QUERY OPTIMIZATION IN BIG DATA USING HADOOP, HIVE AND NEO4J

SUMMER INTERNSHIP PROJECT REPORT

Submitted by

M. ARUN(2016103010)
S. BEN STEWART(2016103513)
P. SANJAY(2016103580)

COLLEGE OF ENGINEERING, GUINDY

ANNA UNIVERSITY: CHENNAI 600 025

MAY 2018

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report "**QUERY OPTIMIZATION IN BIG DATA USING HADOOP, HIVE AND NEO4J"** is the bonafide work of "**M.ARUN (2016013010), S. BEN STEWART (2016103513), P. SANJAY (2016103580)"** who carried out the project work under my supervision.

COORDINATOR
DR. T. RAGHUVEERA
SR. Assistant Professor
Department of
Computer Science and Engineering,
College of Engineering, Guindy
Anna University.

MENTOR
DR. T. RAGHUVEERA
SR. Assistant Professor
Department of
Computer Science and Engineering,
College of Engineering, Guindy
Anna University.

HOD
DR. D. MANJULA
Professor and Head
Department of Computer Science and Engineering
College of Engineering, Guindy
Anna University

ACKNOWLEDGEMENTS

We would like to thank our respectable Dean **Dr. T. V. Geetha** for providing us with this internship at college where we could get a chance to do this project.

We thank the Head of our department **Dr. D. Manjula** who provided us with the lab facilities and all that was necessary for us.

We had some very good guidance and support from our coordinator **Dr. T. Raghuveera**. With this project being conducted in the month of May, and it being the summer vacation, I would be grateful to all the staff who stayed back to help us with timely guidance and motivation especially **Dr.T.V. Gopal**.

We thank the research scholars **Mrs. Shiney Jeyaraj** and **Mr. D. Naveen Raju** who formed the integral part of this intern guiding all the teams.

Also we would like to thank the **other teams** who helped us out with the issues we faced both on and off the project the whole month, with team work and coordination between the teams being strengthened for good. Finally, last but not the least we thank **our parents** who allowed us to take part of this intern and also our **siblings** who could not have our company this summer.

Thanking you

M. Arun
S. Ben Stewart
P. Sanjay

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 PROBLEM STATEMENT

To store, retrieve and analyze Big data in an efficient way using both relational as well as graph database. The Hadoop architecture encompassing Hive, HDFS and Mapreduce is chosen as the relational database while Neo4J is taken as the graph database.

2 PROBLEM DESCRIPTION

The data to be queried has to be loaded into the hdfs storage space provided by Hadoop. The database schema has to be created and the data should be loaded into the database used by Hive which provides the interface for the SQL like syntax. The queries have to be executed and the execution times are noted down. The optimization techniques like indexing, hashing, bucketing and partitioning have to be applied and the performance gains are noted. Also the graph database implementation in Neo4j with data ingestion into it and the cypher queries have to be executed.

3 WORKFLOW

The figure 3.1 sums up the workflow for this project.

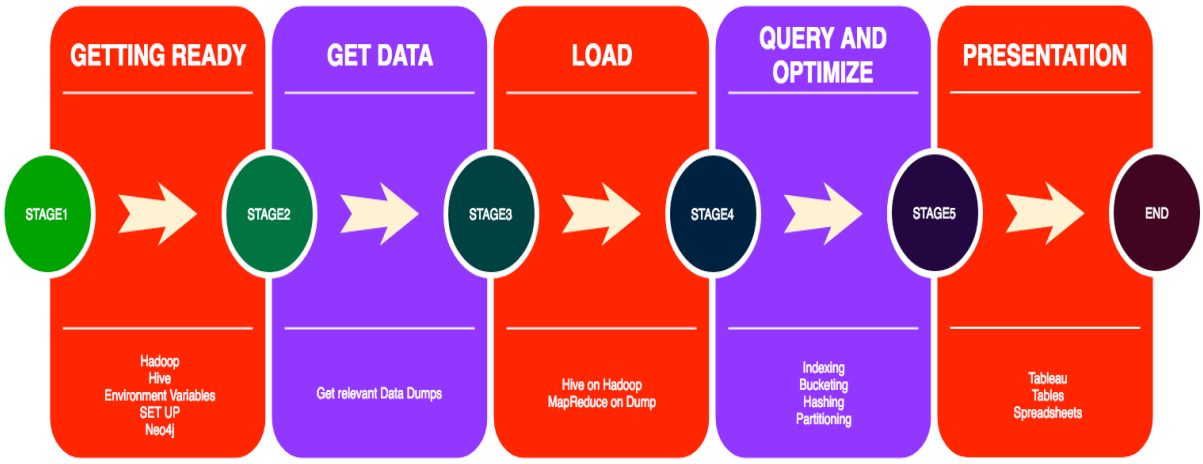

Figure 3.1 General outline of workflow

3.1 INTRODUCTION

In the first phase the Hadoop environment is setup along with Hive. The Hadoop set up is checked if the datanode, namenode, resource manager and the node manager are running.

A test sample data is loaded into the HDFS portioning and the ownership commands be tried out. A Mapreduce task on a simple join with two files (preferably in csv format) can be done using Java. Hive must be set up with all the permissions. A table can be created to check for the right permissions. The above steps make sure that all issues with the installation are fixed.

3.2 HADOOP

Hadoop is an open source software framework for storing data and running applications. It provides massive storage for any kind of data, enormous processing power and the ability to handle limitless tasks. Also Hadoop provides a reliable shared storage and analysis system for large scale data processing.

- Storage provided by HDFS(Hadoop Distributed File System)
- Analysis is provided by MAPREDUCE

3.3 HDFS

HDFS (Hadoop Distributed File System) is used to store large amount of data. Hadoop file storage consists of two nodes those are

- NameNode
- DataNode

Namenode stores all metadata such as filename,location of each block on datanode, file attributes and etc. and it keeps the metadata in the machine's RAM for fast lookup and also the filesystem metadata size is limited to the amount of available RAM on the namenode.
Datanode just stores the file content as blocks. Different blocks of the same file are stored on the different datanodes.

3.4 MAPREDUCE

Mapreduce is a processing technique. The mapreduce algorithm contains two important tasks namely map and reduce. Map takes the set of data and converts it into another set of data, where individual elements are broken down into key value pairs. the intermediate result from each block is shuffled and sorted which is then given as input to the reducer and combines those data tuples into a smaller set of tuples. This is the way that the mapper and the reducer works.

Hadoop contains some components to undertake the process those are

- HIVE
- HBASE
- PIG

And we were working under the concepts of hive.

3.5 HIVE

Hive is a data warehouse infrastructure tool to process structured data in Hadoop. It helps to summarize the big data and makes querying and analyzing easy. And the important thing about hive is, it is not a relational database. Hive is a platform used to develop SQL type scripts to do Mapreduce operations, by providing SQL type language for querying called HiveQL and it is familiar, fast, scalable and extensible comparing to other platforms like HBase, pig and etc, and it was first developed by FaceBook to solve the complex data.

After the Hive finishes the query execution, the result is submitted to the jobtracker, which resides on yarn. The job tracker consists of mapreduce task which runs the mapper and reducer job to store the final result in the HDFS. The Map task deserializer(reading) the data from the HDFS and reduces task serializes (writing) the data as the result of the hive query.

3.6 LOADING DATA

In the second phase the data dump is stored in the hdfs storage space. The data dump in the hdfs storage is then loaded into the metastore that Hive supports. The support jar packages to access the data stored in specific formats is required in the

$HIVE_HOME/lib folder. Now with the third phase some idea about the schema of the database is essential.

the database schema of a database system is its structure described in a formal language supported by the DBMS. The term “schema” refers to the organization of data as a blueprint of how the database is constructed.

3.7 QUERYING AND OPTIMIZATION

The queries are now run on the database. The optimization is done as explained in figure 3.2.

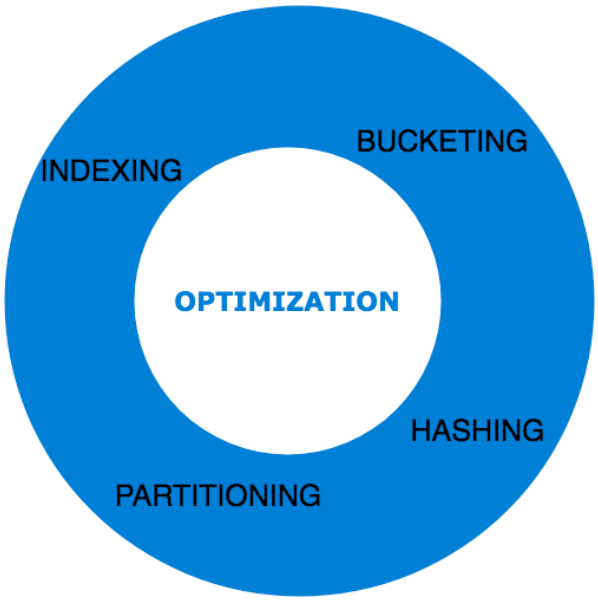

Figure 3.2 Optimization techniques

3.8 PARTITIONING

Partitioning divides the table into related parts based on a particular attribute. It plays a vital role when the database is really huge as each time when a query is written in Hive it has to read the entire dataset and convert it into a mapreduce job and submit it into the Hadoop cluster which will take a lot of time when the data is really large. So by partitioning we split the data and thereby reduce the time of execution.
The syntax is

```
create table tablename (column datatype,....)partitioned by (partition1 datatype,....);
```

Hive partitioning is subdivided into cluster or buckets

3.9 BUCKETING

Bucketing is also a technique for decomposing data sets into more manageable parts. The table or partition is subdivided into buckets based on the hash function of an attribute in the table to give extra structure to the data that may be used for more efficient queries. The number of buckets is determined by the hashfuction(bucketingcolumn) mod numofbuckets. Bucketing a table allows much more efficient sampling than the non-bucketed tables. With sampling, we can try out queries on a section of data for testing and debugging purpose when the original data set is very large.

The syntax is

```
create table tablename(columnname datatype ) partitioned by(partition datatype)
clustered by ()into buckets row format delimited field terminated by '\t';
```

3.10 INDEXING

An index is just a pointer on a particular attribute of a table. It acts as a reference to the records. Instead of searching all the records, we can refer to the index for searching a particular record. Indexes maintain the reference of the records and so it is easy to search for a record with minimum overhead. Indexes also speed up the searching of data. The major advantage is that whenever we perform a query on a table that has an index, there is no need for the query to scan all the tuples of the table. Further, it checks the index first and then goes to the particular attribute and performs the operation.

It is used as follows,

```
create index indexname on   table tablename(col,..) as 'index.handler.class.name'';
```

3.11 PRESENTATION

Now to the final stage a trial version of a data representation software 'Tableau' is used. The interface accepts tables in the form of csv or json files. So the views from Hive are exported into Tableau and the loaded data is interpreted graphically.

3.12 GRAPH DATABASES

Finally, for a different approach to execute certain queries faster a graph database is to be used. It requires cypher queries to get the results. The method of execution of the graph database is explained below.

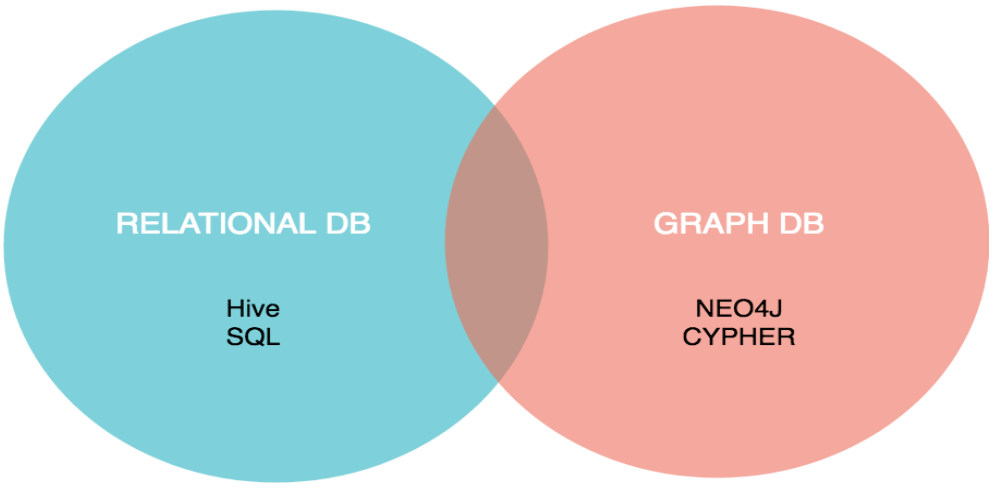

Figure 3.3 Functionality Overlay of database approaches

Relational database management is a type of management which stores data in the form of related data tables. These databases are more powerful than other databases because they require less number of assumptions on how the data is related in order to build relationships among different nodes in a database and also, how we extract the data from databases. As the result the same database can be viewed in many different ways. The main important feature of the relational database system is that a single database can be spread among several tables. Relational database, is a type of SQL database and which is being implemented by MongoDB,mySQL etc.

Graph database is also called a graph oriented database. Graph database is a type of NON_SQL database that uses graph concepts to store data into the database. The graph theory which is used to map the database is also used to query the relationship. A graph data is a collection of nodes and edges, where each node represents an entity (such as person, age and etc.) and each edge represents a relationship between two or more nodes. Each node and edge in a graph database is defined by a unique identifier. Graph databases are being used to build complex relationships among the nodes as shown in table 3.1.

3.13 COMPARING DATABASE APPROACHES

Table 3.1 Database Approaches

RELATIONAL DB	GRAPH DB
Relational database stores the data in the form of related tables.	Graph database stores the data in the form of a graph.
Relational databases are more powerful than other databases.	Graph database is not more powerful comparing to relational database.
Needs some assumptions on how the data is related in order to build relationships.	It does not need any assumption.
Relational database is a type of SQL database.	Graph database is not a type of SQL database, it is a noSQL database.
Relational database does not contain any nodes and edges.	Graph database contains nodes and edges.
It is implemented with the platform called MongoDB,mySQL.	It is implemented with the platform called Neo4j.

4 IMPLEMENTATION

4.1 INSTALLING HADOOP

As we used a machine running MacOS 10.13.4 the brew package manager was tried first.

```
brew install hadoop
```

The above command was run on the terminal line. But the version was not right. The Hadoop that got installed would have conflicts with the Hive framework that had to be installed. So Hadoop 2.7.6 was downloaded from the official Apache site (http://apache.website-solution.net/hadoop/common/hadoop-2.7.6/hadoop-2.7.6.tar.gz)

The above downloaded file was extracted into the admin home folder. And then the following changes were done to the files in the etc folder.

To the hadoop-env.sh which is located at /Users/admin/hadoop-2.7.6/etc/hadoop
The following line was added at the Java runtime options part.

```
export HADOOP_OPTS="$HADOOP_OPTS -Djava.net.preferIPv4Stack=true -Djava.security.krb5.realm= -Djava.security.krb5.kdc="
```

This was because the Java Development Kit had to be used by Hadoop without the out of heap memory error.

4.2 SETTING UP THE CLUSTERS

The core-site.xml was added with the lines below

```
<configuration>
  <property>
    <name>hadoop.tmp.dir</name>
    <value>/usr/local/Cellar/hadoop/hdfs/tmp</value>
    <description>A base for other temporary directories.</description>
  </property>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://localhost:9000</value>
  </property>
</configuration>
```

The above sets the hdfs location for the storage. It can also be accessed from the port 9000 of the local host.

The mapred-site.xml was also changed as below.

```
<configuration>
  <property>
    <name>mapred.job.tracker</name>
    <value>localhost:9010</value>
  </property>
</configuration>
```

This allows for the admin to change the mapreduce configuration.

The hdfs-site.xml was added the following lines.

```
<configuration>
   <property>
      <name>dfs.replication</name>
      <value></value>
   </property>
   <property>
      <name>dfs.namenode.name.dir</name>
      <value>/Users/admin/hadoop-2.7.6/data/dfs/namenode</value>
   </property>
   <property>
      <name>dfs.datanode.data.dir</name>
      <value>/Users/admin/hadoop-2.7.6/data/dfs/datanode</value>
   </property>
</configuration>
```

This sets up the namenode and the datanode.

The above facilitates for a single node with Hadoop cluster.

The machine we used had a zsh shell and so the Hadoop home path was inserted as follows.

```
export HADOOP_HOME="/Users/admin/hadoop-2.7.6"
export PATH="$HADOOP_HOME/bin:$PATH"
```

Then it was reloaded as the source with the command

```
source .zshrc
```

Hadoop also needs the ssh protocol already turned on on all the nodes so that they can access the datanodes. Also the remote login feature had to be turned on.

4.3 STARTING HADOOP

Now the Hadoop cluster may be started by using the command

/Users/admin/hadoop-2.7.6/sbin/start-all.sh

This starts up all the datanodes, secondary datanodes and the namenode.

The command jps is used to check if all the processes have started successfully as shown below.

Figure 4.1 Running Hadoop Processes

Also the overview port is available at http://localhost:50070/

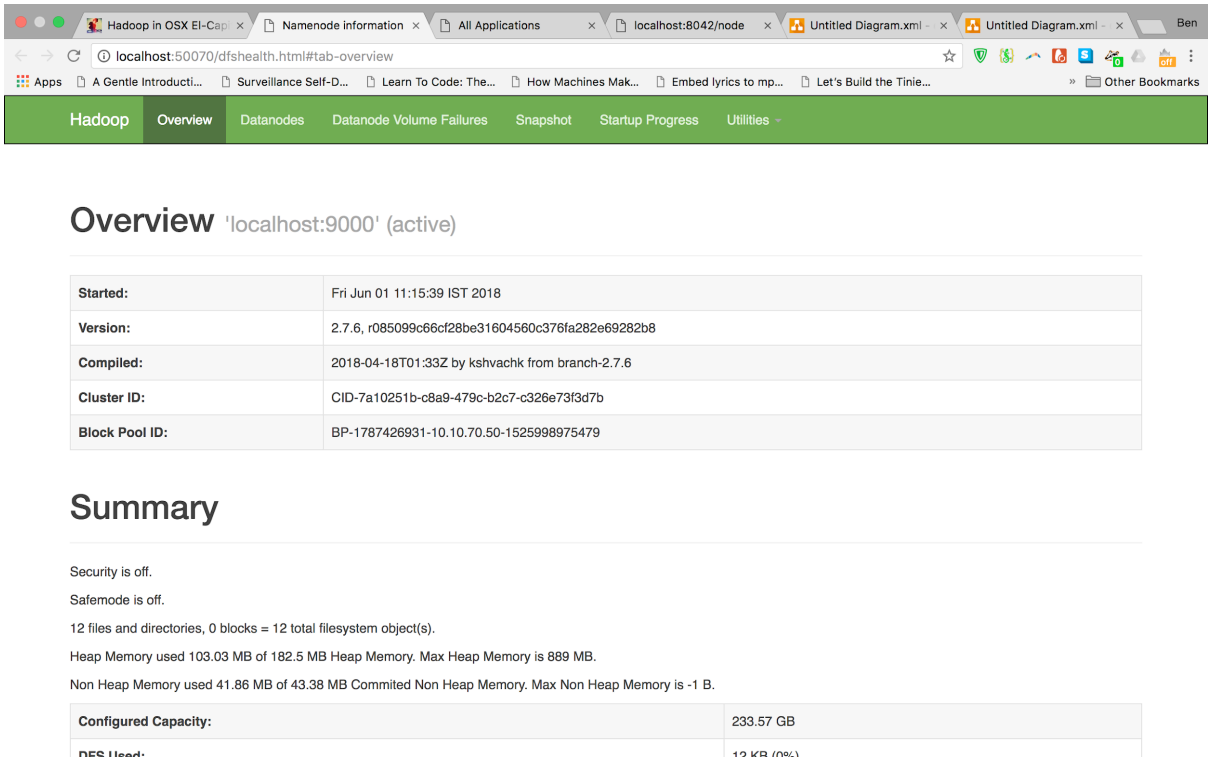

Figure 4.2 Overview port

The cluster details are provided at http://localhost:8088/

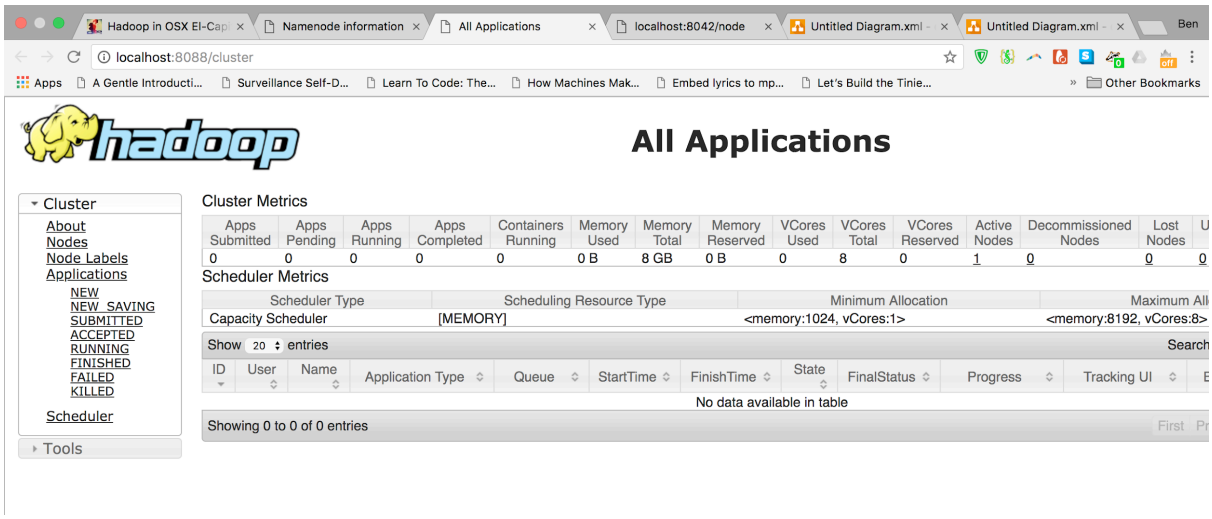

Figure 4.3 Cluster Application Overview

And the information of the nodes at http://localhost:8042/

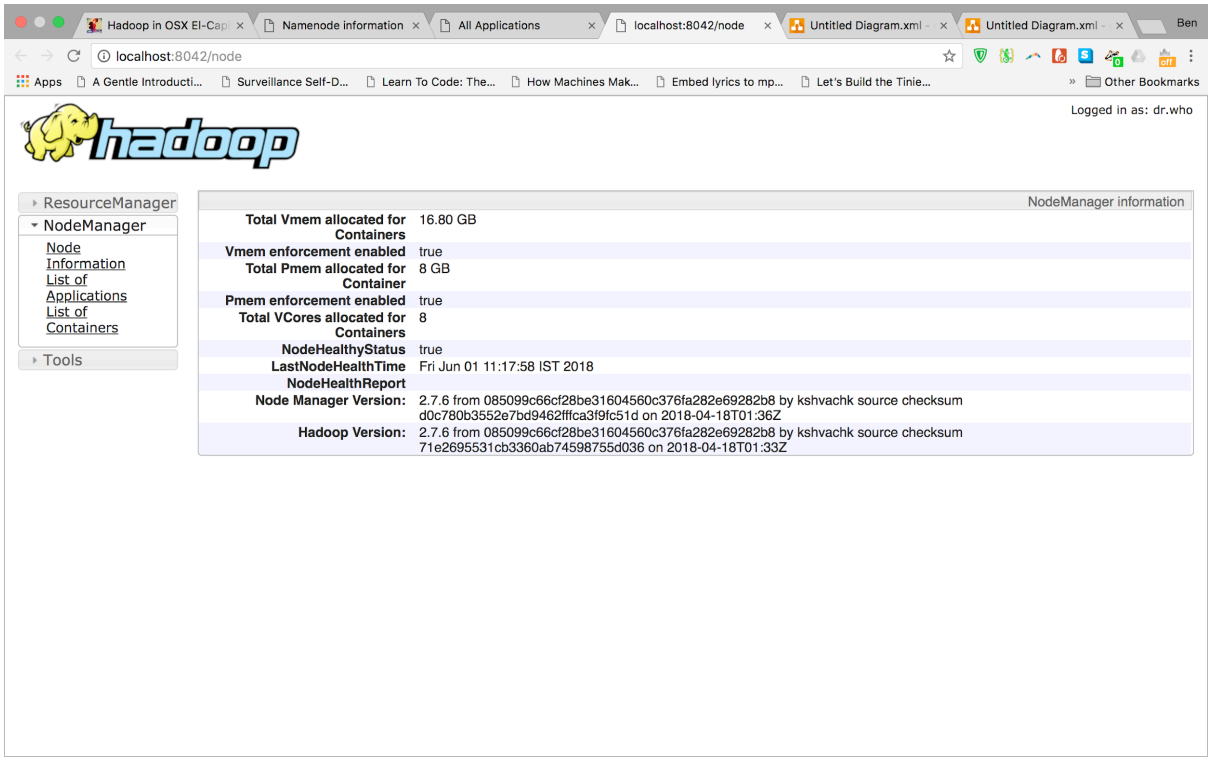

Figure 4.4 Node Overview

4.4 MAPREDUCE TASK

A mapreduce code to join two tables was implemented on using java to get a basic idea of how the data is being manipulated in the Hadoop cluster.
The following tables were joined using the Mapper method, Reducer method and the Extractor code snippets that have been attached in the appendix.

Table 4.1 Comments Table

ID	USERID	COMMENTS	POSTSHARED
1	1	looking awesome:)	http://dummyimage.com/160x166.gif/5fa2dd/ffffff
2	2	full masti	http://dummyimage.com/250x142.png/ff4444/ffffff
3	3	wow gangnam style,cool	http://dummyimage.com/124x173.png/cc0000/ffffff
4	4	welcome to the heaven	http://dummyimage.com/148x156.png/ff4444/ffffff

Table 4.2 Users Table

USERID	USERNAME	EMAILIID	SEX
1	Susan	smendoza0@angelfire.com	Female
2	Kathleen	knichols1@nsw.gov.au	Female
3	Marilyn	mclark2@washington.edu	Female
4	Craig	cscott3@is.gd	Male

Table 4.3 Users Table joined with the Comments Table

ID	USER ID	COMMENTS	POSTSHARED	USER NA-ME	EMAIL IID	SEX
1	1	looking awesome:)	http://dummyimage.com/160x166.gif/5fa2dd/ffffff	Susan	smendoza0@angelfire.com	Female
2	2	full masti	http://dummyimage.com/250x142.png/ff4444/ffffff	Kathleen	knichols1@nsw.gov.au	Female
3	3	wow gangnam style,cool	http://dummyimage.com/124x173.png/cc0000/ffffff	Marilyn	mclark2@washington.edu	Female
4	4	welcome to the heaven	http://dummyimage.com/148x156.png/ff4444/ffffff	Craig	cscott3@is.gd	Male

4.5 INSTALLING HIVE

Hive was then installed on the same machine to run queries with the SQL like interface. The installation was a bit tedious because the java virtual machines were not compatible for the stable version.

This was followed by downloading the Hive 2.3.3 file from the official Apache site http://apache.website-solution.net/hive/hive-2.3.3/apache-hive-2.3.3-bin.tar.gz

It was extracted into the home folder of admin.

Now for Hive to access the Hadoop storage space so they were created with the following commands

```
hdfs dfs -mkdir -p /user/hive/warehouse
```

```
hdfs dfs -chmod g+w /tmp
hdfs dfs -chmod g+w /user/hive/warehouse
```

Now in the Hive home directory the following commands were run

```
mkdir -p hcatalog/var/log
touch hcatalog/var/log/hcat.out
vim conf/hive-site.xml
```

The above commands create the log file to fix errors and also check the mapreduce tasks execution.

Then the configuration of the hive-site.xml is changed so that it accesses the Hadoop storage area.

4.6 SETTING UP STORAGE

Now hive is started by executing the Hive executable in the bin folder. In the environment that loads the following was put to set the mapred job tracker to local.

```
<configuration>
 <property>
  <name>hive.exec.scratchdir</name>
  <value>/tmp/hive-${user.name}</value>
  <description>Scratch space for Hive jobs</description>
 </property>
</configuration>
```

```
set mapred.job.tracker=local;
```

If the Hive environment does not start then initialise the metastore again with the command

```
schematool -initSchema -dbType derby
```

This fixes the metastore permissions and makes it stable. Derby database is used as the default but the ODBC drivers can be copied into the lib folder and then the metastore can be created in the database with permission for the user@'localhost'.

4.7 DATASET CONTENT

The StackOverFlow data dump from https://archive.org/details/stackexchange was used. The data dump had all the information about the users, the posts they have asked, answered, edited and all the comments the users had, the tags of the posts and the badges that the users gained by answering specific tagged posts.

4.8 IMPORTING DATASET

The jar file of xmlserde from
https://search.maven.org/remotecontent?filepath=com/ibm/spss/hive/serde2/xml/hivexmlserde/1.0.5.3/hivexmlserde-1.0.5.3.jar
is loaded into the lib folder of Hive. This allows the files of the xml datatype to be loaded into the database.

Now the data schema is created for all the entities using the SQL like syntax. Down below is for one such entity 'user' which will be loaded into the users_xml table.

```
CREATE TABLE users_xml(
id INT,
reputation INT,
creationdate STRING,
displayname STRING,
lastaccessdate STRING,
websiteurl STRING,
location STRING,
aboutme STRING,
views INT,
upvotes INT,
downvotes INT,
accountid INT)
ROW FORMAT SERDE 'com.ibm.spss.hive.serde2.xml.XmlSerDe'
WITH SERDEPROPERTIES (
"column.xpath.id"="/row/@Id",
"column.xpath.reputation"="/row/@Reputation",
"column.xpath.creationdate"="/row/@CreationDate",
```

```
"column.xpath.displayname"="/row/@DisplayName",
"column.xpath.lastaccessdate"="/row/@LastAccessDate",
"column.xpath.websiteurl"="/row/@WebsiteUrl",
"column.xpath.location"="/row/@Location",
"column.xpath.aboutme"="/row/@AboutMe",
"column.xpath.views"="/row/@Views",
"column.xpath.upvotes"="/row/@UpVotes",
"column.xpath.downvotes"="/row/@DownVotes",
"column.xpath.accountid"="/row/@AccountId"
)
STORED AS
INPUTFORMAT 'com.ibm.spss.hive.serde2.xml.XmlInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.IgnoreKeyTextOutputFormat'
TBLPROPERTIES (
"xmlinput.start"="<row Id",
"xmlinput.end"="/>"
);
```

The above creates the table. Now to load the data into the table the following query statement is used.

```
LOAD DATA LOCAL INPATH 'Users.xml' INTO TABLE users_xml;
```

The inpath refers to the path of the $HIVE_HOME or by default to the user's home folder.

4.9 QUERYING

Twenty queries were run on the database schema which got converted into mapreduce jobs and executed.

The HiveQL was used to frame the queries that were run on Hive. The syntax is similar to that of any SQL based database.

Table 4.4 Executed Queries

Most popular tag of all posts	select tagname from tags_xml where count in (select max(count) from tags_xml);
Number of posts who have answers but have not been accepted	select id from posts_xml where answercount > 0 and acceptedanswerid is NULL;
Number of users who have not accepted any answer for any of their posts	select owneruserid,count(1) from posts_xml where acceptedanswerid is NULL group by owneruserid;
Number of upvotes, downvotes and moderation counts each user has	select postid,votetypeid,count(1) from votes_xml group by postid,votetypeid;
The average score and viewcount for a post that has not been solved	select avg(score),avg(viewcount) from posts_xml where acceptedanswerid is not NULL;
User having the highest reputation in the forum	select displayname,id from users_xml where reputation in (select max(reputation) from users_xml);
The unanswered post with the highest view count	select * from posts_xml where acceptedanswerid is NULL and viewcount=1716347;
Posts that had asked about the AirPort	select * from posts_xml where body like '%AirPort%';
The users who had more downvotes than upvotes	select displayname from users_xml where downvotes > upvotes;
Number of posts related to macos or intel and has not been accepted	select count(1) from posts_xml where tags like '%macos%' or tags like '%intel%' and acceptedanswerid is not NULL;

The post which has the most relations with other posts	select relatedpostid,count(1) as temp from postlinks_xml group by relatedpostid sort by temp desc limit1;
The list of users based on their location	select location,count(1) from users_xml group by location;
The post with the most revisions	select postid,count(1) as temp from posthistory_xml group by postid sort by temp desc limit 1;
Join the users with their badges	select users_xml.name,badges_xml.badge from users_xml join badges_xml on users_xml.userid=badges_xml.userid;

A query to find the most popular tag

select tagname from tags_xml where count in (select max(count) from tags_xml);

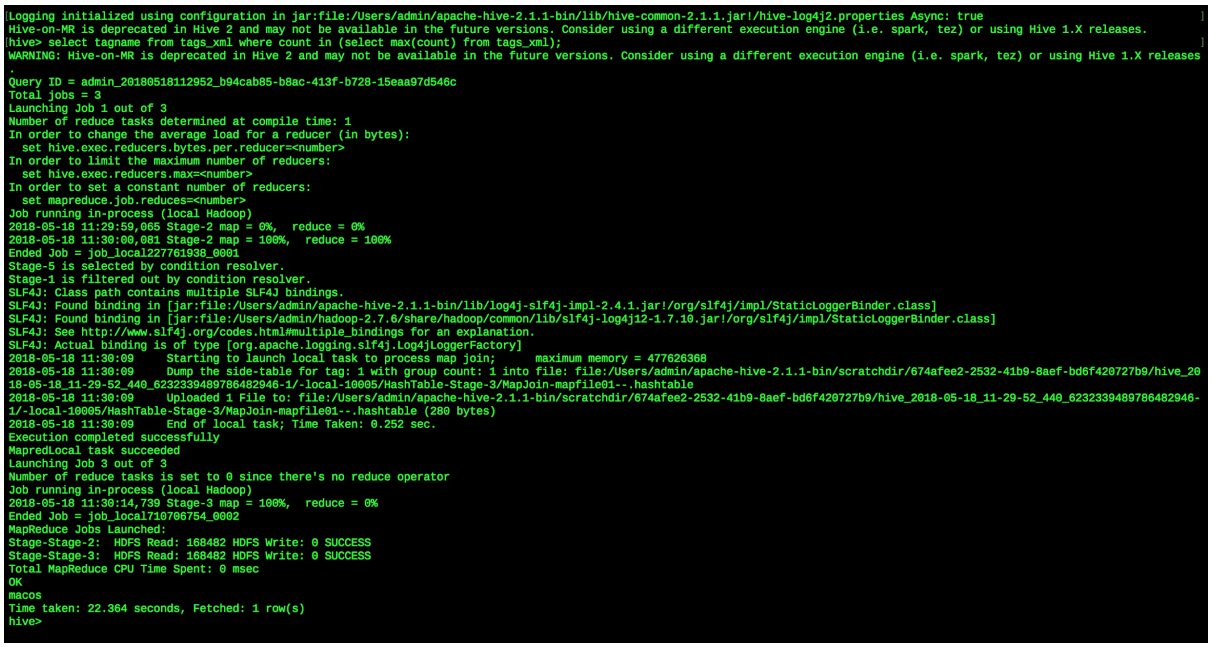

Figure 4.5 Execution of query to find the most popular tag

Another query to find the most reputed user

select displayname,id from users_xml where reputation in (select max(reputation) from users_xml

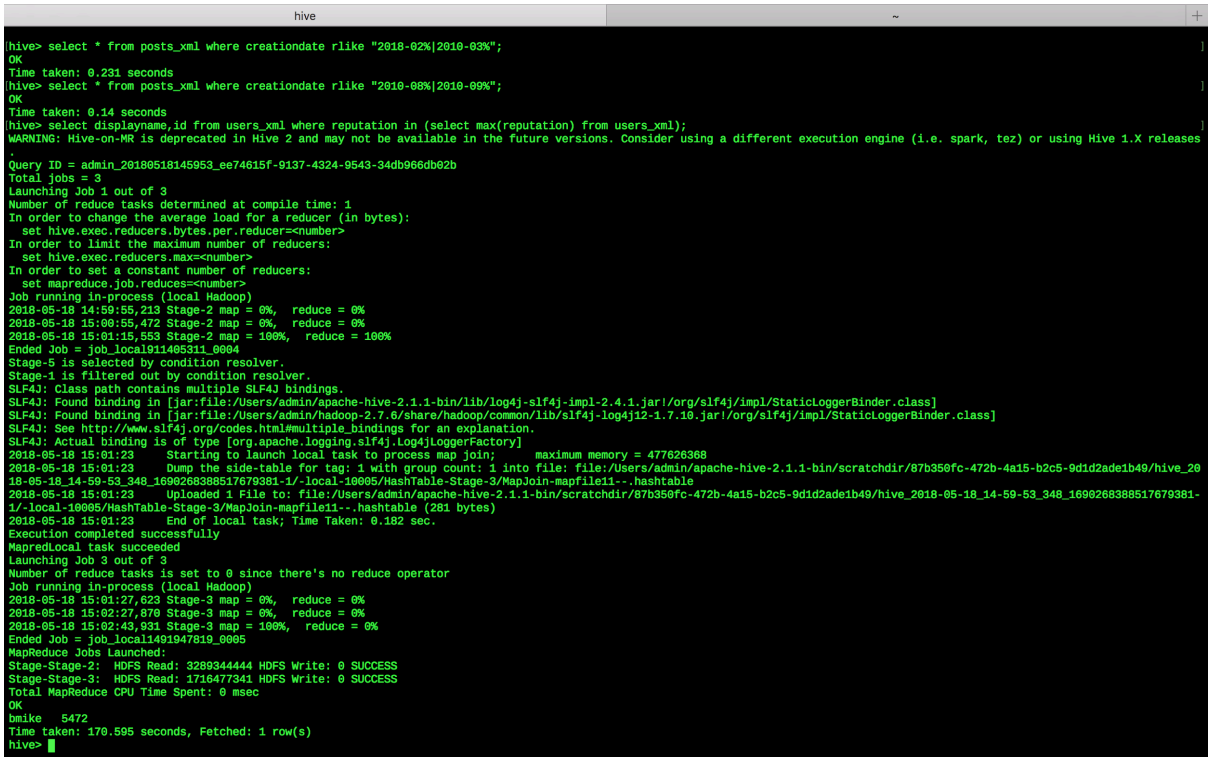

Figure 4.6 Execution of query to find the most reputed user

Finally one more,

The average score and view count for an unanswered question

select avg(score),avg(viewcount) from posts_xml where acceptedanswerid is not NULL;

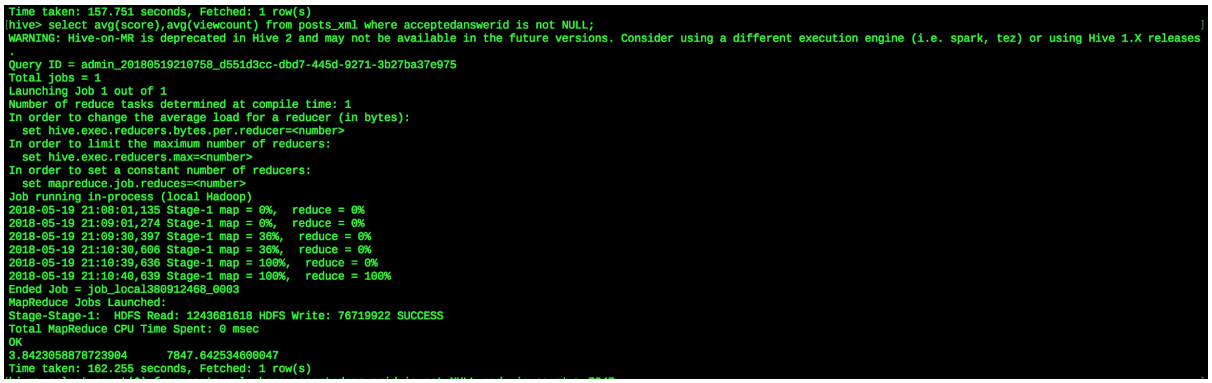

Figure 4.7 Execution of query to find stats of an unanswered post

Indexing the tables based on the attribute and then rerunning the same queries resulted in faster execution times.

To index the posts_xml table the SQL statement below is used.

```
CREATE INDEX posts_xml_index
ON TABLE posts_xml (postid)
AS 'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler'
WITH DEFERRED REBUILD;
```

Now to rebuild the table with the index the following query is used

```
ALTER INDEX posts_xml_index on posts_xml REBUILD;
```

Next the tables are bucketed based on the values of the attributes using a hash function to match a query and then the queries were rerun. But this time the execution times were almost similar to that of the indexed run time.

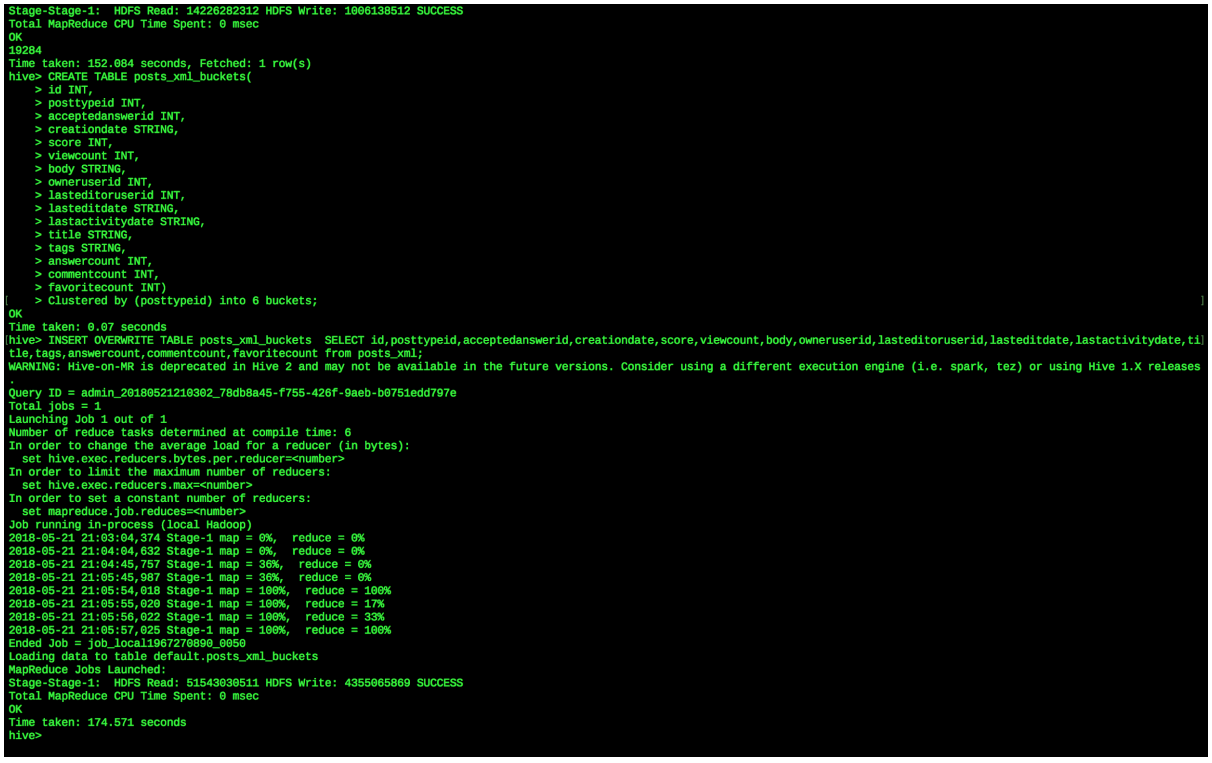

Figure 4.8 Bucketing a table of the database

For partitioning the following have to be set in the Hive environment which is done using

```
set hive.exec.dynamic.partition.mode=nonstrict
```

Now the tables are overwritten with the original values as shown in the screenshot in figure 4.9

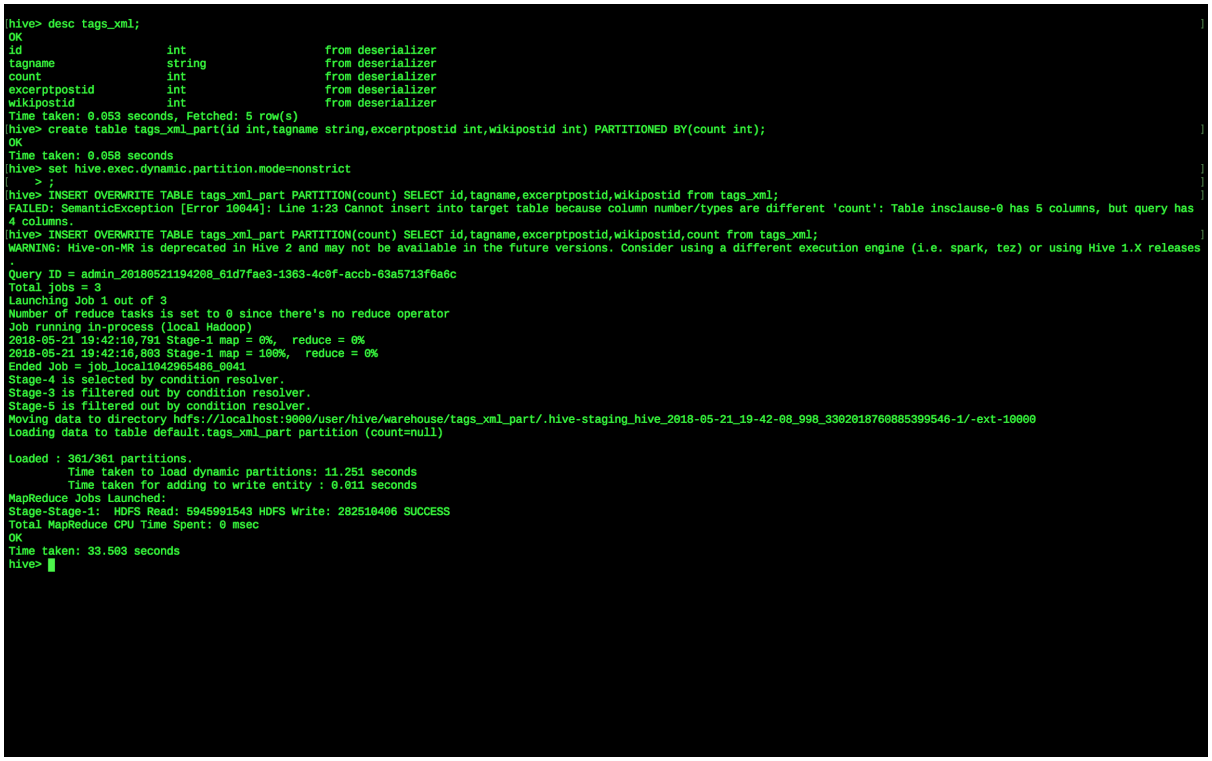

Figure 4.9 Partitioning the tables

To make the Hive execution a bit faster the following were done with the Hive environment.

```
set hive.vectorized.execution.enabled = true;
set hive.vectorized.execution.reduce.enabled = true;
set hive.cbo.enable=true;
set hive.compute.query.using.stats=true;
set hive.stats.fetch.column.stats=true;
set hive.stats.fetch.partition.stats=true;
```

4.10 ANALYSIS

The graph chart below gives the execution times with and without optimization.

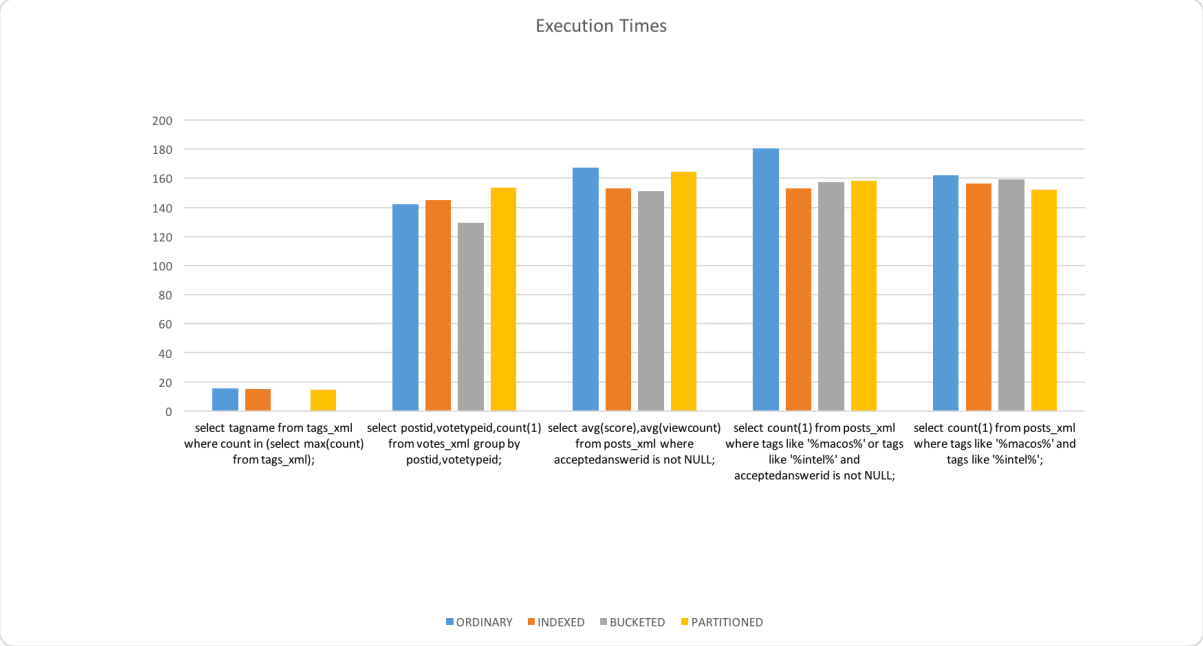

Figure 4.10 Execution Times of Queries

The execution time was given by the Hive engine was into the Tableau worksheet to be form the graphical interpretations. The time was in seconds with three significant digits and so the accuracy was to the millisecond standard. The select all records from a table performed way too faster than those with the where clauses in them.

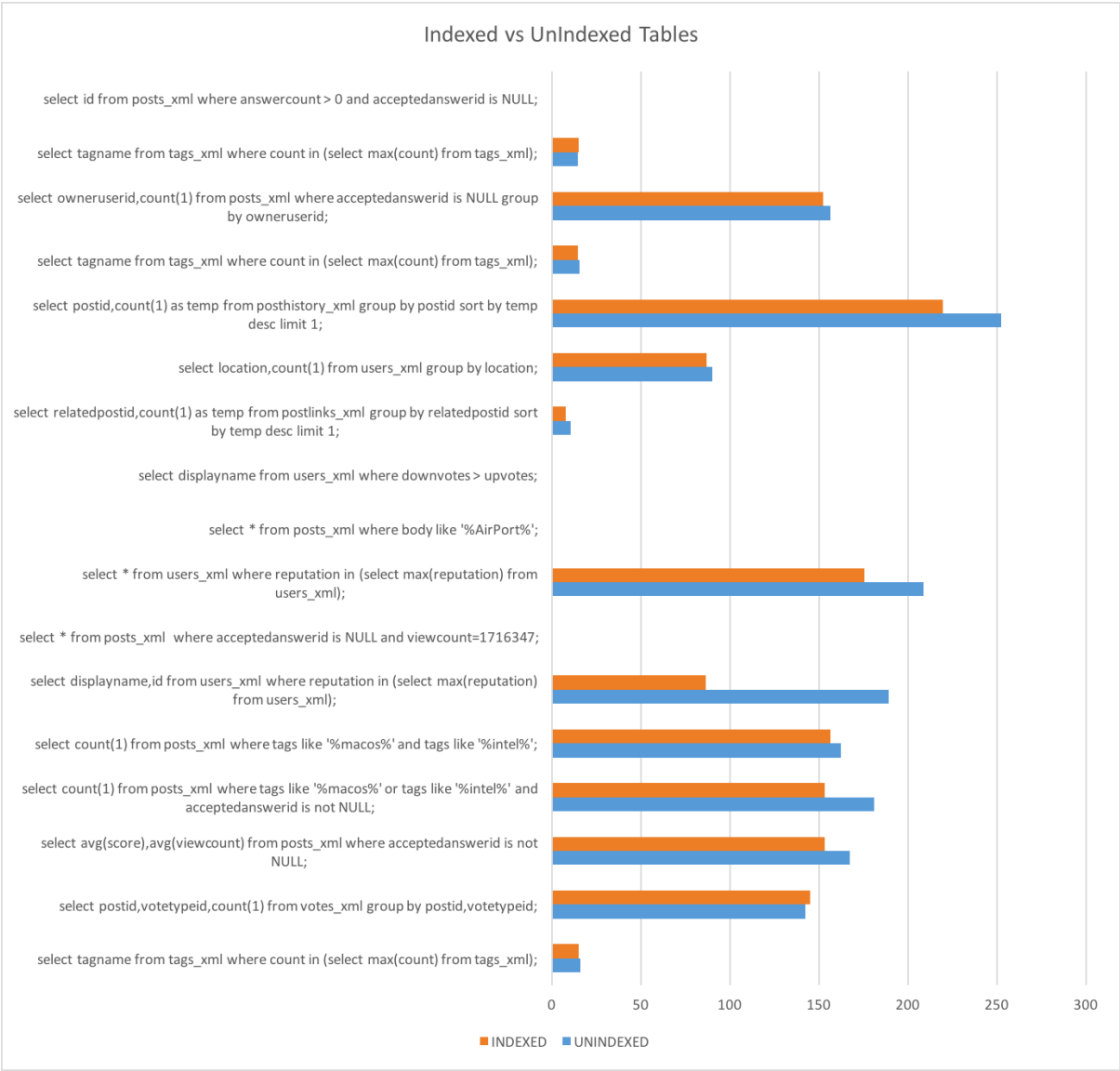

Figure 4.11 Execution times of queries when tables are indexed and unindexed respectively

The trial version of Tableau was used to represent the data in the above form. For the above the query execution times were imported as an excel file into the software.

The conclusions that can be arrived at from running these queries is that indexing is by far the best optimization method providing an average of 30 percentage reduction in the execution time of the queries. Bucketing may be used when the joins are used but they can be used for other types of queries too. Partitioning can be done as per the case involved both in the vertical and the horizontal sense. What was also obvious was that with bucketing or partition care must be taken to use attributes that form a manageable number of buckets or partitions, else the execution times become longer.

Tableau also had the interface for location data interpretation which was used as below.

Figure 4.12 Loactions where the users registered

The above is the image representing the cities where the users had registered their accounts. For this the query was run to create a view which was then exported from the Hadoop storage into Tableau.

4.11 INSTALLATION OF NEO4J

Neo4j was installed on a machine running Windows10. The Neo4j file was downloaded from the official site of Neo4j on https://neo4j.com/download/
Neo4j needed Java-8 for its class files and so the necessary Java Development Kit was installed and configured appropriately.
The Wikipedia dump was downloaded from the internet and the data dump was loaded into Neo4j. So after importing data, Neo4j gives those data in the form of graph. Hence it is also called graph database

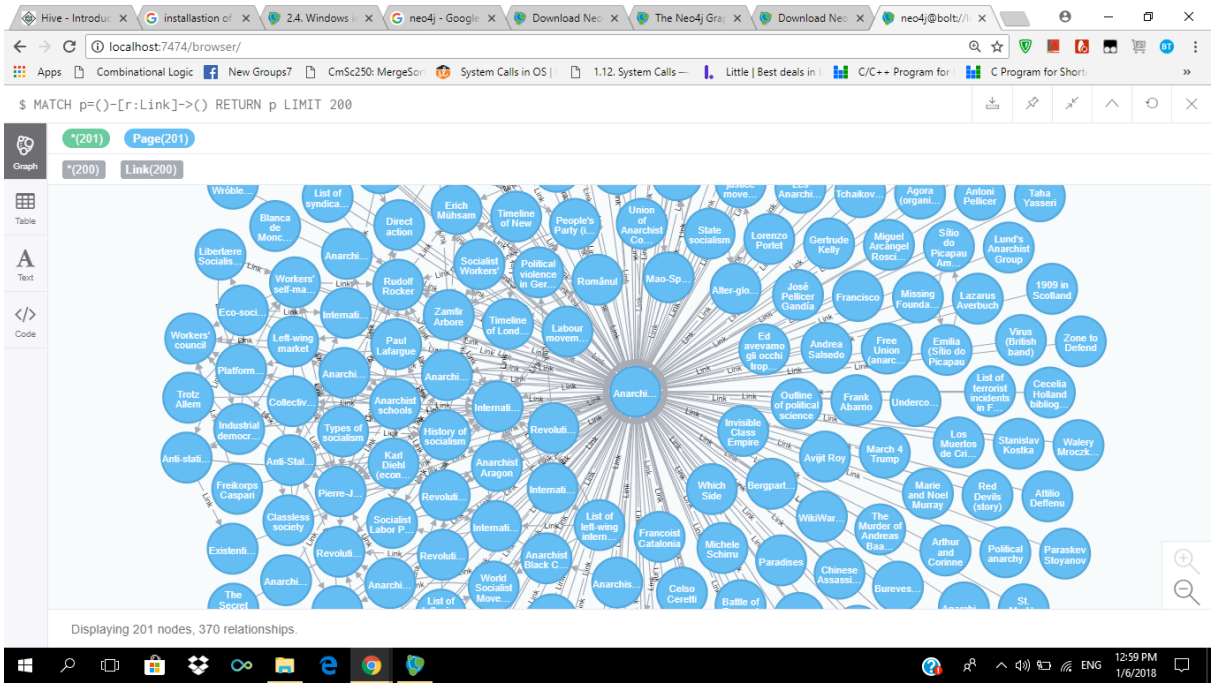

Figure 4.13 Nodes in the Graph database using Neo4j

After importing the data into Neo4j the next task is to optimize the data. It can be done with query. we were executing some queries for data optimization those are

- Finding pagerank for data stored in neo4j
- Finding betweenness centrality
- Indexing

Before executing the queries we have installed the algo.file for fast querying

Query to create the page rank

```
CALL algo.pageRank(null,null,{write:true,writeProperty:'pagerank_g'})
```

Query to display the page rank

```
MATCH (e:Entity) WHERE exists(e.pagerank_g)
RETURN e.name AS entity, e.jurisdiction_description AS jurisdiction,
e.pagerank_g AS pagerank ORDER BY pagerank DESC LIMIT 15
```

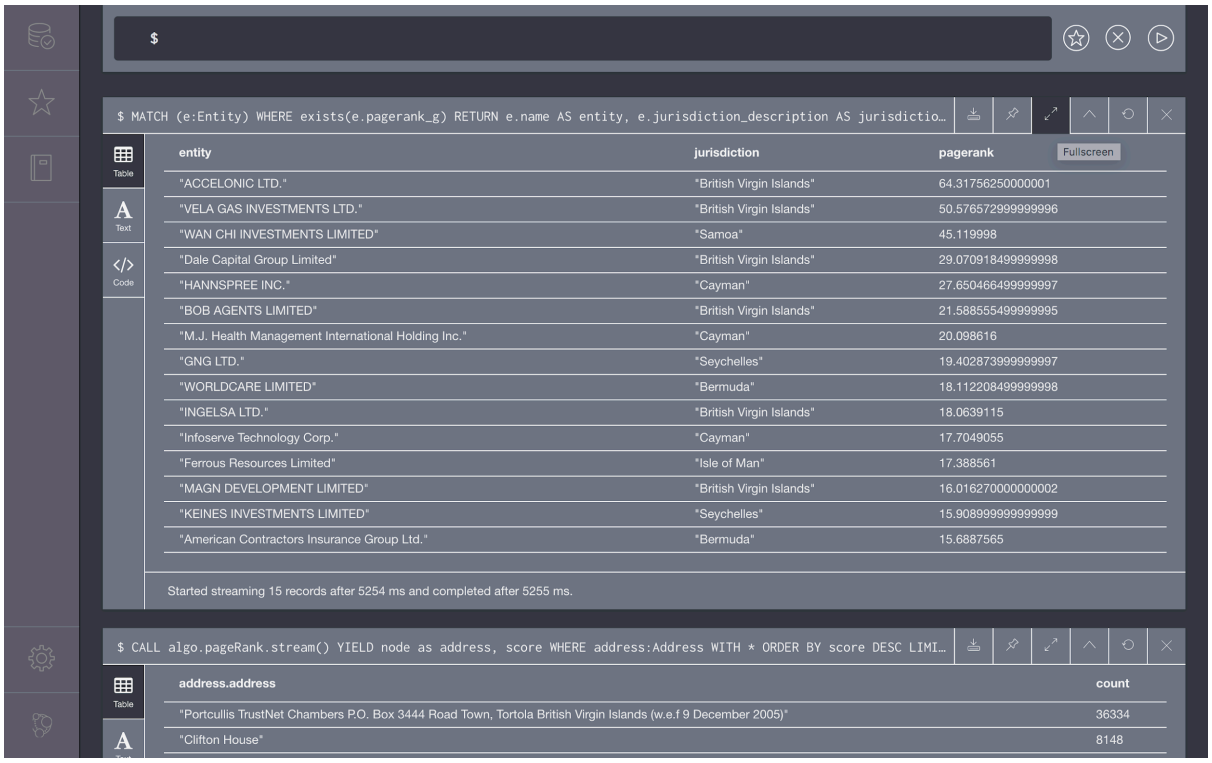

Figure 4.14 Displaying the pagerank results of the database

The figure 4.14 shows the page rank for the imported data.

Before moving to betweenness centrality, let us make it clear that it works under the concept of allshortestpaths.

Query for betweenness centrality

```
MATCH p=allShortestPaths((rex:Officer)-[*]-(queen:Officer))
WHERE rex.name = "Tillerson - Rex" AND queen.name = "The Duchy of Lancaster"
RETURN p
```

Output is as shown in figure 4.15

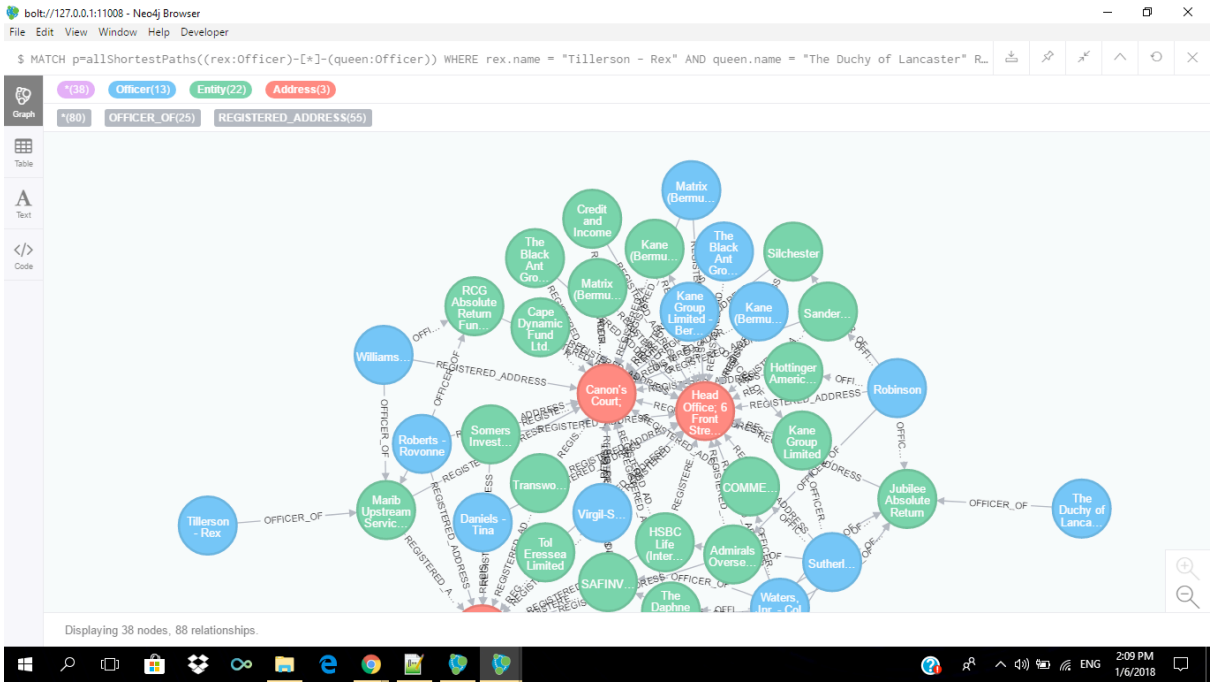

Figure 4.15 Betweeness between two nodes in the database

Before executing the indexing, we should create index which is given below
Query for creating index

```
CREATE INDEX ON :Person(first_name);
```

Query run after indexing

```
MATCH (p:Person {first_name: 'Wilbur'}) RETURN p
```

Output is as shown in figure 4.16

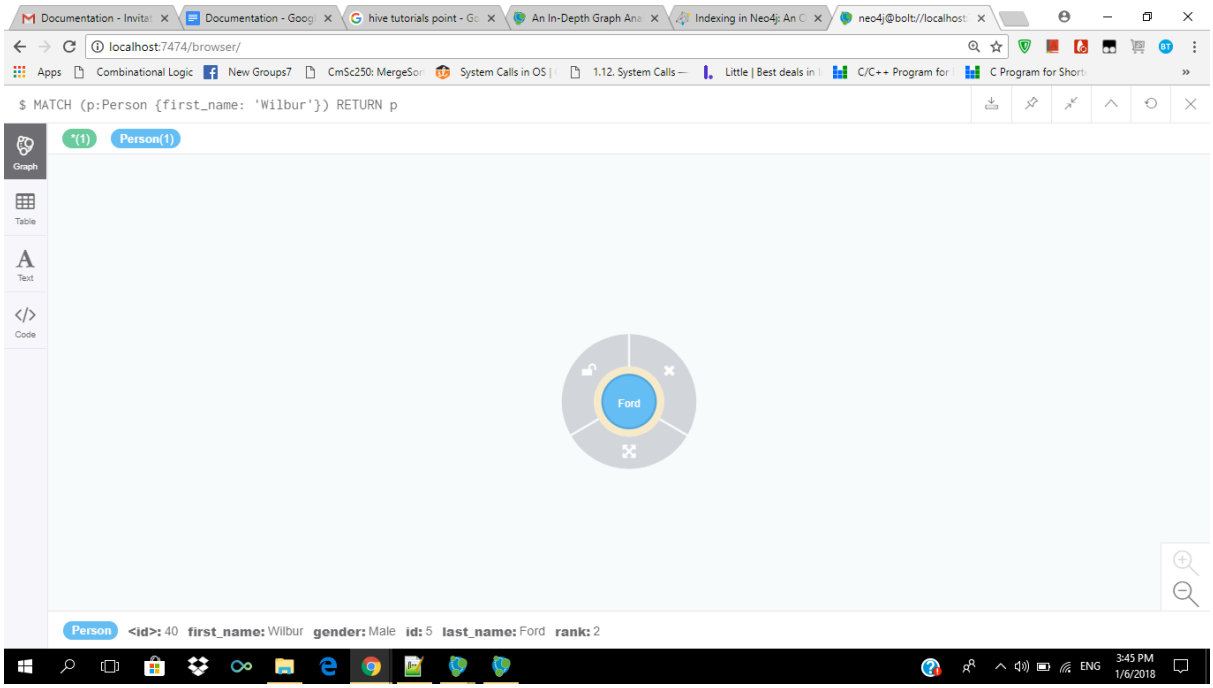

Figure 4.16 Running a Cypher Query after indexing

5 CONCLUSION

With the datadump being loaded into the Hadoop single node cluster and performing mapreduce tasks and also having the Hive architecture to use a simpler SQL interface the query optimization techniques were tried out. Out of all the methods the one that is universal is to index based on a related attribute on the table before any query is executed. Also the features built into the execution engines may be used to the fullest extent. This project can be taken further down the development lane with support for real-time data being crawled into the database and in such a case bucketing becomes more realistic when you know the queries before hand especially the ones involving join operations. Finally, when the datadump becomes very large then the partitioning both vertical and horizontal provide justice for the execution time.

REFERENCES

1. For Hadoop installation
   https://dtflaneur.wordpress.com/2015/10/02/installing-hadoop-on-mac-osx-el-capitan/
2. For Hive installation
   https://cwiki.apache.org/confluence/display/Hive/GettingStarted
3. For importing XML files
   https://github.com/dvasilen/Hive-XML-SerDe/wiki/XML-data-sources
4. For query optimization
   http://hadooptutorial.info/partitioning-in-hive/
5. Mapreduce in hive
   https://www.dezyre.com/article/mapreduce-vs-pig-vs-hive/163
6. Neo4j downloads and installations
   https://neo4j.com/download/
7. Neo4j sample cypher
   https://neo4j.com/blog/depth-graph-analysis-paradise-papers

APPENDIX

Mapper Method

```
package com.javamakeuse.bd.poc.mapper;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import com.javamakeuse.bd.poc.vo.UserActivityVO;
public class UserActivityMapper extends Mapper<LongWritable, Text, IntWritable, UserActivityVO> {
   // user map to keep the userId-userName
   private Map<Integer, String> userMap = new HashMap<>();
   @Override
   protected void map(LongWritable key, Text value,
         Mapper<LongWritable, Text, IntWritable, UserActivityVO>.Context context)
               throws IOException, InterruptedException {
      String[] columns = value.toString().split("\t");
      if (columns != null && columns.length > 2) {
         UserActivityVO userActivityVO = new UserActivityVO();
         userActivityVO.setUserId(Integer.parseInt(columns[1]));
         userActivityVO.setComments(columns[2]);
         userActivityVO.setPostShared(columns[3]);
         userActivityVO.setUserName(userMap.get(userActivityVO.getUserId()));
         // writing into context
         context.write(new IntWritable(userActivityVO.getUserId()), userActivityVO);
      }
   }
   @Override
   protected void setup(Mapper<LongWritable, Text, IntWritable, UserActivityVO>.Context context)
         throws IOException, InterruptedException {
      // loading user map in context
```

```
        loadUserInMemory(context);
    }

    private void loadUserInMemory(Mapper<LongWritable, Text, IntWritable, UserActivityVO>.Context context) {
        // user.log is in distributed cache
        try (BufferedReader br = new BufferedReader(new FileReader("user.log"))) {
            String line;
            while ((line = br.readLine()) != null) {
                String columns[] = line.split("\t");
                userMap.put(Integer.parseInt(columns[0]), columns[1]);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Reducer Method

```
package com.javamakeuse.bd.poc.reducer;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.mapreduce.Reducer;
import com.javamakeuse.bd.poc.vo.UserActivityVO;
public class UserActivityReducer extends Reducer<IntWritable, UserActivityVO, UserActivityVO, NullWritable> {
    NullWritable value = NullWritable.get();
    @Override
    protected void reduce(IntWritable key, Iterable<UserActivityVO> values,
            Reducer<IntWritable, UserActivityVO, UserActivityVO, NullWritable>.Context context)
                    throws IOException, InterruptedException {
        for (UserActivityVO userActivityVO : values) {
            context.write(userActivityVO, value);
        }
    }
}
```

Extractor Method

```
package com.javamakeuse.bd.poc.vo;
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import org.apache.hadoop.io.Writable;
public class UserActivityVO implements Writable {
   private int userId;
   private String userName;
   private String comments;
   private String postShared;
   public int getUserId() {
      return userId;
   }
   public void setUserId(int userId) {
      this.userId = userId;
   }
   public String getUserName() {
      return userName;
   }
   public void setUserName(String userName) {
      this.userName = userName;
   }
   public String getComments() {
      return comments;
   }
   public void setComments(String comments) {
      this.comments = comments;
   }
   public String getPostShared() {
      return postShared;
   }
   public void setPostShared(String postShared) {
      this.postShared = postShared;
   }
   @Override
   public void write(DataOutput out) throws IOException {
      out.writeInt(userId);
```

```
        out.writeUTF(userName);
        out.writeUTF(comments);
        out.writeUTF(postShared);
    }
    @Override
    public void readFields(DataInput in) throws IOException {
        userId = in.readInt();
        userName = in.readUTF();
        comments = in.readUTF();
        postShared = in.readUTF();
    }
    @Override
    public String toString() {
        return "UserActivityVO [userId=" + userId + ", userName=" + userName + ", comments=" + comments
                + ", postShared=" + postShared + "]";
    }
}
```

Main Method

```
package com.javamakeuse.bd.poc.driver;
import java.net.URI;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
import com.javamakeuse.bd.poc.mapper.UserActivityMapper;
import com.javamakeuse.bd.poc.reducer.UserActivityReducer;
import com.javamakeuse.bd.poc.vo.UserActivityVO;
public class UserActivityDriver extends Configured implements Tool {
```

```
    public static void main(String[] args) {
        try {
            int status = ToolRunner.run(new UserActivityDriver(), args);
            System.exit(status);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public int run(String[] args) throws Exception {
        if (args.length != 2) {
            System.err.printf("Usage: %s [generic options] <input1> <output>\n", getClass().getSimpleName());
            ToolRunner.printGenericCommandUsage(System.err);
            return -1;
        }
        Job job = Job.getInstance();
        job.setJarByClass(getClass());
        job.setJobName("MapSideJoin Example");
        // input path
        FileInputFormat.addInputPath(job, new Path(args[0]));
        // output path
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        job.setMapperClass(UserActivityMapper.class);
        job.setReducerClass(UserActivityReducer.class);
        job.setMapOutputKeyClass(IntWritable.class);
        job.setMapOutputValueClass(UserActivityVO.class);
        job.addCacheFile(new URI("hdfs://localhost:9000/input/user.log"));
        job.setOutputKeyClass(UserActivityVO.class);
        job.setOutputValueClass(NullWritable.class);
        return job.waitForCompletion(true) ? 0 : 1;
    }
}
```