Databases For Mediation Systems
Design and Data scaling approach

Nitin Reddy Ayyagari

Faculty of Computing
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden
This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Masters in Electrical Engineering with emphasis on Telecommunication Systems. The thesis is equivalent to 20 weeks of full-time studies.

Contact Information:

Author(s):
Nitin Reddy Ayyagari
E-mail: niay14@student.bth.se

External advisor:
Neeraj Garg
Senior Manager,
Ericsson Global India(R&D)Ltd,
Gurgaon, India

University advisor:
Prof. Lars Lundberg
Faculty of Computing
BTH, Sweden

Faculty of Computing
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57
Abstract

Context: There is continuous growth in data generation due to wide usage of modern communication systems. Systems have to be designed which can handle the processing of these data volumes efficiently. Mediation systems are meant to serve this purpose. Databases form an integral part of the mediation systems. Suitability of the databases for such systems is the principle theme of this work.

Objectives: The objective of this thesis is to identify the key requirements for databases that can be used as part of Mediation systems, gain a thorough understanding of various features, the data models commonly used in databases and to benchmark their performance.

Methods: Previous work that has been carried out on various databases is studied as a part of literature review. Test bed is set up as a part of experiment and performance metrics such as throughput and total time taken were measured through a Java based client. Thorough analysis has been carried out by varying various parameters like data volumes, number of threads in the client etc.

Results: Cassandra has a very good write performance for event and batch operations. Cassandra has a slightly better read performance when compared to MySQL Cluster but this differentiation withers out in case of fewer number of threads in the client.

Conclusions: On evaluation of MySQL Cluster and Cassandra we conclude that they have several features that are suitable for mediation systems. On the other hand, Cassandra does not guarantee ACID transactions while MySQL Cluster has good support. There is need for further evaluation on new generation databases which are not mature enough as of now.

Keywords: Cassandra, Database, Mediation Systems, MySQL, Performance
I would like to take this opportunity to thank Almighty God for his wonderful ways and vision. I would not have garnered strength in many difficult situations without his wisdom and guidance.

I express my deepest gratitude to my supervisor Prof. Lars Lundberg for his academic guidance and time. His ideas and suggestions were very much thought-provoking and showed me the right direction whenever in doubt. I also extend my thanks to our examiner, Dr. Kurt Tutschku for his efforts throughout the course.

I would also like to thank my thesis partner K.N.S.R Sankaran for his assistance during the course of the project. My sincere thanks to Neeraj Garg and the team at Ericsson R&D, Gurgaon for providing us an opportunity and necessary platform and tools for this work.

I am indebted to my Mother for her unconditional love and support without which I would not have been able to achieve anything. I wish to thank Jayakar Uncle and family for encouraging and supporting me at all times.

Finally I thank all my friends and well-wishers at BTH, Karlskrona and JNTU, Hyderabad who never made me feel short of joy and fun.
Dedicated to my father Late A.B Reddy for his values and upbringing that inspires me everyday
Contents

Abstract i
Acknowledgements ii
List of Figures vi
Abbreviations vii

1 Introduction 1
  1.1 Problem Statement . . . . . . . . . . . . . . . . . . . . . . . . . 3
  1.2 Research Questions . . . . . . . . . . . . . . . . . . . . . . . . 3
  1.3 Contribution . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
  1.4 Outline of the Thesis . . . . . . . . . . . . . . . . . . . . . . . 4

2 Background 5
  2.1 SQL Databases . . . . . . . . . . . . . . . . . . . . . . . . . . 5
  2.2 NoSQL Databases . . . . . . . . . . . . . . . . . . . . . . . . . 6
  2.3 CAP Theorem . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
  2.4 ACID Compliance . . . . . . . . . . . . . . . . . . . . . . . . . 9
  2.5 MySQL Cluster . . . . . . . . . . . . . . . . . . . . . . . . . . 10
  2.6 Apache Cassandra . . . . . . . . . . . . . . . . . . . . . . . . 11
  2.7 Requirements of Mediation Systems . . . . . . . . . . . . . . . 13
List of Figures

1.1 Mediation system functional architecture . . . . . . . . . . . . . . 2
2.1 Database Classification according to CAP . . . . . . . . . . . . . . 8
2.2 Architecture of MySQL Cluster . . . . . . . . . . . . . . . . . . . 11
2.3 Architecture of Cassandra . . . . . . . . . . . . . . . . . . . . . . 12
3.1 Setup for Experiment 1 . . . . . . . . . . . . . . . . . . . . . . . . 17
3.2 Setup for Experiment 2 . . . . . . . . . . . . . . . . . . . . . . . . 18
3.3 Database Schema . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
4.1 Insert Throughput (Operations/sec), Batch Data . . . . . . . . . 22
4.2 Select Throughput (Operations/sec), Batch Data . . . . . . . . . 22
4.3 Total Time taken for Insert queries, Event Data . . . . . . . . . . 23
4.4 Average Response Time for Insert queries, Event Data . . . . . . 24
4.5 Total time taken for Insert queries, Batch Data . . . . . . . . . . 25
4.6 Total time taken for Select queries (Incremental) . . . . . . . . . 25
4.7 Average Response time for Select queries (Incremental) . . . . . 26
4.8 Total time taken for Select queries (Proportional) . . . . . . . . . 27
4.9 Average Response time for Select queries (Proportional) . . . . . 27
### Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACID</td>
<td>Atomicity, Consistency, Isolation, Durability</td>
</tr>
<tr>
<td>API</td>
<td>Application Program Interface</td>
</tr>
<tr>
<td>BASE</td>
<td>Basically Available, Soft State, Eventually Consistent</td>
</tr>
<tr>
<td>CAP</td>
<td>Consistency, Availability, Partition Tolerance</td>
</tr>
<tr>
<td>CQL</td>
<td>Cassandra Query Language</td>
</tr>
<tr>
<td>GB</td>
<td>Giga Byte</td>
</tr>
<tr>
<td>GGSN</td>
<td>Gateway GPRS Support Node</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of Things</td>
</tr>
<tr>
<td>I/O</td>
<td>Input/Output</td>
</tr>
<tr>
<td>JDBC</td>
<td>Java Database Connectivity</td>
</tr>
<tr>
<td>LTE</td>
<td>Long Term Evolution</td>
</tr>
<tr>
<td>NoSQL</td>
<td>Not only SQL</td>
</tr>
<tr>
<td>PK</td>
<td>Primary Key</td>
</tr>
<tr>
<td>RDBMS</td>
<td>Relational Database Management System</td>
</tr>
<tr>
<td>SGSN</td>
<td>Serving GPRS Support Node</td>
</tr>
<tr>
<td>TB</td>
<td>Tera Byte</td>
</tr>
</tbody>
</table>
Introduction of 4th Generation mobile services like LTE (Long Term Evolution) and the increased usage of smart devices has changed the scenario of communication systems. This has resulted in the emergence of data based services that are gaining prominence when compared to voice based services which were in demand previously. Further, these have also evolved through the consumption of data in the form of Over the Top platforms like Skype, Chat based services etc. Coupled with rapid broadband explosion, the model IoT (Internet of Things) has also evolved. Ericsson estimates that there would be around 50 Billion devices connected by 2020 [1]. Due to wider adoption methods of networking technologies in healthcare, retail, education and plethora of services, we are currently facing a stage of data explosion.

This trend has led to a huge upsurge in data volumes and provide a formidable task to deal with. Management of data is big issue, though using this raw data to characterize the services, understanding the consumer behavior and extracting commercial revenues which are sustainable in the long term from them are another potential use cases of it. This is where Big Data and data analytics have come into play. Effective usage of these platforms can enhance the way services are provisioned and implemented for a number of applications. Telecommunication networks are a very good choice given their nature of wider penetration and easy accessibility.

Industry partners are continuously looking to leverage this data and are in the process of developing systems which can handle voluminous data efficiently. There are certain challenges in this task that could range from securing the data at the points of generation, providing uniform interfaces to enable data collection across multi-vendor devices etc. There should be a converged platform as an interface between the data nodes and the application based services. Mediation
Mediation systems are those which collect the information from various nodes in the network. These could range from network elements like GGSN (Gateway GPRS Support Node), SGSN (Serving GPRS Support Node), network devices like routers and switches, sensors etc. The data could be in different formats and is processed across a formatter and routed according to the source credentials. It is then distributed for main downstream services like Billing, Fraud detection and for Big Data analysis [2]. Network monitoring and Service assurance which are key part of services from operator point of view can also be handled here. Hence, this acts as a common source of data for monitoring and commercial aspects.

Data involved in mediation can be categorized into two types, event and batch operations. Batch operations are employed in large scale processing while event operations are used in real time processing. In batch mode, multiple number of queries are submitted and processed in a single statement while in the case of event each query is processed sequentially. This could result in faster processing for batch mode which are used in offline data processing while event based processing is used in online or real-time scenario. Call data and usage records are generated whenever a user starts and concludes a session involving data or voice based services. These are consolidated from time to time and then distributed for down-stream services like real time charging and analytics. Processing them by maintaining integrity and avoiding loss of data are crucial in generating revenues for Telecommunication industry players.
Chapter 1. Introduction

1.1 Problem Statement

Databases form the backend part of the mediation systems. There are certain standard concerns for the telecommunication industry such as high availability and ability to handle large data. As high volumes of data are generated, they require rapidly deploying large clusters to accommodate large data. Also processing of this data requires frequent reads and writes to the disk, databases can be very crucial part in reducing processing times, efficient utilization of data and cost reduction [3]. Ericsson previously used relational databases like PostgreSQL etc.

In order to deal with large data and a way to reduce the processing times, they are looking for new generation databases. Currently, there are plethora of databases under different categories like SQL, NoSQL etc. The main concern of the thesis work is to identify the databases which are suitable for the mediation systems, evaluate their performance considering different types of processing mechanisms like batch and event processing that are commonly employed in mediation systems.

1.2 Research Questions

1. What is the throughput (operations/sec) behaviour of the batch data for the databases considered?

2. What is the effect on the performance of the databases considered in terms of total time taken when data volumes are scaled?

1.3 Contribution

The thesis focusses on analyzing the requirements of the mediation systems. Different types of databases, various data models that are used in them are briefly discussed. Requirements of mediation systems are listed. Performance evaluation is done for event and batch operations based on throughput and total time taken for MySQL Cluster and Cassandra, two of the widely used open source databases. Also, ACID guarantees and its implications are briefed. Conclusions are drawn based on the results obtained and scope for future work is also described.
Chapter 1. Introduction

1.4 Outline of the Thesis

Chapter 1 deals about the problems of data explosion, function of mediation systems, problem identified as a part of this work. It also contains the problem statement, research questions and contribution of this work.

Chapter 2 gives an account of the databases. Various concepts related to databases such as CAP theorem are dealt here. It also gives an account of requirements, a brief description of database solutions available and a detailed account of the ones used for evaluation.

Chapter 3 gives an overview of the conditions under which the experiment is conducted. Experimental set up, Configuration details and parameters involved are also described.

Chapter 4 presents the results of the experiment. Brief observations regarding different scenarios of the experiment are presented here.

Chapter 5 contains discussion on related works. Analysis of reliability and threats to validity are also described in this section.

Chapter 6 contains the conclusions derived based on the experiments, answers to research questions and also gives directions for future research work.
Chapter 2

Background

Databases are an important part of many backend systems and applications. They can be defined as “A shared collection of logically related data, and a description of this data, designed to meet the information needs of an organization”. DBMS (Database Management System) is the software through which we can process and maintain our data. For the sake of simplicity and interpretation, database is considered as the collection of related data and the DBMS system [4].

2.1 SQL Databases

SQL stands for Structured Query Language. Databases based on relational model were dominant and were widely used in several domains. They had several contrasting features such as maintaining integrity and accuracy of the data. Also, they reduced the storage of redundant data in order to avoid wastage of storage space. Some of the popular SQL databases include open source solutions like MySQL, PostgreSQL etc. while proprietary solutions include those of Oracle, Microsoft SQL Server, Sybase etc.

Data stored in SQL databases is primarily stored in the form rows and columns. Columns consist of number of attributes the data can be classified into. Multiple records of data is organized as rows. Unique identification of a record can be done with the help of primary key. In addition, for databases consisting of several hundred records, indexing makes it easy to sort and search specific attributes from the available records. Also, foreign keys can be used to establish relationships between several tables.

RDBMS (Relational Database Management System) uses several commands
for management of data and the tables. These could be briefly categorized as DML (Data Manipulative Language) commands primarily used for selecting, inserting data etc., while DDL (Data Definitive Language) commands are used for creating, altering and dropping tables.

2.2 NoSQL Databases

In the later part of 2000s, new databases that have been strikingly distinct from traditional ones have emerged. These were essentially based on meeting the requirements of the new generation data-hungry services and overcoming certain inherent limitations of relational databases. They are classified under the term NoSQL, generally stated as Not only SQL. This is rather loosely defined term. Some of the generally observed characteristics of NoSQL databases are as follows [5]:

**Schema-free design:** Relational databases were an excellent solution and more relevant in the case of data which had complex relational structure. NoSQL databases are meant for those applications that do not impose strict restrictions on the database schema. This is significant in the sense that modern day applications like IoT had data consisting of simple structure. Schema-free design supports semi-structured data, object oriented and document based data models.

**Scalability:** Traditional RDBMS solutions like Oracle, MS-SQL Server were not so efficient in scaling in terms of cost and procurement. More capacity could be accrued only with more powerful machines. NoSQL databases provided a greater scope to enhance scalability by being able to add more machines with less capacity. It was way easier to add or remove nodes in a cluster.

**Relaxed ACID guarantees:** Relational databases were excellent choice for data which required full ACID guarantees. Often deploying large clusters while complying with these guarantees tend to have some limitations. When a query update is made, changes have to be propagated across all the nodes in the cluster for strong consistency guarantees. This is not the case for all types of applications. Some NoSQL solutions have been designed to provide higher availability while sacrificing a part of consistency.

**Storage needs:** Because large scale clusters can be deployed at rapid pace, these are capable of storing large volumes of data. Production clusters of companies like Netflix have been reported at capacities exceeding 300 TB [6].

The key to choose whether relational or NoSQL solutions is to understand
the requirements of the systems. If you have applications that have availability as main concern, it makes sense to use NoSQL databases while sacrificing some portions of consistency. They provide BASE guarantees which is basically available and eventually consistent model. Some solutions like HBase are developed to work with Map-Reduce frameworks like Apache Hadoop. This can significantly reduce the processing times and helps to deliver faster services.

All the available NoSQL solutions briefly comes under the following categories [7]:

- **Key-Value stores**: Data is stored in the form of records which is identified and can be retrieved by using its key. Some examples of key-value store databases include Riak, Voldemort etc.

- **Document Oriented**: Data is stored in the form of documents which itself can consist of several other key-value pairs, nested documents etc. MongoDB is an example of document store databases

- **Column Oriented**: Data is stored in the form as columns of records. This is particularly useful for reading and analyzing large volumes of data. HBase is a widely used column-oriented database.

- **Graph Databases**: They are used for storing data which have numerous interlinks between them. These are relevant for recommendation engines and social networks. Neo4j is an example of graph databases

### 2.3 CAP Theorem

CAP theorem gives us an insight into the capabilities and some of the trade-offs that could arise while implementing distributed systems. It was first proposed by Eric Brewer which states that among availability, consistency and partition tolerance, only two could be achieved at any instance of time [8]. It could be interpreted as, when the system undergoes network partition as is the case in most scenarios, it has to choose between consistency and availability [9].

**Consistency**: User must be able to access the latest changes made to the system. In order to achieve this, updates must be propagated across all the nodes in the cluster as they are written.

**Availability**: Client must always be able to access the system.
Partition Tolerance: The system should operate even at the occurrence of failure either due to network or machine flaws.

Discussion

Meticulous treatment of CAP theorem is necessary in order to fully understand its implications. Availability in the context of this theorem has a special meaning. All the individual partitions must be available even though the system caters to the needs of all its available clients [11]. A noteworthy point is that though consistency and availability cannot be attained simultaneously, they are tunable. In the sense that you can sacrifice a part of your consistency to gain higher availability and vice versa. One can achieve strong consistency by involving more number of nodes but this can drastically increase the request service times. There are some applications where availability is high priority than getting the up to date information. Hence, this is largely dependent on specific applications/systems.

Daniel J Abadi [12] argues that though CAP theorem looks like putting limitations, it does not impose any under normal circumstances except for certain types of failures.
2.4 ACID Compliance

Transaction is said to be a logical aggregation of several statements. These can consist of queries, commands etc [10]. Thomas Connolly, Carolyn Begg [4] defines it as “An action or series of actions, carried out by a single user or application program which reads or updates the contents of the database”. In general, transaction must meet the below requirements:

**Atomicity**: There should not be any incomplete execution of queries or statements in a transaction. Either all should be complete or none. This is important if changes to multiple accounts are involved. Additionally, this helps to maintain the integrity of the database.

**Consistency**: Execution of any operations in the database must be in accordance with the pre-defined rules. Database should not result in an invalid state due to the transaction.

**Isolation**: When a query is set to perform an operation on data that is shared, it should not be exposed to other operations simultaneously. It can be shared only after the completion of the current operation.

**Durability**: Systems are always susceptible to crashes and power outages. If such an event arises, the original state of the system must always be preserved.

**Discussion on ACID Compliance:**

Different authors give wide interpretations for the ACID guarantees. It is often assumed that relational databases strictly adhere to ACID properties while the databases that come under NoSQL category do not. Martin Fowler [9] argues that the key here remains is atomicity property. According to him, NoSQL category can be classified into Aggregate oriented and Graph databases. This is done based on defining an aggregate as collection of related objects that can be treated as a unit [13]. Graph databases such as Neo4j etc. also adhere to ACID guarantees while the various families that come under aggregate oriented databases such as column-stores, key-value pairs etc. do not.

There are certain limitations as to why it is difficult to implement large scale distributed systems while adhering to ACID guarantees. In relational databases, atomicity is extracted by implementing two phase commit protocol. In the case of distributed database, doing this across a number of nodes is complex and throughput becomes bottleneck factor. Given the large amount of data access, this is not a good solution. Replication of data is another area which makes the above said task difficult. Query latency levels are increased because of the need
to provide strong consistency levels [14].

Further, designers of NoSQL databases have come up with some solutions that can provide near-real ACID guarantees while still being able to scale horizontally [14]. Data in relational databases is replicated synchronously, meaning that the updates are passed on to the slave nodes from the master nodes in real-time. However, eventual consistency is a model where updates are propagated to the replicated nodes in an asynchronous manner. This is stated as "If no updates are made eventual access is followed" [15]. This is relevant in scenarios where updates made to data can appear a bit later as in the case of social networking sites.

2.5 MySQL Cluster

MySQL is one of the most popularly used open-source database. There are many storage engines compatible for it like InnoDB, MyISAM and NDB(Network Database) Cluster etc.

MySQL Cluster is an in-memory distributed database that is capable of providing high availability and replication. It uses NDB Cluster as its storage engine. It is inherently built-in to avoid single point failures owing to its design aim of providing high availability. MySQL Cluster has a hierarchical architecture. There are three different types of nodes which have a unique functionality. Node in the context refers to as process.

Management Node(ndb_mgmd): It is responsible for the overall management and maintenance of the cluster. It can be used for monitoring and troubleshooting the nodes in the cluster. Clients look up to the management node for configuration settings at startup. Usually, these can be stopped once the cluster is set in working condition. In most cases, these are redundantly deployed rather than on exclusive servers.

SQL Node(mysqld): They act as an interface between the client applications and the storage (or) data nodes. They receive the client requests from various clients and distribute the queries across the data nodes in the cluster. There are several APIs/Client options to connect to SQL nodes. These include Connector/J, C, and PHP API’s. Native access is also supported for NoSQL APIs like JavaScript, Memcached etc. in newer versions.

Data Nodes(ndbd): These are the nodes at which the original data is stored after it’s written. These are interconnected and the data is replicated at more
than one node. In case of node failure, SQL nodes can connect to other data nodes without much downtime. This makes up the case for high availability concerns.

MySQL Cluster supports synchronous replication i.e, whenever updates are made to the system, they are propagated to all the slaves in the cluster as well in real time. The advantage is that in case of any node failure, the down time for another node to take over is low due to the fact that the most recent updates are made to every node in the cluster. It is reported that asynchronous replication ability and multi-master replication were also added to newer versions of MySQL Cluster database [16].

In MySQL, fragments of data are stored within node groups. It also supports some features such as shared-nothing architecture wherein each node has its own set of resources like memory, CPU etc. Some of other key features of MySQL Cluster are, High availability, and SQL, NoSQL Interfaces [17].

2.6 Apache Cassandra

Relational databases were originally developed for centralized type of architecture. While this was a good case for relational data for strong consistency guarantees, the data volumes were gradually exploding. In order to enhance the capacity, the feasible solution was to scale it vertically, i.e. procure machines of large capacity which in turn had large size. This was not an effective method given
the huge capital investment etc. Keeping in mind the above concerns, Amazon came up with Dynamo, a highly available Key-Value store for its own platforms. It addressed the challenges of high availability while having limitations on strong consistency levels. Google’s BigTable is another distributed storage system that was based on similar design.

Cassandra [18] was developed at Facebook Inc. and gradually grew as an Apache project. It is inspired heavily from Dynamo and BigTable. It was built mainly on the principle of being able to scale horizontally, thus reducing the chances of single point of failure. Modern datacenter infrastructure typically consists of thousands of nodes. Machine failure is a very common phenomenon given the massive scale of nodes. Any system that has the concerns of high availability must cope up and run without breakdown in the services. Cassandra’s architecture is designed as a best fit for this.

Cassandra has a flat and distributed architectural model. All nodes in this design are equal and have no different type of functionality. It was designed keeping in view certain requirements such as:

**Scalability:** Ability to scale by adding large number of machines, often referred to as horizontal scalability.

**Round-the-clock Availability:** The designed system should be highly available. In the event of node failure, client should be able to connect, query and access the information.
Chapter 2. Background

Big Data: Should be able to store large amounts of data from multiple sources, often in the range of Petabytes.

Cassandra supports CQL (Cassandra Query Language) which uses data manipulation language and data definition language commands as in SQL for administration, processing and maintenance of database. In Cassandra, nodes communicate to each other through ‘gossip’ protocol. It is a peer to peer modelled communication protocol that enables nodes to exchange information about their state and this helps in ensuring that all nodes in the cluster are aware of the nodes present. Whenever a new node joins the cluster, it communicates to the seed node to initiate the process. However, there is no designated role as seed node and there could also be several seed nodes in any given cluster [20].

Cassandra uses the concepts of row, column and also column family, keyspace which are similar to table and database in MySQL. Cassandra has in-built features to support multi data-center architecture model. Data can be even replicated in the data centers which can be done while designing the schema for the data models. There is also a scope for choosing the replication factor which comes down to how many nodes we want the data to be replicated and uses tokens for storing replicas in different nodes. Whenever data is written to Cassandra, it is first related to a commit log and then written to an in-memory based structure called memTable. Once these are full, data is flushed in order to be written to the disk. This mechanism also reduces the I/O Operations at the disk level [20].

2.7 Requirements of Mediation Systems

Each application has some inherent needs. Likewise, from our investigation, we found some of the requirements of the mediation systems for Ericsson.

Single solution: A single database is needed which has support for both event and batch mode of operations.

Storage size: Since the data involved in mediation processing is typically very large, the database should be capable of handling volumes from few GBs to several TB.

ACID Support: ACID compliance was a desired feature at row level as well as for batch mode.

Shared nothing architecture: The database solution should not be dependent on shared storage.
Cloud integration: There should be an extensive support to deploy the solution in the cloud.

Linearly Scalable: Performance of the database should not degrade and perform consistently well even while adding more servers.

Fault tolerance: The database should avoid single point of failures.

Distributed nature: The database should be highly distributed in nature and provide mechanisms for reads and writes at different nodes in the architecture.

Data model: Most of the processing involves usage of keys in the database. So databases which can store as key value pair are to be preferred.

Cost: Cost is an important factor in the deployment of any service. Open-source based solutions are to be chosen in this scenario.

Throughput: The database should have a throughput of range 50,000 - 160,000 operations/sec.

These are the requirements in general. Though there are multiple requirements, there are some practical difficulties in realizing them all in a single solution in the sense it is difficult to find databases which are highly available, have strong consistency model while handling scalability issues. The idea was to find the databases which can comply with majority of the requirements. Based on our initial work we chose to evaluate MySQL Cluster and Apache Cassandra.
In this chapter, we present the details about the experiment, elaborate about the workload conditions, metrics used for measurement etc. Several experiments were carried on MySQL Cluster and Apache Cassandra.

MySQL Cluster and Apache Cassandra were chosen because of several features that could be suitable for mediation systems. MySQL Cluster is a distributed database and provided multi-master shared nothing architecture. Also, it guaranteed high availability and provided auto-sharing for reads and writes. Apache Cassandra was also a distributed database and had decentralized flat architecture. It provided mechanisms for high availability and eliminated bottlenecks for failure scenarios. It provided a SQL like interface and easy scalability features. Also, there was extensive documentation and good support in other channels for these two databases.

3.1 Experimental environment

The experiment test bed consists of a cluster consisting of five servers. These are interconnected with 1Gbps link using a switch. Configuration of each of the servers is described in Table 3.1.

Oracle Java Runtime Environment (JRE SE 8) and Java Development Kit (JDK 1.8) are installed in the systems.
Chapter 3. Experiment Method

<table>
<thead>
<tr>
<th>CPU</th>
<th>Intel Quad Core i5 4th Gen</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed</td>
<td>3.2GHz</td>
</tr>
<tr>
<td>RAM</td>
<td>32GB</td>
</tr>
<tr>
<td>OS</td>
<td>RHEL 6.5</td>
</tr>
<tr>
<td>Hard Drive</td>
<td>130GB SSD</td>
</tr>
</tbody>
</table>

Table 3.1: Experiment Environment

Drivers used:

MySQL Connector:

Connector J is the native Java driver for MySQL Cluster that is compatible with JDBC standards. JDBC is the standard connectivity option for interacting with multiple databases using Java. It can be used for connecting to the databases, sending SQL queries and processing the results [21].

CQL Connector:

DataStax Corporation provided a Java driver for Apache Cassandra licensed under Apache License. It works for Cassandra Query Language (CQL) which is similar to SQL and makes it easy for applications to interact with Cassandra database [22].

Database Versions used:

- MySQL Cluster 7.4.4
- Apache Cassandra 2.0.13

3.2 Test bed Setup

3.2.1 Experiment 1

There are two experiments conducted as a part of our work. The first experiment is related to measuring the throughput of batch operations. There are two clients connected to the cluster from which multiple clients are run by varying the number of threads and these try to process the queries concurrently. Queries are performed from the two systems which are in the same network as the main cluster. The database is preloaded with some data. The workload applied here is 50% reads and 50% writes. The experiment is conducted without varying the number
of nodes in the cluster. Accurate time between start and end of the process is measured using `System.currentTimeMillis()` which is a method in Java.

![Figure 3.1: Setup for Experiment 1](image)

Both the systems are started simultaneously. Before the experiment, it was ensured that both of them are time synchronized. During this process, there were two measurements obtained. Select throughput that is obtained from Client A and Insert throughput that is obtained from Client B. The above experiments are repeated for three times to ensure reliability and mean values are taken into account. Throughput gives a measure of how many operations can be processed within a given time frame.

**Parameters:**

The following are the parameters considered for this experiment:

- **Type of databases used:** MySQL Cluster and Apache Cassandra
- **Multiple number of threads:** Number of threads are accessing the database concurrently in both systems
- **Data type:** Batch type which involves multiple data operations at row-level
• **For Read operations:** Number of operations is fixed

• **For Write Operations:** Number of operations is fixed

### 3.2.2 Experiment 2

The second experiment is related to measurement of total time taken while multiple threads try to access the system concurrently and also by varying number of queries. Additionally, average response time is considered when single thread tries to access the system. There is a single client that is connected to the cluster containing a fixed number of nodes. The experiment is conducted in four parts, two for insert scenario and two for select scenario.

![Figure 3.2: Setup for Experiment 2](image)

Total time taken and average response time are the parameters considered to determine the time taken to process a set of queries and are analyzed in this experiment. Queries performed include Insert for event and batch mode. While Select is another query type which is done in two parts, incrementally increasing in one case and proportionately in other. This is done in order to identify the behavior of both the databases rather vigorously. Total time taken indicates how much time it takes to process a set of queries while response time is a measure of time taken to process a single query.
Chapter 3. Experiment Method

Parameters:

The following are the parameters considered for this experiment:

- **Type of database used**: MySQL Cluster and Apache Cassandra

- **Multiple number of threads**: Multiple number of threads are considered for the total time taken parameter. Single threaded client is considered for representing average time.

- **Data type**: Event type which involves single row operations Batch type which involves multiple data operations at row-level

- **For Write Operations**: Number of Operations is varied for event and batch type of data

- **For Read Operations**: Number of Operations is varied in two different approaches

![Database Schema](image)

Figure 3.3: Database Schema

**Layout of Tables:**

There are two tables that are part of our schema considered for the experiment. Table 1 contains two entries with primary key as value, while Table 2 contains four entries with primary key as File index. For Batch operations, for every 17773 inserts into Table 1, there is an entry created in Table 2. These are processed in parallel i.e a bunch of operations are processed in a single statement. Event operations are processed sequentially wherein each query is processed one after the other.
3.3 Description of Workload

There are two operations of prime interest for mediation platform.

**INSERT:**

The workload is designed such that for every 17773 inserts in Table 1, there would be an entry created in Table 2. For batch processing several SQL statements are clubbed using `addBatch()` in JDBC 2.0. Auto Commit feature would be disabled making multiple operations as one transaction. Following command is used to insert values into the database.

\[
\text{INSERT INTO duptera VALUES(value, fileindex);}
\]

For event based operations, for every 100 inserts in Table 1, there will be an entry created in Table 2. Auto Commit would be enabled making each operation an individual transaction.

**SELECT:**

Read operation is mainly focused on retrieving a single value from Table 1. This retrieval time depends on entries already in the database. As the retrieval is of only single value, there exists no difference for event and batch operations. To retrieve values from database the following command is used.

\[
\text{SELECT * FROM duptera WHERE value=PK;}
\]

3.4 Assumptions

Also, the databases that are included in this experiment have different architectural design. MySQL Cluster has a hierarchical design and Cassandra is a peer-to-peer distributed model. As seen in the above sections, the architecture of MySQL Cluster is hierarchical and Apache Cassandra is peer-to-peer. For comparison needs, Cassandra requires four nodes while MySQL Cluster requires 4 data nodes and in addition, a management node and also an SQL node. So for MySQL, one node has an overlapping role so that the number of nodes is limited to only four for both the databases. In order to have a fairer comparison between the two databases, database tuning was not done.
Chapter 4

Results

The following section presents the results of the experiments conducted. Brief discussion on the results is also presented thereafter. Numerical data is presented in the form of tables in Appendix to have greater insight into the results. In order to improve the accuracy and reliability factor, experiments are repeated thrice and the mean of them is represented in the results part. Throughout the course of discussion Insert and Writes are synonymous while Select and Reads would mean the same.

4.1 Part 1

4.1.1 Insert Throughput

The below behaviour is for batch data and an important point to note is that this is for measurement of throughput under 50% reads and 50% writes workload. Throughput for writes and reads are separately calculated from two separate client systems. Set up and experimental conditions are described under Section 3.2. Number of Operations is fixed as 5,687,680, this is done in accordance with the batch size which is fixed at 17773.

For Insert operations, Cassandra has a very high throughput when compared to MySQL Cluster and it increases linearly till thread factor 8. Thereafter, it has nearly same throughput for 8 and 16 threads. MySQL also shows linear behaviour initially but throughput remains near-constant for 4 and 8 threads and slightly degrades for 16 threads too. This behaviour could be possibly attributed to near saturation of the system when higher number of clients try to access the system.
Chapter 4. Results

4.1.2 Select throughput

For Select operations, MySQL exhibits a very linear behaviour, as it can be seen from the graph. Cassandra also exhibits linear read performance until 8 threads. However it degrades slightly for higher number of concurrent threads in client systems.
Chapter 4. Results

4.2 Part 2

Below section deals with results obtained as a part of experiment 2. The measured metric is total time taken when multiple threads access the system. In order to represent the huge data set and for comparison needs, a complex graph like the below one has been chosen. Also the workload applied in this scenario is 100% reads(selects) or 100% writes(inserts). Additionally to have a clear understanding of the behaviour of databases average response time are also plotted. Also, M here represents a Million.

The experiment is conducted in four parts, two insert (write) cases and two select (reads) scenarios.

4.2.1 Insert (Event)

![Figure 4.3: Total Time taken for Insert queries, Event Data](image)

We can infer from the above graph that time taken to process same number of operations under different number of threads is approximately 5 times lower for Cassandra compared to MySQL Cluster. Total time taken is the measure of time required to complete a fixed number of operations. While we look at the figures it is observed that this gap is not so wide for higher number of clients at 16.
The above graph gives us the average response time while inserting variable number of operations for event data which are essentially single row operations. Response time here gives an idea about how much time it takes to process a single query in different cases. This is calculated by the total time taken divided by the total number of operations for thread factor as one. The behaviour of both MySQL and Cassandra is linear, though response time for same number of entries is very less in case of latter when compared to the former. The linearity observed in the graphs could be possibly due to caching phenomenon of the databases.

4.2.2 Insert (Batch)

Total time taken to insert queries of the range 2M, 4M... 10M is measured in this part for batch data. The workload condition is 100% reads or 100% writes.

The graph shown depicts the behaviour of both databases for the time taken to insert queries while multiple threads access the system. Time taken for Cassandra is several times lesser, in the range of 6 times for lower threads and as low as 10 times at higher thread values. The difference between the two could be seen clearly as we scale the size of the operations.
4.2.3 Select (Incremental)

In this scenario, the database is preloaded with 50 Million entries. We then analyzed the total time taken parameter to understand the behaviour of MySQL and Cassandra for reading various number of operations in the scale of 2M, 4M, 6M, 8M and 10M (M stands for Million) etc.

For lower number of client threads, Cassandra has near equivalent behaviour
compared to MySQL Cluster in terms of total time taken. As we increase the thread factor, there is considerable difference between the two and Cassandra has almost magnitude of half compared to MySQL.

![Figure 4.7: Average Response time for Select queries (Incremental)](image)

While considering average response time performance, both the databases have very linear behaviour as it could be seen from the graphs above. An interesting aspect is though Cassandra has lower response times taken when compared to MySQL, the magnitude of difference is not high as we have seen for insert queries.

### 4.2.4 Select (Proportional)

The below one is to correlate the total time taken to read 10% entries for each of databases containing 5M, 10M, 15M and 20M entries respectively and is taken for both MySQL Cluster and Apache Cassandra.
Chapter 4. Results

Figure 4.8: Total time taken for Select queries (Proportional)

When considering multiple threads, Cassandra has lesser average time taken for lower number of threads but for higher number of threads, the time taken is almost the same.

Figure 4.9: Average Response time for Select queries (Proportional)

The behaviour is plotted as above. Response times for both databases is linear and although the Cassandra has better performance than MySQL, it is very minimal.
Chapter 5

Discussion

5.1 Related Work

Several works have been carried out previously in the study of relational and non relational databases. Some are related to studies in general [23] while others [24] focus on specific functionality or application based systems.

Ananda Sentraya Perumal Murugan conducted a study of NoSQL and NewSQL databases for data aggregation on Big Data [25]. The focus of the work has been on carrying out performance evaluation on MySQL Cluster and HBase for sensor data analysis. It was reported that MySQL Cluster had higher performance than HBase but it lacked certain functionalities like implementation of Map-Reduce framework which is tightly integrated with HBase. This also contains a brief description in general about all the available database solutions and their features.

Eric Eliasson and Daniel Öhrlund [26] did an empirical study to identify the capabilities for sensor data of vehicular monitoring systems. The reported results were promising that Cassandra was able to meet their needs for scalability, storage and processing needs.

Ruslan Mukhammadov [27] analyzed ways to deal with large data for Remote patient monitoring systems. MySQL and HBase were benchmarked by using Yahoo Benchmark tool (YCSB). MySQL was reported to have better read performance and HBase had better write performance. Ansir Rafique [28] conducted an investigation about databases for storing historical financial data. Cassandra was extensively evaluated keeping Apache Lucene and MySQL. It drew several conclusions about its scalability, performance and indexing.
Tilmann Rabl et al [29] did a comprehensive performance evaluation on a number of databases using YCSB which is a generic framework for evaluation of Key-Value stores [30]. Tai Anh Mai Phan investigated several databases in order to identify the suitable databases for storing large data generated from IoT devices. Cloud environment was chosen as the preferred platform for this study.

In most of the above scenarios, when MySQL was studied, the underlying storage engine was InnoDB, while MySQL Cluster version uses NDB Cluster as the storage engine. High availability design is a hallmark feature of MySQL Cluster and is designed to support distributed clusters on a large scale. There are considerable difference between the two and are listed in the official documentation [31].

There are very few works which were focused on mediation systems. Also much work has not been carried out in regards to processing techniques like batch etc. which this thesis has major focus on. Mediation systems specifically uses two types mainly listed as batch and event operations.

5.2 Reliability and Validity Analysis

Reliability is an important requirement that must be fulfilled in any research work. It helps to ensure in getting similar results whenever the experiment or study is done under similar conditions [32]. For analyzing the recent works in evaluation of databases, standard sources were chosen. These include literary sources like diva-portal, Compendex, ACM digital library and IEEE Xplore. There were several other non-standard sources were also used but it was ensured that those which are authored by professionals in their respective fields. Whitepapers are also used in order to gain understanding of certain aspects from the authorized websites of MySQL, Cassandra etc. Proper documentation with relevant citations for sources of information also enhances the reliability factor of the research study.

Validity threat analysis can be useful in understanding the factors that could threaten the validity of the research work either directly or indirectly. According to Wohlin [33], there are four threats for validity viz. internal validity, external validity, construct validity and conclusion validity. These are discussed briefly with relevance to this thesis work.

**Internal Validity:**

Internal Validity is concerned with whether the treatment of independent
variables and its impact on the outcome of the system [33]. Previous works have been discussed which are from independent sources and unbiased literary sources. Though there could be vast number of parameters involved, only parameters such as throughput and total time taken were taken which could be used for evaluating databases for multiple number of clients. Further the experiment is conducted in isolated environment and it was ensured that database state was restored after every phase of the experiment.

**External Validity:**

External Validity relates to the extent at which the results obtained could be generalized beyond the extent of the project [33]. Experimentation on databases with different architectures is a potential threat. Necessary discussion and the ways to overcome is detailed in the work. Characterizing the behavior of databases with regards to application-specific systems could also be a threat. However, description of the internal system needs of the mediation systems helps in accruing the results with systems that possess only such behavior.

**Construct Validity:**

Construct Validity refers to the relation between the theory behind the experiment and observation [34]. In order to overcome this, relevant aspects of databases, mediation systems and previous works have been studied in detail before the experimentation phase.

**Conclusion Validity:**

Conclusion validity refers to the degree to which the results obtained are reliable. Experiment results are repeated and mean is calculated in each case to improve the reliability factor. Results from previous works has also been taken into account to get an insight into the performance of the databases. Also, these were discussed with the regular meetings with the external supervisor.
The purpose of the thesis was to investigate for the ideal databases which can serve the requirements of the mediation systems. The database solutions were analyzed which are capable of dealing and processing large data volumes that are usually associated with mediation systems. Two databases namely, MySQL Cluster and Apache Cassandra were studied and their performance was benchmarked using a Java based client.

Performance was benchmarked for these two databases. Measuring variables included throughput, total time taken. In addition, average response time was also included in order to have a clear picture of the behaviour of the databases. In addition to these, other parameters which were varied are multiple number of threads in the clients. The two workload conditions which were considered are 50% reads and 50% writes (or)either 100% reads or 100% writes.

Insert and Select throughput were measured for both the databases and are discussed in detail with graphs under part 1 in the results section. Performance of the databases is evaluated when data volumes are increased in terms of total time taken and average response time. These are also included under part 2 of the results section. Impact of multiple threads in the client is also observed.

By analyzing the measurements, certain observations can be made. Apache Cassandra has a very high write throughput compared to MySQL Cluster. On increasing the threads, both of them tend to exhibit a near constant behavior at higher number of threads. While considering read throughput, both the databases have a linear performance, even though Cassandra has a slightly more performance for lower number of threads and this behavior decreases slightly at higher number of threads.
Chapter 6. Conclusions and Future Work

While analyzing the total time taken behavior, two scenarios were considered for each of write and read performance. For event operations, time taken to process several insert operations was nearly 5 times lower for Cassandra when compared to MySQL. For batch operations, the time taken was lower for Cassandra as well, but it was still lower when higher number of threads were applied.

For read performance, two approaches were considered, one incrementally increasing the read operations and other in a proportional manner. For the former approach, Cassandra has near equivalent total time compared to MySQL for lower number of threads. As we increase the thread factor, there is considerable difference between the two and Cassandra has almost magnitude of half compared to MySQL. While for increasing the number of queries proportionally, when considering multiple threads, Cassandra takes less time for lower number of threads but for higher number of threads, the time taken is almost the same.

Linking to Research Questions:

1. **What is the throughput(operations/sec) behaviour of the batch data for the databases considered?**

   Apache Cassandra has a very high write throughput compared to MySQL Cluster. On increasing the threads, both of them tend to exhibit a near constant behavior at higher number of threads. While considering read throughput, both the databases have a linear performance, even though Cassandra has a slightly more performance for lower number of threads and this behavior decreases slightly at higher number of threads.

2. **What is the effect on the performance of the databases considered in terms of total time taken when data volumes are scaled?**

   For event operations, time taken to process several insert operations was nearly 5 times lower for Cassandra when compared to MySQL. For batch operations, the time taken was lower for Cassandra as well, but it was still lower when higher number of threads were applied.

   Cassandra has near equivalent total time parameter compared to MySQL for lower number of threads. As we increase the thread factor, there is considerable difference between the two and Cassandra has almost magnitude of half compared to MySQL. While for increasing the number of queries proportionally, when considering multiple threads, Cassandra takes less time for lower number of threads but for higher number of threads, the time taken is almost the same.
Apache Cassandra and MySQL Cluster are compatible for both event and batch based operations. MySQL Cluster guarantees full ACID transaction support [35]. Writes to Cassandra are atomic, isolated and durable. Consistency is not applicable to Cassandra as there is no referential integrity or support for foreign keys. Instead, it trades for high availability and performance [20]. However in the context of CAP theorem, a noteworthy point is that Cassandra supports tunable consistency, in the sense that we can tune the system to have a strong or eventually consistency level.

In general, Cassandra has tremendous performance for write operations which could be possibly due to the way it writes to the disk. In case of read operations too, Cassandra is slightly better compared to MySQL though this behaviour varies moderately in certain scenarios. Although, this might lead to conclusions that Cassandra is far superior in terms of performance, reasonable factors like transaction support also have due consideration, MySQL strongly adheres to ACID guarantees while Cassandra has some relaxations, notably for Consistency guarantees. A better solution could be use of multiple databases for different needs which can bring out the best of both worlds slated as polyglot persistence by Martin Fowler [9]. Choice of databases that can be used, may also be dependent upon factors like system integration, licensing costs etc.

**Future Work**

The scenario of database solutions is widely changing. We have witnessed a far more introduction of new generation databases in the last decade. In the recent times, there are some new solutions proposed such as Trafodion(SQL running on top of HBase), using high performance hardware for more transaction speeds, etc. These are not mature as of now and their relative usage in specific fields is a matter of concern. These could be studied in future. Integrating them with processing frameworks like Hadoop or Apache Storm could also be a field of interest.
References


References


[14] D. Abadi, “DBMS Musings: The problems with ACID, and how to fix them without going NoSQL.”


References


## APPENDIX

<table>
<thead>
<tr>
<th>Number of Threads</th>
<th>MySQL Cluster 7.4.4</th>
<th>Apache Cassandra 2.0.13</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2299.744</td>
<td>2472.065</td>
</tr>
<tr>
<td>2</td>
<td>4746.296</td>
<td>5482.838</td>
</tr>
<tr>
<td>4</td>
<td>9706.097</td>
<td>11700.008</td>
</tr>
<tr>
<td>8</td>
<td>18356.765</td>
<td>21842.601</td>
</tr>
<tr>
<td>16</td>
<td>35609.034</td>
<td>29540.746</td>
</tr>
</tbody>
</table>

Table 1: Select Throughput (Operations/sec), Batch Data, 50% reads and 50% writes workload

<table>
<thead>
<tr>
<th>Number of Threads</th>
<th>MySQL Cluster 7.4.4</th>
<th>Apache Cassandra 2.0.13</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12577.967</td>
<td>76775.843</td>
</tr>
<tr>
<td>2</td>
<td>17254.035</td>
<td>118794.091</td>
</tr>
<tr>
<td>4</td>
<td>18329.004</td>
<td>160517.400</td>
</tr>
<tr>
<td>8</td>
<td>18063.377</td>
<td>186684.683</td>
</tr>
<tr>
<td>16</td>
<td>17835.995</td>
<td>198240.126</td>
</tr>
</tbody>
</table>

Table 2: Insert Throughput (Operations/sec), Batch Data, 50% reads and 50% writes workload
<table>
<thead>
<tr>
<th>No. of Threads</th>
<th>Database</th>
<th>Number of Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0.2Million</td>
</tr>
<tr>
<td>1</td>
<td>MySQL</td>
<td>227.893</td>
</tr>
<tr>
<td></td>
<td>Cassandra</td>
<td>47.915</td>
</tr>
<tr>
<td>2</td>
<td>MySQL</td>
<td>105.278</td>
</tr>
<tr>
<td></td>
<td>Cassandra</td>
<td>21.324</td>
</tr>
<tr>
<td>4</td>
<td>MySQL</td>
<td>49.575</td>
</tr>
<tr>
<td></td>
<td>Cassandra</td>
<td>10.178</td>
</tr>
<tr>
<td>8</td>
<td>MySQL</td>
<td>23.308</td>
</tr>
<tr>
<td></td>
<td>Cassandra</td>
<td>5.924</td>
</tr>
<tr>
<td>16</td>
<td>MySQL</td>
<td>11.249</td>
</tr>
<tr>
<td></td>
<td>Cassandra</td>
<td>4.956</td>
</tr>
</tbody>
</table>

Table 3: Total Time taken for Insert Operations (in sec), Event Data
100% reads or 100% writes workload

<table>
<thead>
<tr>
<th>No. of Threads</th>
<th>Database</th>
<th>Number of Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>2Million</td>
</tr>
<tr>
<td>1</td>
<td>MySQL</td>
<td>145.759</td>
</tr>
<tr>
<td></td>
<td>Cassandra</td>
<td>24.388</td>
</tr>
<tr>
<td>2</td>
<td>MySQL</td>
<td>105.348</td>
</tr>
<tr>
<td></td>
<td>Cassandra</td>
<td>14.854</td>
</tr>
<tr>
<td>4</td>
<td>MySQL</td>
<td>97.972</td>
</tr>
<tr>
<td></td>
<td>Cassandra</td>
<td>11.063</td>
</tr>
<tr>
<td>8</td>
<td>MySQL</td>
<td>98.93</td>
</tr>
<tr>
<td></td>
<td>Cassandra</td>
<td>9.571</td>
</tr>
<tr>
<td>16</td>
<td>MySQL</td>
<td>99.158</td>
</tr>
<tr>
<td></td>
<td>Cassandra</td>
<td>8.697</td>
</tr>
</tbody>
</table>

Table 4: Total Time taken for Insert Operations (in sec), Batch Data,
100% reads or 100% writes workload
<table>
<thead>
<tr>
<th>No. of Threads</th>
<th>Database</th>
<th>Number of Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>2Million</td>
</tr>
<tr>
<td>1</td>
<td>MySQL</td>
<td>818.345</td>
</tr>
<tr>
<td></td>
<td>Cassandra</td>
<td>563.359</td>
</tr>
<tr>
<td>2</td>
<td>MySQL</td>
<td>383.154</td>
</tr>
<tr>
<td></td>
<td>Cassandra</td>
<td>243.896</td>
</tr>
<tr>
<td>4</td>
<td>MySQL</td>
<td>180.463</td>
</tr>
<tr>
<td></td>
<td>Cassandra</td>
<td>158.113</td>
</tr>
<tr>
<td>8</td>
<td>MySQL</td>
<td>85.665</td>
</tr>
<tr>
<td></td>
<td>Cassandra</td>
<td>53.659</td>
</tr>
<tr>
<td>16</td>
<td>MySQL</td>
<td>49.072</td>
</tr>
<tr>
<td></td>
<td>Cassandra</td>
<td>30.031</td>
</tr>
</tbody>
</table>

Table 5: Total Time taken for Select Operations(in sec)
Incremental approach

<table>
<thead>
<tr>
<th>No. of Threads</th>
<th>Database</th>
<th>Number of Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>500,000 from 5M</td>
</tr>
<tr>
<td>1</td>
<td>MySQL</td>
<td>205.203</td>
</tr>
<tr>
<td></td>
<td>Cassandra</td>
<td>186.894</td>
</tr>
<tr>
<td>2</td>
<td>MySQL</td>
<td>96.032</td>
</tr>
<tr>
<td></td>
<td>Cassandra</td>
<td>83.517</td>
</tr>
<tr>
<td>4</td>
<td>MySQL</td>
<td>45.294</td>
</tr>
<tr>
<td></td>
<td>Cassandra</td>
<td>37.271</td>
</tr>
<tr>
<td>8</td>
<td>MySQL</td>
<td>21.012</td>
</tr>
<tr>
<td></td>
<td>Cassandra</td>
<td>18.288</td>
</tr>
<tr>
<td>16</td>
<td>MySQL</td>
<td>12.252</td>
</tr>
<tr>
<td></td>
<td>Cassandra</td>
<td>12.991</td>
</tr>
</tbody>
</table>

Table 6: Total Time taken for Select Operations(in sec)
Proportional approach