.6. Where our solution different from others

In the research papers, it appears that Cassandra is mostly used for healthcare systems, followed by use of MongoDB and mostly these two are compared. In this project we will be focusing only on MongoDB in comparison with RDBMS. In case of performance comparison, the use of JDBDT tools or inbuilt performance monitors are common, thus we aim to use Jmeter and python.

3.7. Why our solution is better?

The use of cloud-based databases increases availability of data, extremely affordable and ease of use. MongoDB being a document store supports storage in the form of flexible documents and allows data to be stored in multiple formats- JSON, XML, etc. In MongoDB, similar data is stored together so that data access is quick and easy. MongoDB stores most of its data in RAM thus allowing fast performance while querying. It supports multiple languages and thus can be used by a range of applications. In the healthcare domain, use of MongoDB makes record management an easy task.

As the comparison tool, JMeter is an optimal and precise tool. The workloads are easy to define and the evaluation recording processes are automated, thus making it a very easy tool to use.

4. Hypothesis / Goals

1. RDBMS performs better when working with a small subset of data as compared to MongoDB which performs better with large sized data.
2. Leveraging MongoDB for data retrieval and RDBMS for a small subset of operations (through polyglot persistence/ hybrid approach) is very efficient.

5. Methodology

5.1. How to generate/collect input data

Taking inspiration from public healthcare insurance datasets, we created synthetic data with python code.

```
import pandas as pd
import numpy as np
from datetime import datetime, date
import random
np.random.seed(4)
val1 = np.random.choice(['Male', 'Female'],nrows)
val2= pd.to_datetime(np.random.randint(pd.Timestamp(start).value,
    pd.Timestamp(end).value, nrows, dtype=np.int64)).strftime('%Y/%m/%d')
```

Fig 3: Sample code for creation of 'Patients' table

The data consists of two tables – 'Insurance' & 'Patients'

The schema for each table looks as follows:

Insurance:

plan_id	plan_name	state_code	issuer_id	issuer_name	business_year	federal_itin	age_group	tobacco_use	dependents_status	dependents_count	Premium	copay	yearly_deductible($)	out_of_pocket_max($)
OL83AM65395	GOJPQBA VGGZM	AL	39714	ZATEZAWRWZ	2020	12-064787	0-20	Yes	Yes	1	2150	18%	3900	4869
VQ30AM10973	VMQDQWN TRBTP	AL	58550	GISHGFWWUF	2020	16-093230	0-20	Yes	Yes	1	2000	18%	2800	5225
GY56BW80061	HAWMSWS XKULY	AL	59200	YNGLKCOJOY	2020	14-072796	0-20	Yes	Yes	1	2050	18%	4300	5386
JK43LR87749	WCDFZLQ HNKJP	AL	26558	QKWZMYADEY	2020	13-072212	0-20	Yes	Yes	1	2100	18%	3100	5827
JL40ZP6801	PXRYWZR EWZXO	AL	35312	RSHMXCDBMV	2020	18-034062	0-20	Yes	Yes	1	2150	18%	4500	5220

Fig 4: Sample schema of 'Insurance' table

Patient:

PatientID	First_Name	Last_Name	Gender	DOB	Age	Street	City	State	Zip	Home_Contact	Work_Contact	EmailID	Date_of_visit	No_of_Dependents	Tobacco_preference
JY-12010	Brian	Park	Male	9/2/1993	28	672 Carl Locks	Laurenton	NC	70844	574-628-6117	343-607-8917	zscpyib@xyz.com	3/22/2021	0	No
FW-93422	Matthew	Bentley	Male	2/2/1957	65	43044 Fred Highway	Christophermouth	MS	66817	373-142-3050	233-379-9895	aevspyx@xyz.com	7/3/2021	1	No
MW-43512	Denise	Scott	Female	1/23/1955	67	4184 Regina Ports	Melaniehaven	MT	56040	987-693-0106	585-641-9517	lkyaipz@xyz.com	7/13/2020	1	Yes
AO-22602	Morgan	Barrett	Female	2/21/2009	13	2344 Christopher Mountain Suite 887	Dustinborough	MT	27065	446-515-7598	167-621-0216	gxnrrvd@xyz.com	6/21/2020	2	No
PD-35838	Elizabeth	Sutton	Female	6/6/1987	34	2569 Walters Flat Suite 103	Kathybury	MA	23771	719-083-5473	957-481-4250	gsrwzxi@xyz.com	10/31/2020	0	Yes

Fig 5: Sample schema of 'Patients' table

Record sizes for the Insurance table increased from 100k to 300k and for Patients table 50k to 150k for each experiment (CRUD) conducted. Starting from the minimum number of records, we kept increasing record sizes by adding new records to the same table/collection and repeating experiments (CRUD).

5.2. How to solve the problem

Step 1: Synthetic data Creation

Step 2: Choose the optimum cloud provider for the required databases (MongoDB & MySQL)

Step 3: Load data in parts into the shortlisted databases

Step 4: Perform Create, Read, Update, Delete operations in each database for each data size.

Step 5: Record time after each increased dataset.

Step 6: Compare performances of each database and as a combined data source (polyglot).

Step 7: Visualize the results obtained by repeating the above experiments.

Step 8: Choose the database with the least execution time for each operation.

5.2.1. Algorithm Design

- Data Creation

  - Create synthetic data for per patient records and healthcare plans & details.

- Evaluate performance of each database system for each operation/ workload using benchmark tools [Python/ Jmeter).
- Record, compare and visualize the performance evaluation between databases.

The below figure is a representation of the detailed steps of execution for solving the comparison problem.

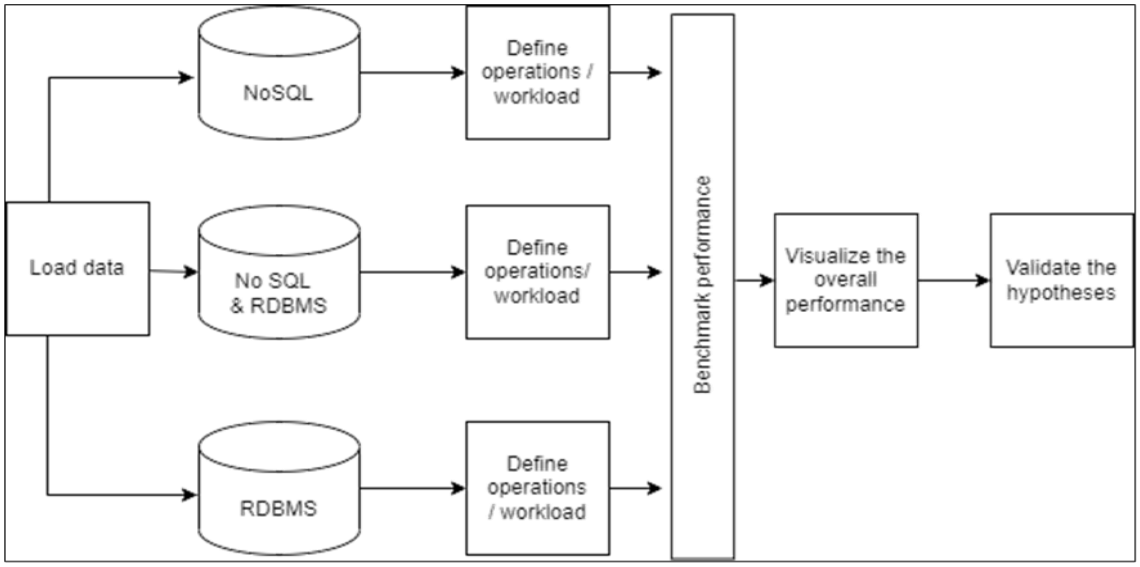


Fig 6: Algorithm design for Performance evaluation

5.2.2. Language Used

- MQL (Mongo Query Language)
- SQL (Structured Query Language)
- Python
- Java/ Groovy

5.2.3. Tools Used

- PyCharm/ Jupyter Notebook
- Jmeter
- MySQL
- Excel (Visualization of query execution)

5.3. How to generate Output

- Python and Jmeter are used to connect to Databases and load testing is performed from respective tools.
- Output of the Performance tool is analyzed
- The data is plotted in the form of graph through visualization tools like excel/ others and query performance is observed.

5.4. How to test against Hypothesis

- **Hypothesis 1:** Using a NoSQL and Polyglot for any application is always better than a traditional RDBMS for CRUD operations.

Test 1: Compare the time taken by create, read, update and delete operations with the 2 types of databases.

- **Hypothesis 2:** Using a benchmarking tool makes the database comparison process seamless compared to leveraging traditional application-based comparison

  Test 2: Compare the executional implications of comparison process across Python based comparison method vs JMeter.

6. Implementation

6.1. Code

We generated the synthetic data using python (Jupyter Notebook) for Healthcare Insurance Plans and Patient Records. We split the data into three parts and upload one set of records at a time through python. On each set of data loaded, Read, Update, Insert and Delete operations are performed from python and JMeter (Groovy Language) and the time taken for execution is recorded for each. The source code is given in the appendix.

6.2. Design Document and Flowchart

We have prepared a design flowchart, which explains the steps of the database performance evaluation.

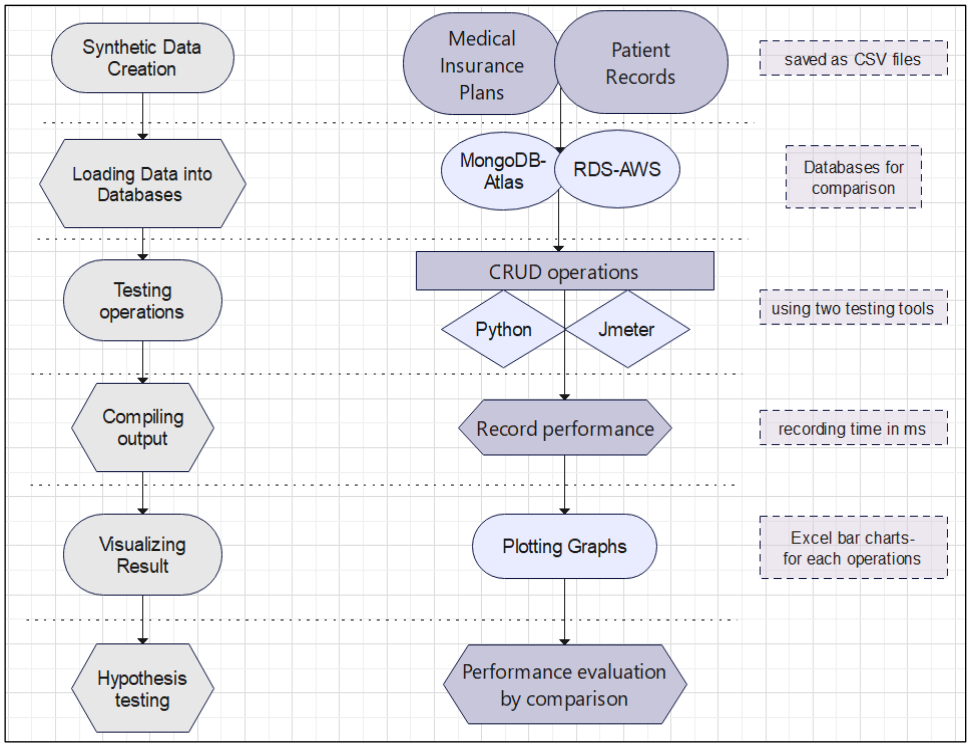


Fig 7: Design Flowchart

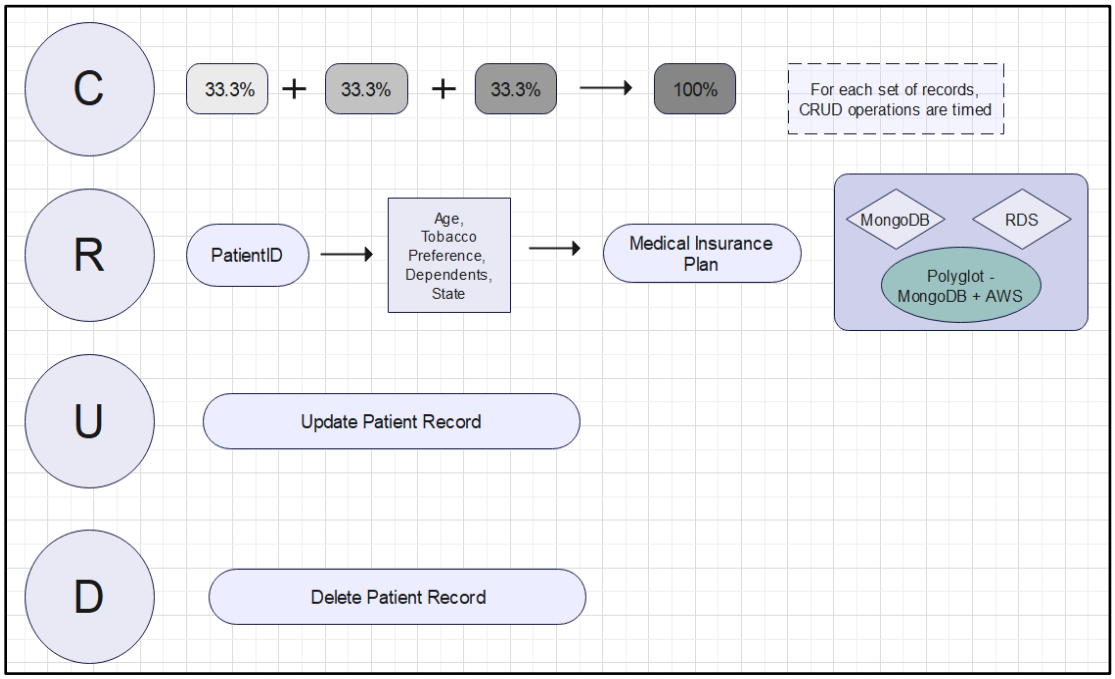


Fig 8: CRUD Operations

7. Data Analysis and Discussion

7.1. Output Generation

Performance analysis for the CRUD operations have been implemented systematically through various workloads. This is done so as to understand the stress management of the RDS and MongoDB clusters for various sizes of input data. The synthetic data of insurance that had 306k records and the patients data that had 150k records was systematically pushed to the servers hosted in the cloud. The workloads are enlisted below,

Workload 1: Insurance - ~100k records, patients - ~50k records

Workload 2: Insurance - ~200k records, patients - ~100k records

(This is post adding 100k records of the insurance data and 50k records of the patients data)

Workload 3: Insurance - ~300k records, patients - ~100k records

(This is post adding 100k records of the insurance data and 50k records of the patients data)

Each experiment has been conducted 20 times to improve the accuracy of the results.

A constant data load has been chosen to understand the stress handling ability of the database as the data size increases. Below is the output from the experiments conducted.

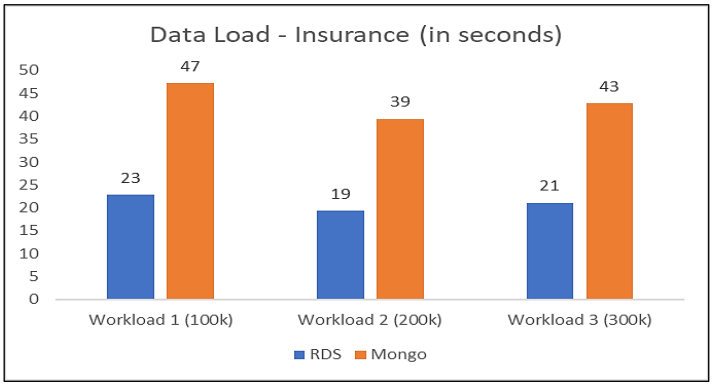
Also, the analysis has been done in Python as well as the Jmeter benchmarking tool

7.2. Output Analysis

Using the procedure mentioned in the previous section, the experiments were conducted and graphs and visuals were plotted. Below are the visuals from each of the experiment.

Data Load using Python:

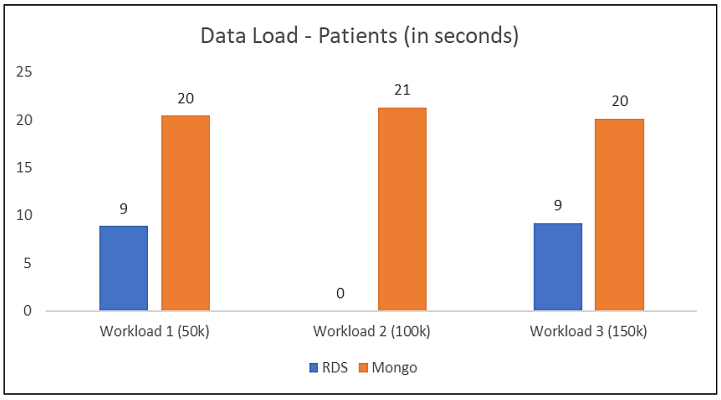
Insurance Data:



Workloads	RDS	Mongo
Workload 1	22.79	47.17
Workload 2	19.30	39.39
Workload 3	21.09	42.81

Fig 9: Data Load (Insurance) - Python

Patients Data:

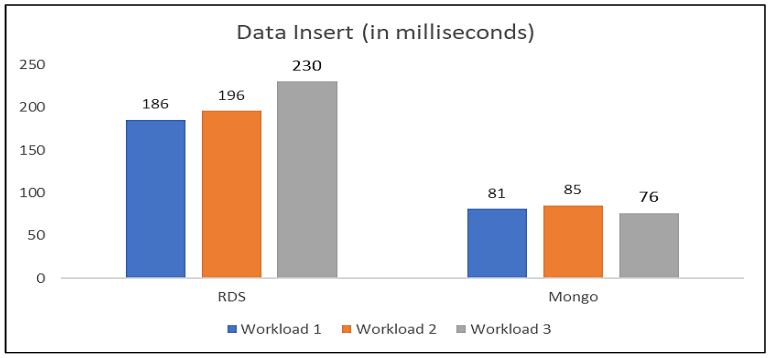


Workloads	RDS	Mongo
Workload 1 (50k)	8.87	20.45
Workload 2 (100k)	0.06	21.32
Workload 3 (150k)	9.20	20.13

Fig 10: Data Load (Patients) - Python

Insert Data:

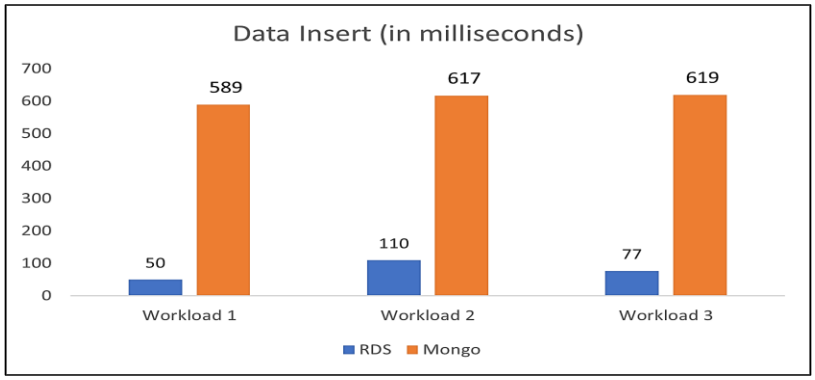
Python



Workloads	RDS	Mongo
Workload 1	185,878	81,425
Workload 2	185,878	81,425
Workload 3	230,241	76,130

Fig 11: Insert Data- Python

JMeter

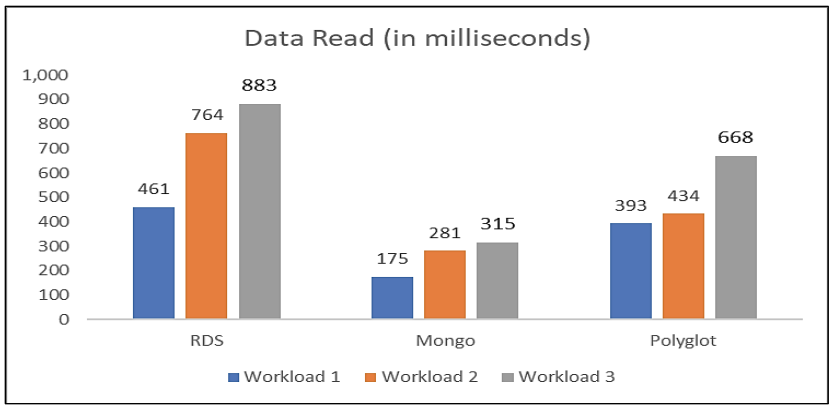


Workloads	RDS	Mongo
Workload 1	50,000	589,000
Workload 2	110,000	617,000
Workload 3	77,000	619,000

Fig 12: Insert Data- JMeter

Read Data:

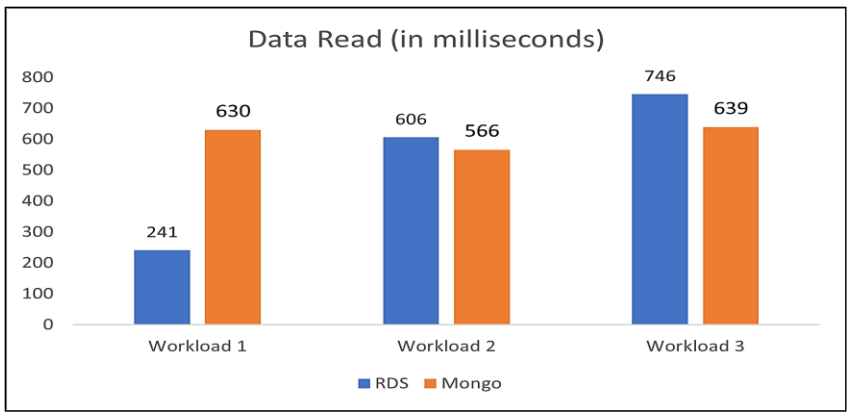
Python



Workloads	RDS	Mongo	Polyglot
Workload 1	460,714	174,977	392,532
Workload 2	763,772	281,127	433,510
Workload 3	882,682	314,799	668,054

Fig 13: Read Data- Python

JMeter

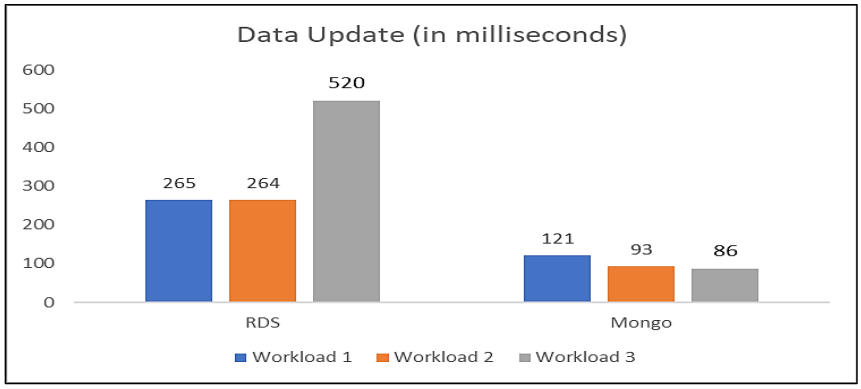


Workloads	RDS	Mongo
Workload 1	241,000	630000
Workload 2	606,000	566000
Workload 3	746,000	639000

Fig 14: Read Data- Jmeter

Update Data:

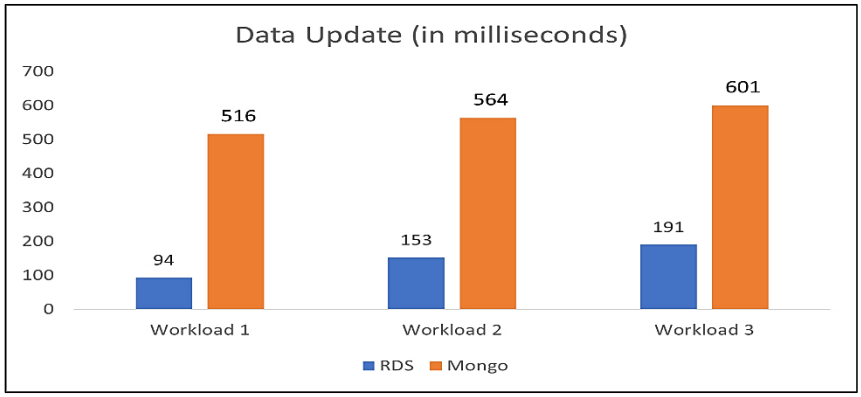
Python



Workloads	RDS	Mongo
Workload 1	264,780	121,353
Workload 2	264,311	93,431
Workload 3	520,219	86,422

Fig 15: Update Data- Python

JMeter

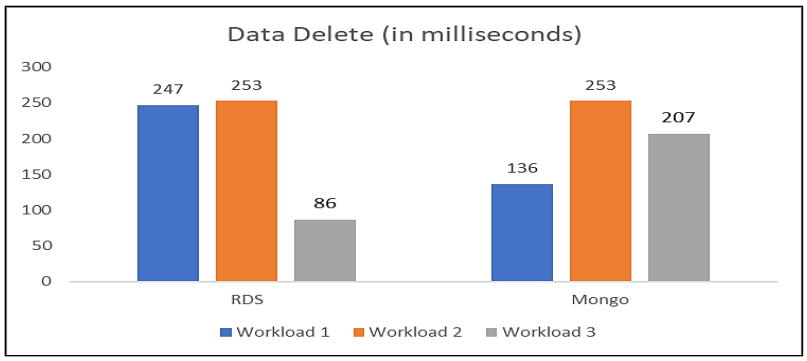


Workloads	Jmeter
Workload 1	94,000
Workload 2	153,000
Workload 3	191,000

Fig 16: Update Data- JMeter

Delete Data:

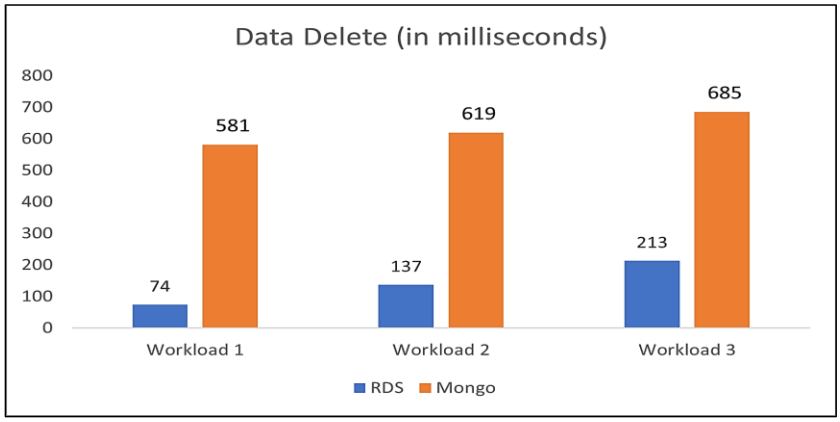
Python



Workloads	RDS	Mongo
Workload 1	247,059	136,406
Workload 2	253,278	253,278
Workload 3	86,422	207,023

Fig 17: Delete Data- Python

JMeter



Workloads	Jmeter
Workload 1	74000
Workload 2	137000
Workload 3	213000

Fig 18: Delete Data- JMeter

7.3. Comparison of Output against Hypothesis

Compare the time taken by create, read, update and delete operations with the 2 types of databases: It is observed that MongoDB and Polyglot perform better for read, update, insert and delete operations whereas they do not perform as good for data load. This might be attributed to the minimal size of the data and because of thet we cannot get any concluding results

- Compare the executional implications of comparison process across Python based comparison method vs JMeter: User friendly, automates results, own visualization, multiple metrics, reduced turnaround time. In the case of using RDBMS bases database seemed even more friendly

7.4. Discussion

The experiments conducted above prove that, overall, the performance of MongoDB is better than that of RDS, except in case of Load operation. The Polyglot approach worked the best for read operation in comparison to the individual databases. In case of insurance plan recommendation, where read operation is of utmost importance, Polyglot would be the best choice.

8. Conclusions and Recommendations

8.1. Summary and Conclusions

The observations performed on the patients and insurance data gives a detailed performance comparison of SQL and MongoDB considering a real time scenario with a read heavy situation. The Polyglot model is suitable and suggested for this kind of data model which leverages the advantage of each of the databases. Here, we used insurance data hosted in MongoDB because of its big size and least consistency requirements, however, the patients data hosted in AWS RDS works better for the patients data because of concurrency and its consistency required.

8.2. Recommendations for Future Studies

Business Improvisations:

- Increase the number of insurance plan selection attributes for the patient to make the model robust

Data Improvisations for Future:

- Increase data size and number of experimentations for accurate results and insights
- Explore JMeter further to handle data load and multi record operations
- Experiment on availability, consistency and concurrency to check how the system responds and observe specific characteristics
- Stress testing with JMeter
- Finding best method to upload data to RDS (AWS S3 with LAMBDA fun)
- Experiment with number of users to stress test read heavy situations

9. Bibliography

[1] Seyyed Hamid Aboutorabia, Mehdi Rezapourb, Milad Moradic, Nasser Ghadirid, "Performance evaluation of SQL and MongoDB databases for big e-commerce data" IEEE, 2015.

[2] Z. Parker, S. Poe, and S.V. Vrbsky, "Comparing NoSQL MongoDB to SQL db", ACM, 2013.

[3] Dimpal Tomar, Jai Prakash Bhati, Pradeep Tomar, Gurjit Kaur, "Healthcare Data Analytics and Management"- chapter 2, 2019.

[4] S. Khan, and V. Mane, "SQL support over MongoDB using metadata" International Journal of Scientific and Research Publications 3, no. 10 (2013).

[5] TANG, Ming; LIAO, Huchang, "From conventional group decision making to large-scale group decision making: What are the challenges and how to meet them in the big data era? A state-of-the-art survey", Omega, 2021, 100: 102141

[6] Biswajit Maity, Anal Acharya, Takaaki Goto, Soumya Sen, "A Framework to Convert NoSQL to Relational Model", ACM, June 2018, 1-6

[7] M. M. Patil, A. Hanni, C. H. Tejeshwar and P. Patil, "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", International Conference on I-SMAC, 2017, 325-330

[8] Abramova, Veronika, Jorge Bernardino and Pedro Furtado, "Experimental evaluation of NoSQL databases.", International Journal of Database Management Systems 6.3 (2014): 1

[9] Enqing Tang, Yushun Fan, "Performance Comparison between Five NoSQL Databases", IEEE 7th International Conference on Cloud Computing and Big Data (2016): 105-109

[10] Z. Bicevska and I. Oditis, ``Towards NoSQL-based data warehouse solutions," Procedia Comput. Sci., vol. 104 (2017): 104 – 111

[11] FHIR Applications with MongoDB - https://www.mongodb.com/blog/post/building-fhir -applications-with- mongodb-atlas? tck=healthcarepage

[12] Weider D. Yu Manjula Kollipara Roopa Penmetsa Sumalatha Elliadka, “A Distributed storage solution for cloud-based e-Healthcare Information System” IEEE, 2013.

10.Appendices

10.1. Program Source Code with Documentation

Our source code is available on the Github. The link is provided below:

https://github.com/Revathi-Praveen/DB225_TermProject_Group1

10.2. Input/output Listing

All the input/output files can be found at github link above.

Input files:

Data creation - Patient_Insurance-data_creation.ipynb

Output files:

CRUD operations -

Create-MongoDB_SQL.ipynb, Insert Mongo_SQL.ipynb, Read-MongoDB_SQL_RB.ipynb,

Update-MongoDB_SQL_RB.ipynb,Delete - MongoDB_SQL.ipynb

10.3. Other Related Material

Apache JMeter: https://jmeter.apache.org/usermanual/index.html