TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

PULCHOWK CAMPUS

THESIS NO: 072MSCS664

A Comparative Analysis of Cloud based Recommendation System on Mapreduce and Spark

by

Sarala Ghimire

A THESIS

SUBMITTED TO THE DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING IN PARTIAL FULFULLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE IN COMPUTER SYSTEM AND KNOWLEDGE ENGINEERING

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING

LALITPUR, NEPAL

NOV, 2017

A Comparative Analysis of Cloud based Recommendation System on Mapreduce and Spark

by

Sarala Ghimire

072MSCS664

Thesis Supervisor

Prof. Dr. Subarna Shakya

A thesis submitted in partial fulfillment of the requirements for the Degree of Master of Science in Computer System and Knowledge Engineering

Department of Electronics and Computer Engineering

Institute of Engineering, Pulchowk Campus

Tribhuvan University

Lalitpur, Nepal

Nov, 2017

COPYRIGHT ©

TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

PULCHOWK CAMPUS

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING

The undersigned certify that they have read and recommended to the Department of Electronics and Computer Engineering for acceptance, a thesis entitled "**A Comparative Analysis of Cloud based Recommendation System on Mapreduce and Spark**", submitted by **Sarala Ghimire** in partial fulfillment of the requirement for the award of the degree of "**Master of Science in Computer System and Knowledge Engineering**".

…..…….
Prof. Dr. Subarna Shakya
Supervisor
Department of Electronics and Computer Engineering
Institute of Engineering, Pulchowk Campus

……
Om Bikram Thapa
External Examiner
Vianet Communication Pvt.Ltd

…..……..
Prof. Dr. Subarna Shakya
Committee Chairperson
Department of Electronics and Computer Engineering
Institute of Engineering, Pulchowk Campus

……
Date:

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to my supervisor Prof. Dr. Subarna Shakya for his encouragement and precious guidance during my thesis research. I am thankful to our program coordinator Dr. Aman Shakya for providing a suitable platform to make this research success. I would like to show my appreciations to all my professors, classmates and faculty member of Department of Electronics and Computer Engineering for providing me their views and ideas regarding thesis work.

ABSTRACT

Today, Big Data is a hot issue both in industrial and academic fields. The need of data processing is changing with the gradual increase in data volume and with the mass of sources leading to a diversity of structures. Although relational database management system (RDBMS) remaining the primary technology for data management of structured data and been proven best for more than 40 years, it has reached its limit, and the reason is massive growth in the diverged volume of data. Several researchers and organizations now focused on MapReduce and Spark framework that has discovered huge success in processing and analyzing a large volume of data on several clusters. In this study, the performance of MapReduce, RDBMS, and Spark with various comparison measures are evaluated. To conduct a comparison and analysis, three processes are computed: (a) developed recommendation system with all three algorithms, (b) run that system on various data networks and data sizes, and (c) the output is then analyzed and compared on the basis of time computation, memory consumption, and CPU usage. Moreover, statistical validation of the observed results from all the algorithms with respective node and network configuration using Friedman rank test and Holm post-hoc test are performed. Overall, observations show that Spark is about 2.5x and 5x faster than MapReduce, and 10/20 times faster than RDBMS. The reason for these speedups is the efficiency of the alternative least square algorithm and reduced CPU and disk overheads due to RDD caching in spark.

Keywords: Cloud Computing, Hadoop, MapReduce, Multi-node cluster, RDBMS, Spark, and Single-node cluster

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 Background Theory

Big Data is a characterization of the huge volume of various data type, mostly unstructured [1]. This entails data, which is too vast and huge that relational database management systems will not be able to analyze, because of its size, volume and unstructured in nature. So there should be a tool that is capable of making use of data fusing from various sources in the best way to generate value that can create better financial output for the company and better experiences for the end user and the customers. The tool that can be considered as effective if it can provide higher efficiency with limited resources. Big Data Analytics is that tool which provides precise solutions to analysts and researchers making use of the huge volume of previously unknown raw and unusable data. Using such analytics along with various data mining, machine learning, and natural language processing techniques, it is easier to find valuable and convenient insight, which aids enterprises and business to make the right decision at the right time. Open source technology like Hadoop/Mapreduce and Spark provides an effective solution for Big Data Analytics. In this study comparison of recommendations by RDBMS, Spark and Mapreduce framework on Hadoop Distributed File System are studied.

Recommendation systems are part of information filtering system, which predicts the preference; the user might give to an item. Eliminating the static experience that needs searching for static information for purchasing any products, recommendation systems have generated a new experience to the user interacting with the websites by collective interaction among users dynamically. These systems calculate recommendations for each user based on their past experiences, searches and other users' preferences and search behaviors. Unlike search engines, recommendation engines try to present people with relevant content that they did not necessarily search for or that they might not even have heard of. Typically, a recommendation engine tries to model the connections between users and some type of item. Even if people do not know exactly what a recommendation engine is, they have most likely experienced one through the use of popular websites such as Facebook, Twitter, LinkedIn, and Amazon, etc. Recommendation systems can be extremely effective on a large scale if they are

implemented correctly. These systems are a core part of all these businesses, and in some cases, they drive significant percentages of their revenue.

Collaborative filtering is one of the most popular and successful algorithmic approaches of recommendation algorithms. It is used in many websites and recognized as the most successful recommendation systems. The system calculates predicted preferences of users for items with which they have not yet interacted with using the set of preferences of many other users concerning items. So the algorithm is based on the notion of similarity. There are two popular approaches to perform this similarity, user and item based, and are referred to as nearest-neighbor models since the predicted values are calculated based on the set of most similar users or items.

In a user-based approach, two users are considered similar if they have similar preferences and taste, that is, interacting pattern matched for the same item. So preference of another user that has a similar interacting pattern with target user can be used to calculate recommendations for unknown items. This can be done by selecting a set of similar users, known as neighbor formation, and calculating the score based on the items they have shown a preference for. The overall logic is that if others have tastes similar to a set of items, these items will tend to be good candidates for recommendation.

In an item-based approach, the similarity between two items is calculated. This is usually based on the existing user-item preferences or ratings. Items that tend to be rated the same by similar users will be classed as similar under this approach. Once getting these similarities, it can represent a user regarding the items they have interacted with and find items that are similar to these known items, which can then recommend to the user. Again, a set of items similar to the known items is used to generate a combined score to estimate for an unknown item.

The goal of collaborative filtering algorithms is to either make suggestions about new items or to make a prediction about the acceptance of a certain item for recommendations when providing opinions about various items. Also, it even

aims to either make suggestions of new items or to make a prediction about the acceptance of a certain item for a particular user based on users past experiences and similarity with others users.

Moreover, Big Data analytics using data mining algorithms possess high computing requirements, which require high-performance processors to accomplish the task. The cloud provides a good platform for big data storage, processing, and analysis, addressing two of the main requirements of big data analytics, high storage, and high-performance computing [2].

The cloud-computing environment offers development, installation, and implementation of software and data applications ‘as a service.' Three services that exist are, namely, platform as a service (PaaS), software as a service (SaaS), and infrastructure as a service (IaaS). Infrastructure-as-a-service is a model that provides computing and storage resources as a service. Similarly, PaaS provides a software platform as a service whereas SaaS provides software itself to its clients. Also, the notion of commodity hardware and the ‘pay-as-you-go’ model creates an efficient way of processing of huge volume data in a timely fashion, giving the conception of ‘big data as a service’ justice. Google Cloud Dataproc can be taken as an example, which provides real-time vision in a cloud environment for big data.

A relational database management system (RDBMS) is a database management system (DBMS) that is based on a relational model in which data and relationship among the data is stored in the form of tables. Relational databases are powerful in the sense that they require few assumptions about how data is related or how it will be extracted from the database. As a result, the same database can be viewed in many different ways. An important feature of relational systems is that a single database can be spread across several tables. These are used to store information like financial records, personal data, manufacturing information and other applications [3]. Despite receiving a challenge by object-oriented database system and XML database management system, RDBMS possess most of the market. Nearly all full-sized database systems are RDBMS's.

However, this traditional data management tools cannot be used for Big Data Analytics for the large volume and complexity of the datasets because of its limited capacity to support variety and volume of data.

1.2 Hadoop Mapreduce Framework

Hadoop is an open source software framework that supports distributed storage and processing of big data using the Mapreduce Programming Model. The cluster is maintained here using commodity hardware [4] and is designed in such a way that the hardware failures are automatically handled [5]. It processes Big Data in parallel and a fault tolerant manner. Hadoop splits files into some chunks and distributes them across nodes in a cluster along with the packaged code for processing of data in parallel. As data is processed in the local system, i.e., on every node of the cluster, the data manipulation is faster and more efficient. This is the reason why the system is faster and efficient in Hadoop system than it would be in conventional super-computer architecture where processing and data are distributed through high-speed networking.

The base Apache Hadoop framework is composed of the following modules:

- Hadoop Common – contains libraries and utilities needed by other Hadoop modules;
- *Hadoop Distributed File System (HDFS)* – a distributed file-system that stores data on commodity machines, providing very high aggregate bandwidth across the cluster [6];
- *Hadoop YARN* – a platform responsible for managing computing resources in clusters [7] and using them for scheduling users' applications; and
- *Hadoop Mapreduce* – an implementation of the Mapreduce programming model for large-scale data processing.

Mapreduce library is written in many programming languages, so the Hadoop framework supports different languages.

The architecture of Hadoop consists of a storage part, known as Hadoop Distributed File System (HDFS), and a processing part which is a Mapreduce programming model as shown in Figure 1. Hadoop cluster includes master and worker nodes. If the cluster is a

small cluster, then there exists single master and multiple worker nodes. If the cluster is a larger cluster, then the cluster may consist more than one number of masters having secondary name node for replication of name node's memory to prevent loss of data and file corruption. A master node consists of task tracker, name node, and resource manager whereas worker node contains job tracker and data node. The detail of architecture is illustrated in Figure 2.

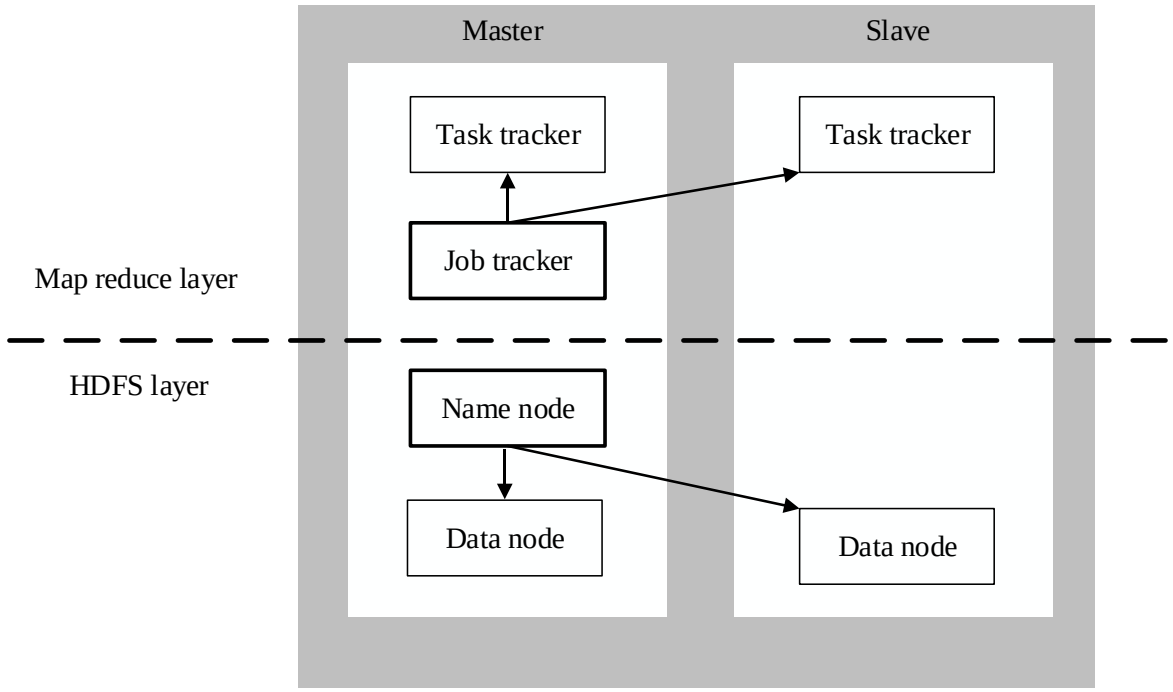


Figure 1 Architectural view of Hadoop.

HDFS Layer

HDFS contains name node and data node in master/slave architecture. A NameNode is a master server that manages the file system namespace and regulates access to files by clients. NameNode only stores the metadata of HDFS – the directory tree of all files in the file system, and tracks the files across the cluster. It does not store the actual data or the dataset. The data itself is actually stored in the DataNodes. Namenode knows the list of the blocks and its location for any given file in HDFS. With this information, NameNode knows how to construct the file from blocks. The NameNode executes the operations like

opening, renaming and closing files and directories. It also calculates the mapping of blocks to DataNodes.

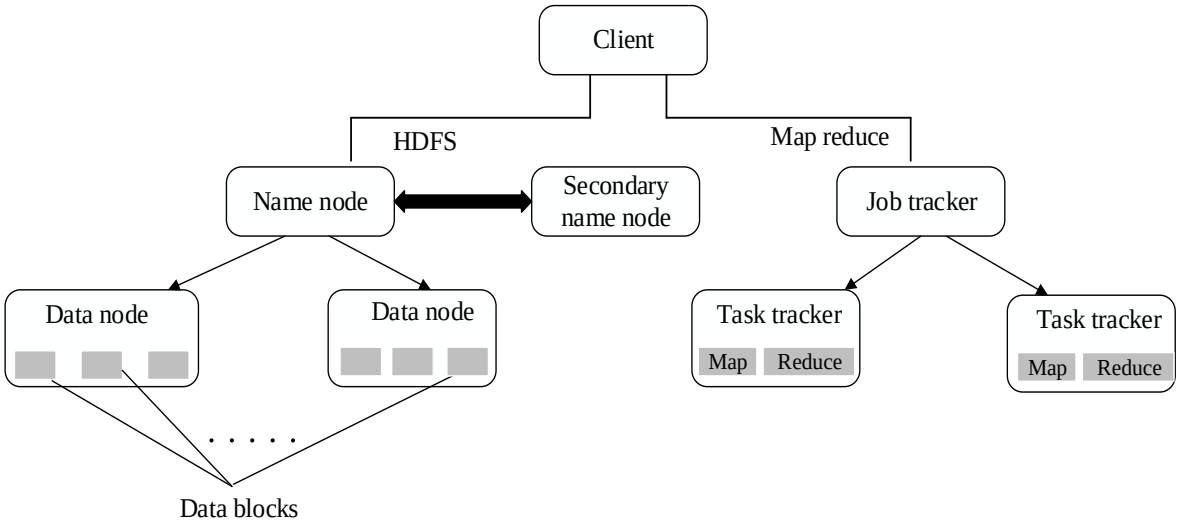


Figure 2 The detail architecture of Hadoop.

There are some DataNodes in HDFS cluster, usually one per node in the cluster, which manages data storage on the nodes on which they are running. The DataNode is responsible for storing the actual data in HDFS. A file is split into a number of chunks/blocks, and these blocks are stored in a set of such DataNodes. When a DataNode starts up it announce itself to the NameNode along with the list of blocks it is responsible for. The DataNodes performs read and write requests from the file system's clients and creation, deletion, and replication of blocks upon instruction from the NameNode. When a DataNode is down, it does not affect the availability of data or the cluster. NameNode will arrange replication for the blocks managed by the DataNode that is not available.

The NameNode and DataNode are pieces of software designed to run on commodity machines. These machines typically run a GNU/Linux operating system (OS). HDFS is built using the Java language; any machine that supports Java can run the NameNode or the DataNode software.

Mapreduce Layer

This layer contains two trackers, job tracker and task tracker for implementing the Mapreduce job. Job tracker to which a Mapreduce job is submitted is responsible for pushing work to the task tracker available in the node in the cluster, to keep the work as close to the data as possible. If the task tracker fails to work, then the work is rescheduled. The allocation of work to TaskTrackers is very simple. Every TaskTracker has a number of available *slots* (such as "4 slots"). Every active map or reduce task takes up one slot. The Job Tracker allocates work to the tracker nearest to the data with an available slot.

Likewise, Mapreduce is a programming model that is capable of processing huge dataset with parallel, distributed algorithm on a cluster. Mapreduce program is composed of map() and reduce() function where map function is responsible for filtering and sorting and reduce function is responsible for summarizing. So this model is based on the split-apply-combine strategy for data analysis. It takes advantage of the locality of data, processing it near the place it is stored to minimize communication overhead. The parallelism that a Mapreduce provides offers the possibility of recovery from partial failure of storage units during operations. Similarly, the data replication prevents the data loss due to any failure. Three different stages of Mapreduce are Map, Shuffle and Reduce.

- **"Map" step:** Each worker node applies the "map()" function to the local data, and writes the output to a temporary storage. A master node ensures that only one copy of redundant input data is processed.
- **"Shuffle" step:** Worker nodes redistribute data based on the output keys (produced by the "map()" function), such that all data belonging to one key is located on the same worker node.
- **"Reduce" step:** Worker nodes now process each group of output data, per key, in parallel

1.3 Spark

The third technology Spark is an open source big data processing framework built around speed, ease of use, and sophisticated analytics. It gives us a comprehensive, unified framework to manage big data processing requirements with a variety of datasets that are diverse in nature (text data, graph data, etc.) as well as the source of data (batch v. real-time

streaming data). It enables applications in Hadoop clusters to run up to 100 times faster in memory and ten times faster even when running on disk and lets quickly write applications in Java, Scala, or Python. It comes with a built-in set of over 80 high-level operators. In addition to Map and Reduce operations, it supports SQL queries, streaming data, and machine learning and graph data processing.

Spark takes Mapreduce to the next level with less expensive shuffles in the data processing. With capabilities like in-memory data storage and near real-time processing, the performance can be several times faster than other big data technologies. It also supports lazy evaluation of big data queries, which helps with optimization of the steps in data processing workflows. It provides a higher-level API to improve developer productivity and a consistent architect model for big data solutions. It holds intermediate results in memory rather than writing them to disk, which is very useful especially when you need to work on the same dataset multiple times. It's designed to be an execution engine that works both in-memory and on-disk. Spark operators perform external operations when data does not fit in memory. It can be used for processing datasets that is larger than the aggregate memory in a cluster. It will attempt to store as much as data in memory and then will spill to disk. It can store part of a data set in memory and the remaining data on the disk. With this in-memory data storage, Spark comes with a performance advantage.

The architecture of Spark consists of Driver, Master, and Executer as shown in Figure 3.

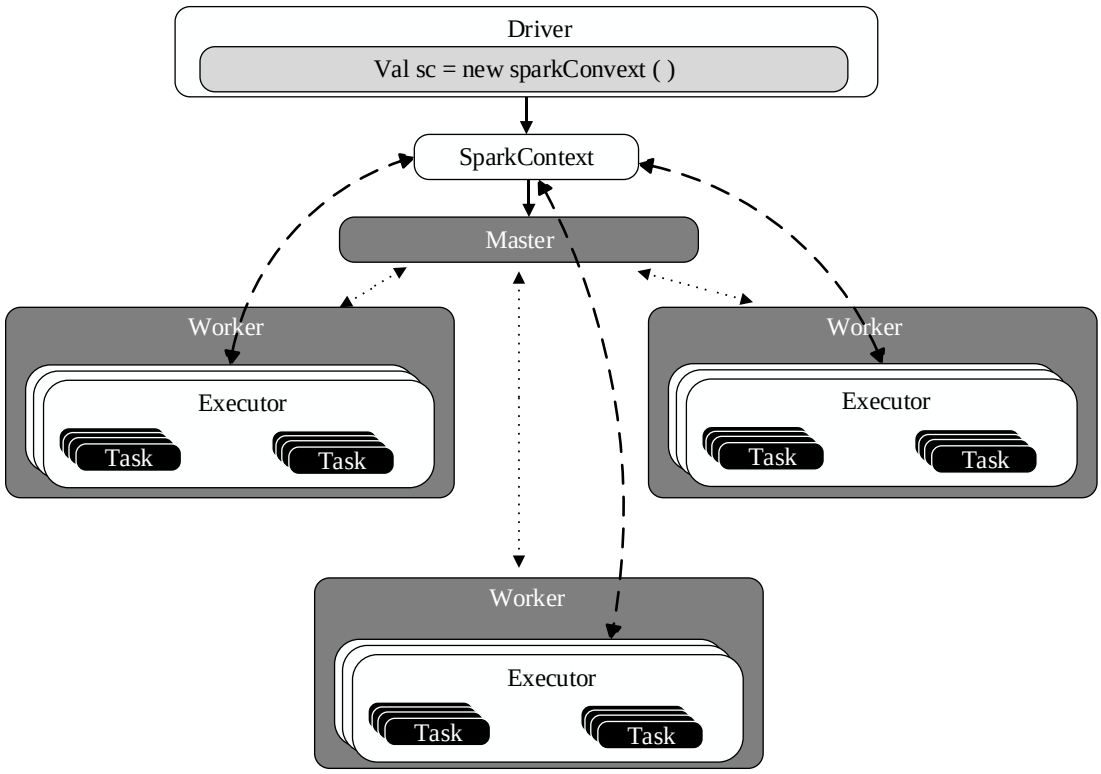


Figure 3 Architectural view of Spark.

Spark uses a **master/worker architecture**. It consists of Driver, Master, and Executer. A driver that talks to a single coordinator called master manage workers in which executers run. The driver and the executors run in their Java processes and can be run on the same or separate machines or in a mixed machine configuration. A Spark Driver is a master node in spark application that hosts spark context for spark application. Spark context establishes a connection to a spark execution environment which when created is used to create RDD, accumulators, access spark services and, run jobs. Specifically, to run on a cluster, the SparkContext can connect to several types of cluster managers (either Spark's standalone cluster manager, Mesos or YARN [8]), which allocate resources across applications. Once connected, Spark acquires executors on nodes in the cluster, which are processes that run computations and store data for the application. Next, it sends application code (defined by JAR or Python files passed to SparkContext) to the executors. And finally, SparkContext sends tasks to the executors to run. Because the driver schedules tasks on the cluster, it should be run close to the worker nodes, preferably on the same local area network.

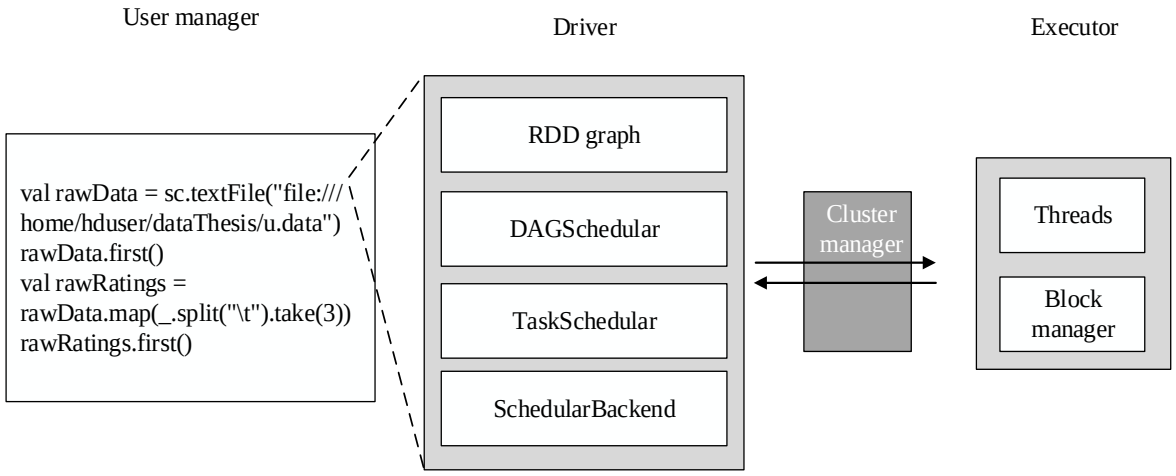


Figure 4 Detail architecture of Spark.

As the Figure 5 entails, the spark application is split into several stages by DAG Scheduler which is scheduled to run on executors by the task scheduler that lives inside the driver [9]. The executor is a distributed agent that is responsible for executing tasks. It provides in-memory storage for RDDs that are cached in Spark applications. When an executor starts it first registers with the driver and communicates directly to execute tasks. It can run multiple tasks over its lifetime, both in parallel and sequential manner. They track running tasks and send metrics (and heartbeats) using the internal heartbeat.

1.4 Purpose

The purpose of this research is to analyze the performance of different big data analytics system on the cloud.

1.5 Problem Statement

Acquiring a solution to improve the accuracy of any Big-data system is a big issue while implementing a huge and complex data, with minimum cost and infrastructure. Moreover, the selection of Big-data analytics tools for different areas is a complicated task for the learner. The comparative study along with could computing paradigm will enable an easier way to apply machine-learning algorithm with an efficient data processing.

1.6 Objective

1. To build a recommendation system using Mapreduce, Spark, and RDBMS.
2. To perform the comparative analysis of Mapreduce, Spark and RDBMS based on computational time, memory utilization, and CPU usage.

2. LITERATURE REVIEW

Hongyong Yu, Deshuai Wang [10] proposed a system for data processing and mining log data of SaaS cloud using Hadoop. The results given in this thesis proved that Hadoop data processing performance is very high as compare to RDBMS, i.e., 28% improvement in the data processing. Apriori algorithm is used for data mining in the cloud which is the best to find association rules from big data. It uses tree structure and bottom-up approach to counting item sets efficiently from data. Parallel computing approach is used in adaptive Apriori algorithm to improve the performance of the system having large data size.

Similarly, Kong Xiangsheng [11] proposed a system for processing and mining scientific data using Mapreduce in cloud environments. It states that the traditional supercomputing centers consisting only of petascale computing resources are not sufficient to tackle the broad range of e-Science challenges. The cloud computing model, based on scientific data centers that scale well enough to support extremely large on-demand loads, are needed to:

Support large numbers of science gateways and their users.
Provide a platform that can support the creation of collaboration and data & application sharing spaces that can be used by virtual organizations.
Manage the computations that are driven by streams of scientific instrument data.

The [12] study of data mining in cloud computing gives the depth knowledge of algorithm for data mining in cloud computing using Mapreduce.

Similarly, [13] MRDS data processing and mining using Hadoop in cloud depict the data processing of mineral resources all over the world. The system is for processing and mining a big mineral resources data system's data. To enhance the performance of data processing Hadoop's Mapreduce architecture is used. For better improvement in data mining for MRDS, Apriori algorithm is used and improves the performance of the existing system more than 30%. This shows that, by grouping a good and open source technology such as Hadoop and Apriori algorithm together, we can achieve a better data processing and mining

for any system.

Every day, 2.5 quintillion bytes of data are created and 90 percent of the data in the world today were produced within the past two years [14] [15] . The commonly used software technology cannot cope with massive data, and the big challenge is to extract important information from it. Big data has a large volume, heterogeneous format, and decentralized data control. The example of big data applications is Facebook, Twitter, and Google. It is a big challenge to manage and mining a massive data because of its volume, different file formats and growth rate of the data in the world. There are many challenges with big data such as storage, processing, variety, and cost.

There are several practical simulation-enabled analytics systems. One such system is given by Li, Calheiros, Lu, Wang, Palit, Zheng and Buyya, which is a Direct Acrylic Graph (DAG) form analytical application used for modeling and predicting the outbreak of Dengue in Singapore.

It was in the 1980s that artificial intelligence-based algorithms were developed for data mining. Wu, Kumar, Quinlan, Ghosh, Yang, Motoda, McLachlan, Ng, Liu, Yu, Zhou, Steinbach, Hand and Steinberg mention the ten most influential data mining algorithms k-means, C4.5, Apriori, Expectation Maximization (EM), PageRank, SVM (support vector machine), AdaBoost, CART, Nave Bayes and kNN (k-nearest neighbors). Most of these algorithms have been used commercially as well.

Aaron N. Richter, Taghi M. Khoshgoftaar, Sara Landset, and Tawfiq Hasanin proposed a complete multidimensional examination of different open source devices likes Mahout, MLlib, H2O, and SAMOA for machine learning with huge information. An assessment standard is proposed alongside correlations of the structures talked about these open source technologies.

Satish Gopalani and Rohan Arora [16] gives the analysis between Hadoop Map Reduce and the as of late presented Apache Spark utilizing a standard machine learning calculation for

K-Means clustering.

Juwei Shi, YunjieQiu, Umar Farooq Minhas, Limei Jiao, Chen Wang, Berthold Reinwald, and Fatma O¨zcan [17]assess the major compositional segments in Mapreduce and Spark systems including merging, execution time, and storing, by utilizing an arrangement of critical investigative workloads.

Sara Landset, Taghi M. Khoshgoftaar, Aaron N. Richter and Tawfiq Hasanin, [18] gives a rundown of criteria to making determinations of devices for Big Data Analytic alongside an investigation of the focal points and downsides of each.
Jai Prakash Verma, Bankim Patel, and Atul Patel, [19] give execution of information investigation utilizing Hadoop Framework for the content dataset.

3. METHODOLOGY

Here the recommendation list is computed by using three approaches, a traditional approach using RDBMS, Mapreduce programming paradigm, and spark programming paradigm. The performance is analyzed and compared by the execution time by all the approaches to perform. Figure 5 shows a basic block diagram of methodology, which includes processing, analysis and comparison units. The system begins with input dataset, which is a data set of songs from Yahoo. The processing and analysis section comes under Cloud. Data will be processed with three different processing tools, Hadoop, Spark and the RDBMS. The processed dataset will be analyzed with a fixed recommendation filter, for which Collaborative filtering is used. The collaborative is a standardized filter and widely used in research works. To verify the comparison analysis study, a standard filter is taken as reference. However, any kinds of recommendation filter can be used. Moreover, finally, the filtering result of individual processing methods will be compared by execution time.

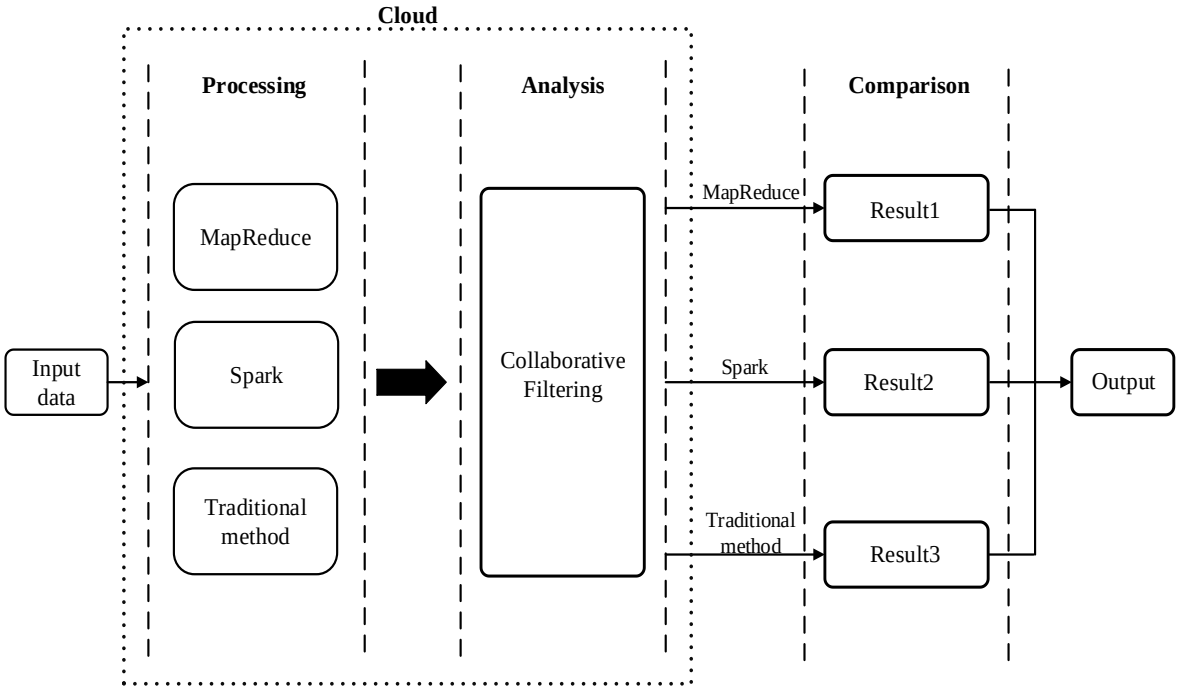


Figure 5 Basic block of methodology.

Mapreduce jobs run in parallel in Hadoop cluster. The large dataset file is divided into several blocks and distributed across several data nodes, and Mapreduce job uses the dataset relevant to that node and performs the required action specified on the job. Each

map task process the input and generates the required intermediate data value as output and then the reduce task takes those output as its input and process it according to the reduce job specified to give an output with key-value pair of the unique key.

Similarly, the spark runs the job in parallel. It revolves around the concept of a resilient distributed dataset (RDD) [20], which is a fault-tolerant collection of elements that can be operated on in parallel [21]. RDDs support two types of operations: transformations, which create a new dataset from an existing one, and actions, which return a value to the driver program after running a computation on the dataset. Spark translates the RDD transformations into something called DAG (Directed Acyclic Graph) and starts the execution. At high level, when any action is called on the RDD, Spark creates the DAG and submits to the DAG scheduler. The DAG scheduler divides operators into stages of tasks. A stage is comprised of tasks based on partitions of the input data. The DAG scheduler pipelines operators together. E.g., Many map operators can be scheduled in a single stage. The final result of a DAG scheduler is a set of stages. The Stages are passed on to the Task Scheduler.The task scheduler launches tasks via cluster manager. However, task scheduler will not know about dependencies of the stages. Moreover, finally, the Worker executes the tasks on the Slave.

The working procedure of the RDBMS is different from that of technologies described above; it differs largely in data processing. The parallel processing of data is not as more comfortable as on two other technologies, and the storage and processing of unstructured data are also not possible because of its presentation of data in row and column format. For big data, what is needed is, storing data in different systems and copying code into those systems and processing it in parallel with taking advantage of locality. This is not possible in RDBMS, the working of which starts with copying data from storage to the RAM and then processing it there, which is inefficient and time-consuming task while considering big data.

3.1 Computing Similarities between Item Pairs

To compare three different techniques, the comparison is made by recommendation system

using those approaches. So firstly the calculation of similarities is computed on all the three technologies. There are many ways to formalize this calculations, such as the similarity metrics of cosine and Pearson correlation. For this research, cosine similarity measure has been chosen. In an item-based approach, the similarity between two items is calculated.

$$\mathrm{Cos}(\theta) = \frac{A.B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i-1}^{n}(A_i)^2} \times \sqrt{\sum_{i-1}^{n}(B_i)^2}} \quad (1)$$

It is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them. The cosine of 0° is 1, and it is less than 1 for any other angle. It is thus a judgment of orientation and not magnitude: two vectors with the same orientation have a cosine similarity of 1, two vectors at 90° have a similarity of 0, and two vectors opposed have a similarity of -1, independent of their magnitude. So if two items, i.e., songs are similar, then the cosine similarity is equal to 1 and a value of 0 if they are not similar, and if they are opposite (dissimilar) then the cosine value will be -1. So the similarity score will always be in the range -1 to 1.

3.1.1 Calculation using RDBMS

To obtain the required output, the similar item needs to be calculated, for which neighborhood formation is done. Neighborhood formation consists of finding the most similar items to the item preferred by the active user based on their past agreement on ratings. Items that tend to be rated the same by similar users will be classed as similar under this approach. After getting these similarities, it can represent a user regarding the items they have interacted with and find items that are similar to these known items, which they can recommend to the user.

The cosine formula (1) above is used to calculate similarity score which gives the list of scores for item pair. This can be done using any languages, but this research had used java as front end and MySQL as backend for RDBMS implementation. Moreover, all the process is done on the same machine especially carried out on RAM.

3.1.2 Calculation using Mapreduce

The similarity computation of recommendation system is divided into 2 Mapreduce jobs. The first job takes input from a yahoo dataset having each row of data separated by a tab as shown in Figure 6.

196	242	3	881250949
186	302	3	891717742
22	377	1	878887116
244	51	2	880606923
166	346	1	886397596
298	474	4	884182806
115	265	2	881171488
253	465	5	891628467
305	451	3	886324817
6	86	3	883603013
62	257	2	879372434
286	1014	5	879781125
200	222	5	876042340
210	40	3	891035994
224	29	3	888104457
303	785	3	879485318
122	387	5	879270459
194	274	2	879539794
291	1042	4	874834944
234	1184	2	892079237
119	392	4	886176814
167	486	4	892738452
299	144	4	877881320
291	118	2	874833878
308	1	4	887736532
95	546	2	879196566
38	95	5	892430094
102	768	2	883748450
63	277	4	875747401
160	234	5	876861185
50	246	3	877052329
301	98	4	882075827
225	193	4	879539727
290	88	4	880731963
97	194	3	884238860
157	274	4	886890835
181	1081	1	878962623
278	603	5	891295330
276	796	1	874791932
7	32	4	891350932
10	16	4	877888877
284	304	4	885329322
201	979	2	884114233
276	564	3	874791805
287	327	5	875333916
246	201	5	884921594
242	1137	5	879741196
249	241	5	879641194
99	4	5	886519097
178	332	3	882823437
251	100	4	886271884
81	432	2	876535131
260	322	4	890618898
25	181	5	885853415
59	196	5	888205088
72	679	2	880037164
87	384	4	879877127
290	143	5	880474293
42	423	5	881107687
292	515	4	881103977
115	20	3	881171009
20	288	1	879667584
201	219	4	884112673
13	526	3	882141053
246	919	4	884920949
138	26	5	879024232
167	232	1	892738341
60	427	5	883326620
57	304	5	883698581
223	274	4	891550094
189	512	4	893277702
243	15	3	879987440
92	1049	1	890251826
246	416	3	884923047
194	165	4	879546723
241	690	2	887249482
178	248	4	882823954
254	1444	3	886475558
293	5	3	888906576
127	229	5	884364867
225	237	5	879539643

Figure 6 Input to Mapreduce-1.

The output of this job is a key-value pair of a key representing userid and the value with the set of all the songs with their ratings given by that user. This output will be the input to the next Mapreduce job, which is responsible for computing a similarity between items. The overall process is illustrated in Figure 7.

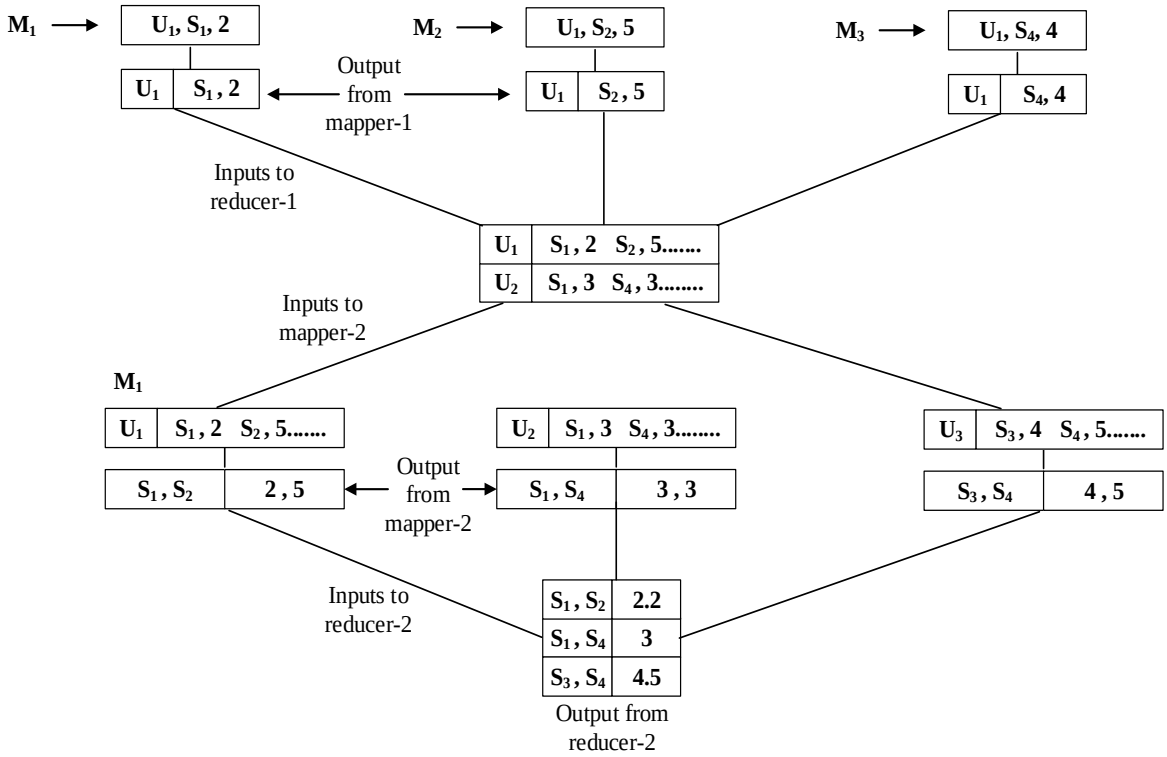


Figure 7 Process of Mapreduce for similarity calculation. Here, U, S, and M represent userid, songid, and Mapper, respectively.

Finally, the output is represented in the form of key-value pair, where the key represents the pair of items and value gives the similarity score for those items.

First Mapreduce Job

As mentioned earlier, the first Mapreduce job computes the input to give a set of songs with ratings by every single user. Each row of data is processed by the mapper-1 where the output pair is generated by splitting the input data by a tab' /t'. This output intermediate key-value pair contains the key that is a userid and value that represents a set of songid and rating. Finally, the output from the first mapper is now an input to the reducer-1. Reducer-1 collects all the songs along with rating given by the user, where the key is a unique user and value is a collection of songs with ratings by that user.

Second Mapreduce Job

Here the output from the reducer-1 from a first Mapreduce job is input to the mapper-2 where mapper-2 splits the value field of the input data by tab '/t' and delimiter ',' and

generates an intermediate pair of the item-item pair as key and rating-rating pair as value. This intermediate pair is given to the final reducer, which is responsible for computing similarity calculation using cosine similarity (1).

Finally, the output from the reducer-2 is a pair of item-item pair of the song as key and similarity score between them as a value, which will be the output from the similarity calculation phase.

3.1.3 Calculation using Spark

The Spark's recommendation system is based on the matrix factorization model, which is easier to compute recommendation. These models are often called latent feature models, as it discovers some form of hidden features which are represented by the factor matrices. While the latent features or factors are not directly interpretable, they might, perhaps, represent things such as the tendency of a user to like songs from a certain singer, genre, musician, or group of artists, for example. Matrix factorization models a user-item matrix by representing it as a product of two smaller matrices of lower dimension as shown in Figure 8 and 9. Thus, it is a dimensionality-reduction technique. For a user-item matrix with the dimension **k**, the two matrices are one for users of size U x k and one for items of the size I x k as presented. Moreover, these are known as factor matrices. The multiplication of these two-factor matrices reconstructs an approximate version of the original rating matrix.

	Song1	Song2	Song3
Ram	4	5	5
Shyam		5	5
Hari		5	?

Figure 8 Users Item Rating Matrix.

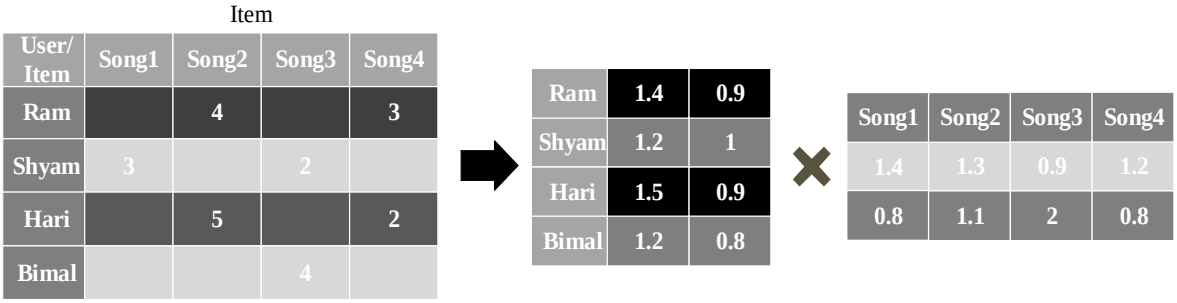


Figure 9 Calculation of a recommendation.

To find out the similarity between two items, the same measures of similarity as in the nearest-neighbor models can be used, except the use of the factor vectors directly by computing the similarity between two item-factor vectors.

However, these models are relatively more complicated to understand and interpret compared to nearest-neighbor models. So to solve the problem of this model spark implements an algorithm called alternative least square which is powerful technique and has proven to be relatively easy to implement in a parallel fashion. ALS works by iteratively solving a series of least squares regression problems. In each iteration, one of the user- or item-factor matrices is treated as fixed, while the other one is updated using the fixed factor and the rating data [22]. Then, the factor matrix that was solved for is, in turn, treated as fixed, while the other one is updated. This process continues until the model has converged.

Computing similarity between items has two processes, one, training a model that generates user and item factors and the other, cosine similarity calculation for similarity computation. To train the model, Mllib library of spark is used that makes use of ALS algorithm. The training method of this algorithm returns a *MatrixFactorizationModel* object, which contains user and item factors in the form of an RDD of *(id, factor)* pairs. Operations used in MLlib's *ALS* implementation are lazy transformations, so the actual computation will only be performed once some action on the resulting *RDDs* of the user and item factors are called.

Now to compute the similarity score, jblas linear algebra library is used to compute the

required vector dot product for cosine similarity. Cosine similarity is a measure of the angle between two vectors in an *n*-dimensional space. It is computed by first calculating the dot product between the vectors and then dividing the result by a denominator, which is the norm (or length) of each vector multiplied together (specifically, the L2-norm is used in cosine similarity) which is given by the equation below.

$$\text{Val simScore} = \frac{\text{itemVector1. dot (itemVector2)}}{\text{itemVector1. norm2 ()} * \text{itemVector2. norm2 ()}} \quad (2)$$

The cosine similarity measure takes on values between -1 and 1. A value of 1 implies completely similar, while a value of 0 implies independence (that is, no similarity). This measure is useful because it also captures negative similarity, that is, a value of -1 implies that not only are the vectors not similar, but they are also completely dissimilar.

3.2 Deriving Recommendation

The final part of the recommendation system is to derive a recommendation list.

3.2.1 Deriving recommendation list using RDBMS

The list generated after calculating cosine similarity is sorted by similarity score for item pairs in descending order, which gives the final list of recommendation for the item.

This process is carried out on a single machine using MySQL for data storage and processing purpose. MySQL falls into the category of relational database management system, so it stores data in row-column format and process data on RAM taking it from its original storage. So the sorting of data from the list is carried out on RAM, and final output after processing is only then stored on the storage.

3.2.2 Deriving recommendation list using MapReduce

The derivation of recommendation list is accomplished by MapReduce-3, which is based on the similarity value calculated for the item pair as illustrated in Figure 11. The pair having highest similarity value will have the highest priority rank for that item.

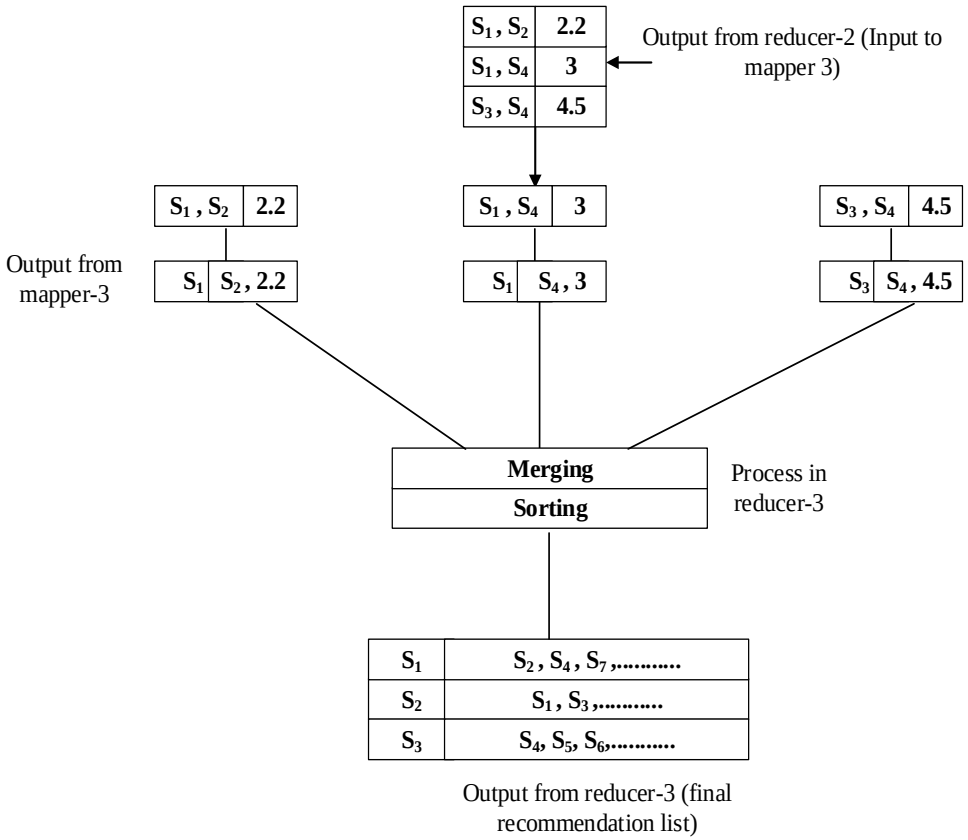


Figure 10 Deriving Recommendation list.

The output from the MapReduce-2 is given to the input of MapReduce-3 where the merging and sorting are done to accomplish desired output of recommendation list. Mapper-3 will process the data coming from MapReduce-2 by splitting each row of data by a tab '\t' and a delimiter ','. The output from the mapper-3 is now an intermediate key-value pair of songid as key and value having a pair of song and similarity scores with that key. Finally, the output from mapper-3 is given to the final reducer, which processes data to give a final list of recommendation. The data is merged and sorted based on the similarity score. So the final recommendation list output will be the key value pair of songid as key and similar songs to that key as a value in descending order of similarity metrics as shown in Figure 12.

3.2.3 Deriving recommendation list using Spark

Finally, the recommendation list is generated after calculating similarity measures. This list is the list of item pairs and similarity value sorted in descending order of similarity.

This is achieved by using the top method of ALS algorithm, which automatically sorts the

list in the required format as shown in Figure 13.

3.3 Cloud computing

To overcome the need for large-scale computing, processing, and its infrastructure maintenance Cloud computing would be the best service. So the Cloud Dataproc is chosen as well suited service that is matched with the requirement of this research. Cloud Dataproc supports Spark and Hadoop clustering with faster processing where operations that used to take hours or days take seconds or minutes giving a powerful and complete platform for data processing, analytics, and machine learning [23]. 'Pay as you go' is available in this service along with initial free service of value $300. Creating Cloud Dataproc clusters is quicker and can be resized them at any time—from three to hundreds of nodes so no need to be worried about data pipelines outgrowing clusters. With each cluster action taking less than 90 seconds on average, more time can be focused on insights, with less time lost to infrastructure.

Creating a cluster is the easiest task in Cloud Dataproc. The specification of the size and quantity of node and master is only the requirement to accomplish it. The list of clusters can be visible with the option of update to resize a cluster in future need.

After creating a cluster, job or a program can be run using the interface with specifying the cluster name, job type, i.e., whether it is spark or Hadoop and a jar file of that program. Just a click on the submit button will then give the required output of desire.

3.4 Execution time calculation

The execution time of every system is calculated to compare the performance. Moreover, this time is calculated by considering starting point and end point of execution of a task. Subtracting the start value with an end value of execution period gives overall execution time taken by the system to complete its task, which is as below:

Execution Time = (starting time-end time) in sec (3)

This value is considered as a metric to calculate the performance of every system.

4 RESULT ANALYSIS AND COMPARISION

Different technologies with their algorithms are designed to compute recommendations based on the similarity between two items. To derive recommendations, yahoo dataset is used as input data for the experiments, which is a collection of rating data by the user for their preferred song. Experiments are performed on three of the algorithms for a single node and multi-node clusters on both cloud-based and non-cloud based system, and performance metrics of each are recorded. For the cloud-based experiment, Cloud Dataproc is used as a cloud system where the single node and multi-node clusters are created and performed an experiment for all the three algorithms. In case of RDBMS, it always experiments on single machine whether it is cloud or non-cloud. The hardware and software configuration for the system is listed in Table 1.

Table 1 System hardware and software configurations for experiments.

SN	Cloud/Non-cloud	Single/Multi Cluster	Name	Specification
1	Both	Single	Primary Disk Size	27GB
2	Both	Single	Memory	3GB
3	Non-cloud	Single	Operating System	Ubuntu (64bit)
4	Cloud	Multi (Worker node)	Primary Disk Size	50GB
5	Cloud	Multi (Worker node)	Memory	7.5GB
6	Cloud	Multi (master node)	Primary Disk Size	100GB
7	Cloud	Multi (master node)	Memory	7.5GB
8	Apache Spark: version 2.2.0			
9	Apache Hadoop: version 2.8.1 in cloud			
10	Apache Hadoop: version 2.7.3 noncloud			
11	MySQL Database: version 2.2.0			
12	Number of Nodes in multi-node cluster: 3			

4.1 The Evaluation of Results

4.1.1 Dataset Selection

One important component of evaluation is the dataset used to perform experiment. It is

essential to use a constant and easy to reproduce dataset. Since the goal, it is to perform analysis on recommendation output from different framework; information about music preferences from different people is needed. Here dataset of the song with different data size from yahoo is used that stores information related to music preferences from millions of songs and users which are listed in Table 2. So the same dataset is used for performance analysis too. The sample input dataset is shown in Figure 11.

Table 2 Dataset Specifications.

No of songs	No of users	Total ratings
1682	943	100000
127771	200000	2000000
1639720	2988761	6231790
9086238	13565203	16231790

```
196     242     3       881250949
186     302     3       891717742
22      377     1       878887116
244     51      2       880606923
166     346     1       886397596
298     474     4       884182806
115     265     2       881171488
253     465     5       891628467
305     451     3       886324817
6       86      3       883603013
62      257     2       879372434
286     1014    5       879781125
200     222     5       876042340
210     40      3       891035994
224     29      3       888104457
303     785     3       879485318
122     387     5       879270459
194     274     2       879539794
291     1042    4       874834944
234     1184    2       892079237
119     392     4       886176814
167     486     4       892738452
299     144     4       877881320
291     118     2       874833878
308     1       4       887736532
95      546     2       879196566
38      95      5       892430094
102     768     2       883748450
63      277     4       875747401
160     234     5       876861185
50      246     3       877052329
301     98      4       882075827
225     193     4       879539727
290     88      4       880731963
97      194     3       884238860
157     274     4       886890835
181     1081    1       878962623
278     603     5       891295330
276     796     1       874791932
7       32      4       891350932
10      16      4       877888877
284     304     4       885329322
201     979     2       884114233
276     564     3       874791805
287     327     5       875333916
246     201     5       884921594
242     1137    5       879741196
249     241     5       879641194
99      4       5       886519097
178     332     3       882823437
251     100     4       886271884
81      432     2       876535131
260     322     4       890618898
25      181     5       885853415
59      196     5       888205088
72      679     2       880037164
87      384     4       879877127
290     143     5       880474293
42      423     5       881107687
292     515     4       881103977
115     20      3       881171009
20      288     1       879667584
201     219     4       884112673
13      526     3       882141053
246     919     4       884920949
138     26      5       879024232
167     232     1       892738341
60      427     5       883326620
57      304     5       883698581
223     274     4       891550094
189     512     4       893277702
243     15      3       879987440
92      1049    1       890251826
246     416     3       884923047
194     165     4       879546723
241     690     2       887249482
178     248     4       882823954
254     1444    3       886475558
293     5       3       888906576
127     229     5       884364867
225     237     5       879539643
```

Figure 11 Sample input dataset.

4.1.2 Performance Measurement and Metrics

For the three case studies, on single node cluster in local mode and single node and multi-node cluster in the cloud, performance in RDBMS, Hadoop and Spark are compared by running time, memory consumption and CPU usage. To keep a fair comparison, the following metrics are guaranteed, which are applied, to RDBMS, Hadoop, and Spark:

- ❖ RDBMS, Hadoop, and Spark run on the same machine and configuration for single node and multi-node cluster environment on cloud and single cluster on non-cloud based system
- ❖ Hadoop and Spark platforms run on the same cluster machines
- ❖ Both Hadoop and Spark use HDFS as the file storage system
- ❖ Case studies implemented in all systems are based on the same programing language
- ❖ Start Time and Finish Time are listed to calculate the elapsed time of all three applications, as shown in the Figures 12, 13, and 14 for cloud-based and non-cloud based environments respectively:
- ❖ At last memory consumption and CPU usage is recorded for all applications in all environments, as shown in Figure 15 and 16.

Completed Jobs (12)

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
11	take at <console>:44	2017/10/29 15:22:03	0.2 s	1/1 (23 skipped)	1/1 (23 skipped)
10	collect at <console>:37	2017/10/29 15:22:00	39 ms	1/1 (23 skipped)	1/1 (23 skipped)
9	first at MatrixFactorizationModel.scala:67	2017/10/29 15:21:58	24 ms	1/1 (23 skipped)	1/1 (23 skipped)
8	first at MatrixFactorizationModel.scala:67	2017/10/29 15:21:58	62 ms	1/1 (24 skipped)	1/1 (24 skipped)
7	count at ALS.scala:280	2017/10/29 15:21:57	0.4 s	1/1 (23 skipped)	1/1 (23 skipped)
6	count at ALS.scala:279	2017/10/29 15:21:51	7 s	22/22 (3 skipped)	22/22 (3 skipped)
5	count at ALS.scala:865	2017/10/29 15:21:49	0.3 s	2/2 (1 skipped)	2/2 (1 skipped)
4	count at ALS.scala:857	2017/10/29 15:21:47	2 s	3/3	3/3
3	isEmpty at ALS.scala:843	2017/10/29 15:21:47	28 ms	1/1	1/1
2	isEmpty at ALS.scala:240	2017/10/29 15:21:47	32 ms	1/1	1/1
1	first at <console>:36	2017/10/29 15:21:47	45 ms	1/1	1/1
0	first at <console>:32	2017/10/29 15:21:43	0.6 s	1/1	1/1

Figure 12 Spark Application Web UI in the non-cloud based environment.

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
6	0	0	6	0	0 B	8 GB	0 B	0	8	0	1	0	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation
Capacity Scheduler	[MEMORY]	<memory:1024, vCores:1>	<memory:8192, vCores:8>

Show 20 entries Search:

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI	Blacklisted Nodes
application_1509255588375_0001	hduser	CFThesis.jar	MAPREDUCE	default	Sun Oct 29 14:41:38 +0900 2017	Sun Oct 29 14:42:41 +0900 2017	FINISHED	SUCCEEDED		History	N/A
application_1509255588375_0002	hduser	CFThesis.jar	MAPREDUCE	default	Sun Oct 29 14:42:45 +0900 2017	Sun Oct 29 14:45:07 +0900 2017	FINISHED	SUCCEEDED		History	N/A
application_1509255588375_0003	hduser	CFThesis.jar	MAPREDUCE	default	Sun Oct 29 14:45:10 +0900 2017	Sun Oct 29 14:46:23 +0900 2017	FINISHED	SUCCEEDED		History	N/A

Figure 13 Hadoop Application Web UI in the non-cloud based environment.

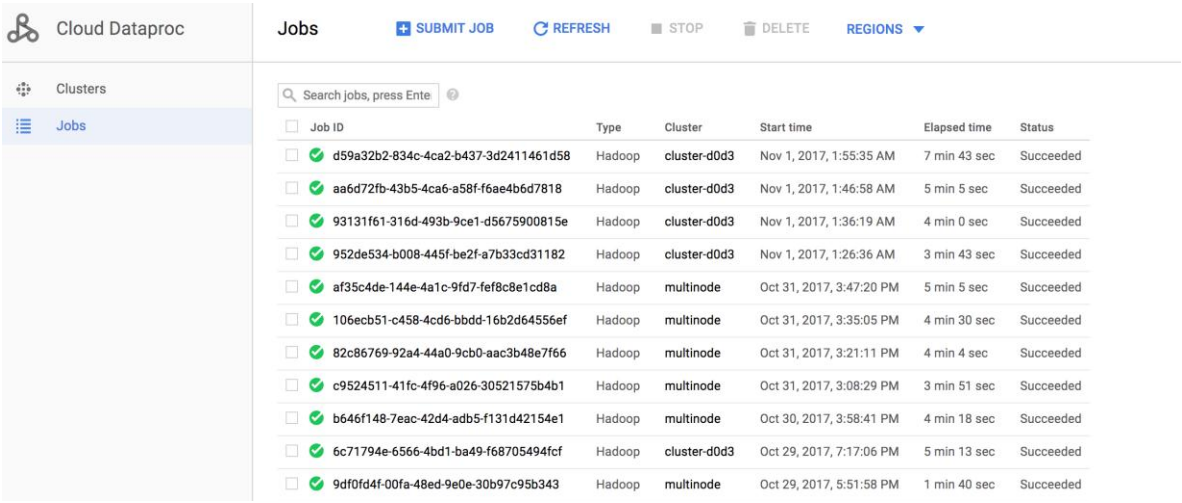


Figure 14 Hadoop Application Web UI in cloud-based environment.

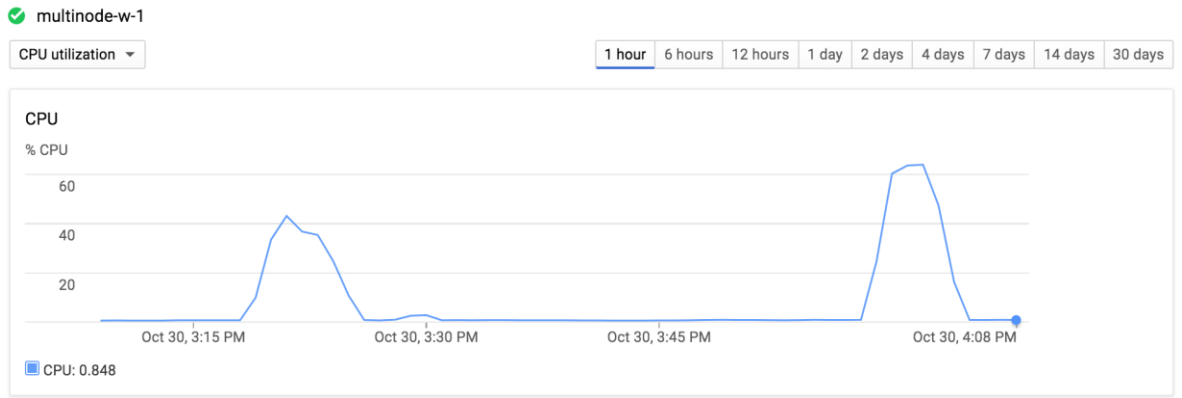
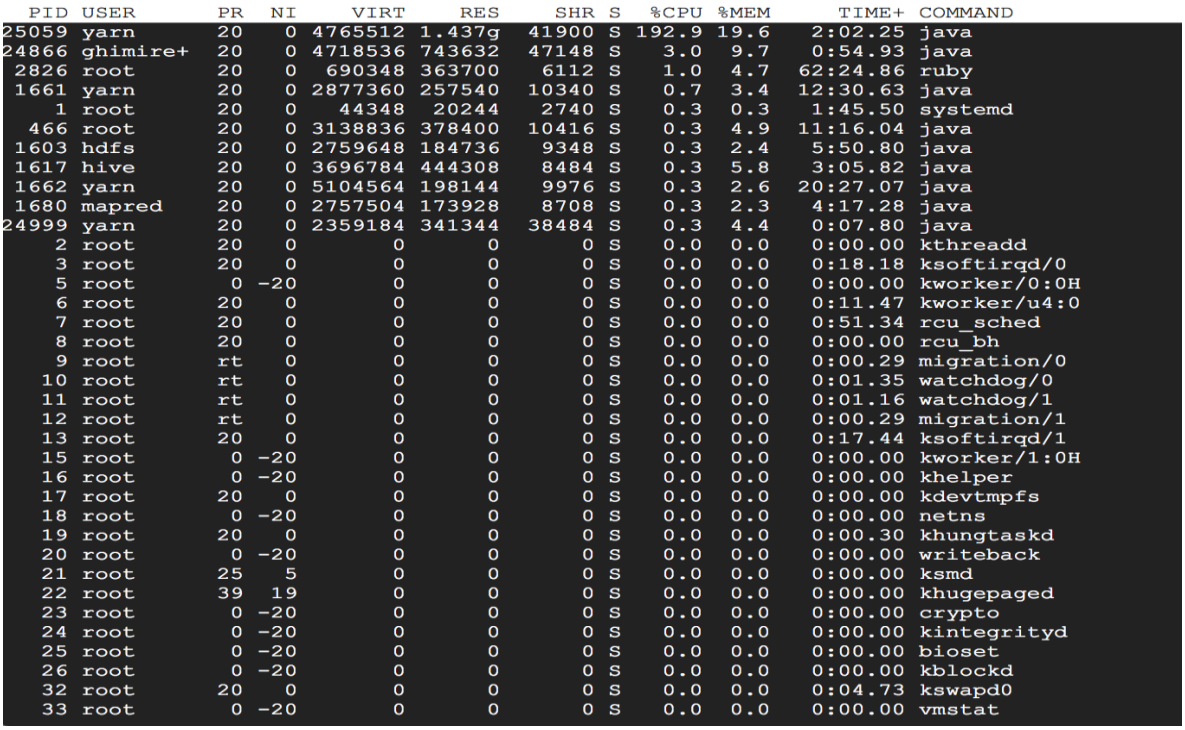


Figure 15. CPU utilization by Spark in the cloud.



PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
25059	yarn	20	0	4765512	1.437g	41900	S	192.9	19.6	2:02.25	java
24866	ghimire+	20	0	4718536	743632	47148	S	3.0	9.7	0:54.93	java
2826	root	20	0	690348	363700	6112	S	1.0	4.7	62:24.86	ruby
1661	yarn	20	0	2877360	257540	10340	S	0.7	3.4	12:30.63	java
1	root	20	0	44348	20244	2740	S	0.3	0.3	1:45.50	systemd
466	root	20	0	3138836	378400	10416	S	0.3	4.9	11:16.04	java
1603	hdfs	20	0	2759648	184736	9348	S	0.3	2.4	5:50.80	java
1617	hive	20	0	3696784	444308	8484	S	0.3	5.8	3:05.82	java
1662	yarn	20	0	5104564	198144	9976	S	0.3	2.6	20:27.07	java
1680	mapred	20	0	2757504	173928	8708	S	0.3	2.3	4:17.28	java
24999	yarn	20	0	2359184	341344	38484	S	0.3	4.4	0:07.80	java
2	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kthreadd
3	root	20	0	0	0	0	S	0.0	0.0	0:18.18	ksoftirqd/0
5	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworker/0:0H
6	root	20	0	0	0	0	S	0.0	0.0	0:11.47	kworker/u4:0
7	root	20	0	0	0	0	S	0.0	0.0	0:51.34	rcu_sched
8	root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcu_bh
9	root	rt	0	0	0	0	S	0.0	0.0	0:00.29	migration/0
10	root	rt	0	0	0	0	S	0.0	0.0	0:01.35	watchdog/0
11	root	rt	0	0	0	0	S	0.0	0.0	0:01.16	watchdog/1
12	root	rt	0	0	0	0	S	0.0	0.0	0:00.29	migration/1
13	root	20	0	0	0	0	S	0.0	0.0	0:17.44	ksoftirqd/1
15	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworker/1:0H
16	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	khelper
17	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kdevtmpfs
18	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	netns
19	root	20	0	0	0	0	S	0.0	0.0	0:00.30	khungtaskd
20	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	writeback
21	root	25	5	0	0	0	S	0.0	0.0	0:00.00	ksmd
22	root	39	19	0	0	0	S	0.0	0.0	0:00.00	khugepaged
23	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	crypto
24	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kintegrityd
25	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	bioset
26	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kblockd
32	root	20	0	0	0	0	S	0.0	0.0	0:04.73	kswapd0
33	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	vmstat

Figure 16 Monitoring memory utilization.

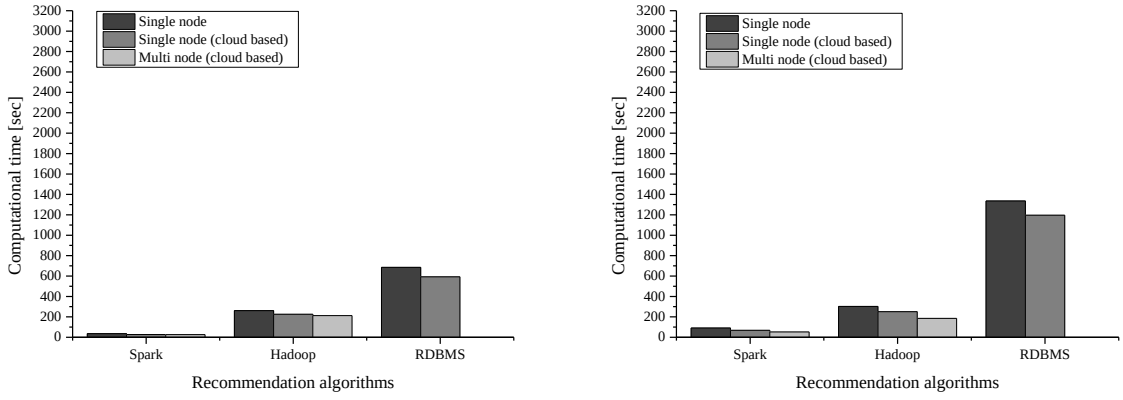
4.1.3 The Comparison of Results

Case studies carried out consist of the evaluation of three different recommendation algorithms with a different configuration of data networks. Each algorithm with each node configuration of data networks is evaluated individually with four different sizes of datasets. Each of the case studies was repeated three times to obtain the average running results. Sometimes because of unstable network traffic, there are a few seconds of error band for a small dataset or tens of seconds of error band for a big dataset. Tables 3 shows the average running time, memory utilization and CPU usage based comparison on different sizes of data for each case study in RDBMS, Hadoop and in Spark. The observations are as follows:

Table 3 Experiments carried out on various recommendation algorithms at different configurations of data networks with various sizes of datasets.

	SN	100000 ratings			2000000 ratings			6231790 ratings			16231790 ratings		
Case Studies		C.Time	%Memory	%CPU	C.Time	%Memory	%CPU	C.Time	%Memory	%CPU	C.Time	%Memory	%CPU
	1	37.3	18.2	97.2	90.144	26.2	98.3	118.18	41.8	98.7	293.7	43	98.7
	2	31.75	17.8	96.7	90.484	26.8	98.3	121.421	42.7	98.7	278.56	43.6	98.7
spark-single node(non-cloud based)	3	36.52	18.3	95	90.737	26.3	98.3	118.875	39.5	98.3	299.3	43.5	98.6
	1	27.52	5.11	56	67.24	15.3	80.9	102.08	19.7	80.44	135.64	25.8	92.16
	2	27.53	5.9	53.29	73.01	12.7	80.2	109.88	19	81.2	138.642	26.4	90.24
spark-single node(cloud based)	3	27.36	6.2	51	64.59	15.7	79.4	115.04	19.8	83.45	132.628	27.5	92.87
	1	25.73	4.525	17.05	52.67	11.6	51.84	85.01	16.85	63.65	116.7	23.04	85.01
	2	26.079	5.16	24.2	55.07	9.7	55.6	90.04	15.02	69.02	116.98	22.87	86.28
spark-multi node(cloud based)	3	26.02	5.78	23.4	48.28	9.2	53.48	89.01	12.01	66.78	116.96	24.5	84.2893
	1	267.98	6.193	87.68	312.487	7.3	90.67	379.34	7.425	81.6625	936.66	8.55	63.881
	2	260.34	6.01	86.82	296.3	7.61	89.362	336.39	8.6	84.98	1080.8	8.2	67.467
Hadoop-single node(non-cloud based)	3	255.21	5.98	87.09	298.76	7.01	89.04	398	7.89	80.67	964.45	9.1	66.98
	1	224.45	5.2	41.77	251.67	7.1	50.07	290.732	9.8	50.51	458.519	12.1	56.87
	2	227.673	5.6	42.07	251.79	6.9	51.22	295.37	9.8	51.23	460.231	12.3	54.23
Hadoop-single node(cloud based)	3	223.23	5.1	41.2	248.512	7.5	50.7	296.026	9.6	51	460.42	12	54.87
	1	211.29	4.7375	50.1175	233.534	5.25	60.6875	262.64	6.7687	59.18	388.814	9.031	65.544
	2	213.45	4.123	52.324	233.62	5.08	56.04	260.76	6.245	58.0232	301.558	8.79	68.61
Hadoop-multi node(cloud based)	3	211.598	4.87	51.09	233.01	6.01	58.97	258.975	7.02	61.089	308.357	9.867	69.345
	1	684.33	56.72	63.57	1421.23	63.33	76.42	2001.25	64.9	81.53	3301.2	71.45	92.46
	2	685.1	55.89	61.4	1256.77	63.67	78.2	1866.12	65.76	81.01	3111.43	70.62	91.9
RDBMS(non-cloud based)	3	684.61	56.02	62.9	1330	62.9	78.01	1903.72	64.02	81.78	3032.01	70.1	91.05
	1	593.389	51.07	23.32	1189	54.32	46.3	1698.32	58.3	42.3	2985.23	63.2	67.32
	2	591.49	50.23	26.7	1208	52.01	44.3	1687.34	58.9	45.67	3001.34	64.01	68.9
RDBMS(cloud based)	3	592.83	49.89	29.3	1191.2	53.98	30.2	1690.94	58.02	43.98	2989.01	63.9	63.56

Moreover, all the configurations mentioned above are realized to publish three different kinds of evaluation results namely: computational time, the percentage of memory utilization, and percentage of CPU usage during the operation. The computational time required by Spark, Hadoop, and RDBMS with a single node and multi-node configurations are illustrated in Figure 17.



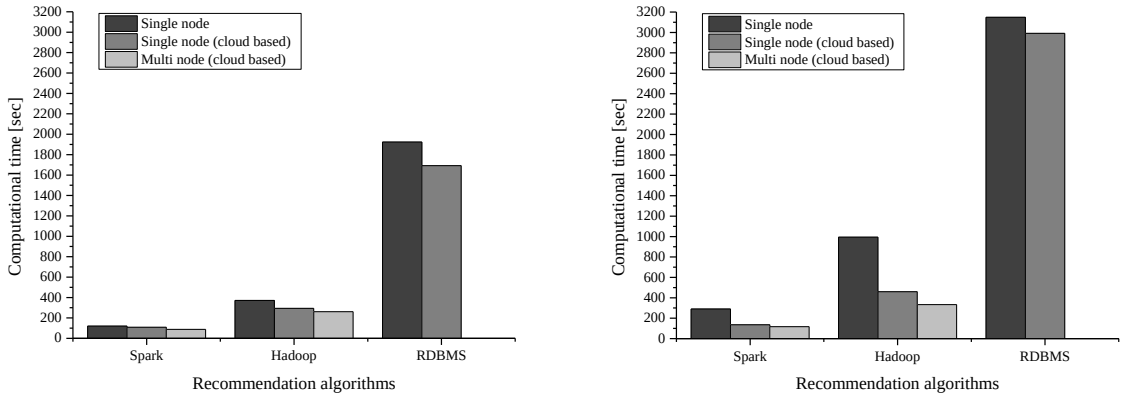


Figure 17. The computational time required by various recommendation algorithms at different configurations of data networks with various sizes of datasets: (a) 100000 ratings, (b) 2000000 ratings, (c) 6231790 ratings, and (d) 16231790 ratings.

As shown in Figure 17, Since, Spark utilizes memory-based storage for RDDs but MapReduce in Hadoop processes disk-based operations, and RDBMS not supporting parallelism and memory-based processing, it stands to reason that the performance of Spark with all the node configurations outperforms other methods at the computational time. Moreover, the computational time of all the methods has increased gradually with the increase in the size of datasets. Also, it is also seen that the cloud-based approach is effective at maintaining lesser computational time than the non-cloud based when the datasets are gradually increased. The performance ratio value shown in Table 4 over RDBMS by the Spark is higher than that by Hadoop because of larger number of iterations and calculations. Also, this ratio is decreasing in Hadoop and fluctuating in Spark for increasing dataset size, with the highest value of 22.057 for the fourth dataset on cloud-based single node cluster. Similarly, the Spark has better performance than Hadoop, with maximum speedup up to 8.17 in a multi-node cluster environment in the cloud. However, the advantage decreased as the input size increases and reached to 2.84 as shown in Table 5.

Table 4 Spark's and Hadoop's Speedup over RDBMS.

Single node cluster- Non-Cloud					Single node cluster- Cloud			
Input/DataSize(ratings)	100000	2000000	6231790	16231790	100000	2000000	6231790	16231790
Hadoop	261.18	302.52	371.24	993.97	225.12	250.66	294.04	459.72
Spark	35.19	90.455	119.49	290.52	27.47	68.282	109	135.64
RDBMS	684.61	1336	1923.72	3148.21	592.83	1196.07	1692.2	2991.9
Hadoop's Speedup	7.42199488	3.3444254	3.10687087	3.42134793	8.19512195	3.67095281	2.69761468	3.3892657
Spark's Speedup	19.4546746	14.769775	16.0994225	10.8364656	21.5809975	17.5166222	15.5247706	22.0576526

Table 5 Spark's Speedup over Hadoop.

Multi node cluster- Cloud				
Input/Data Size(ratings)	100000	2000000	6231790	16231790
Hadoop	212.11	233.39	260.79	332.91
Spark	25.943	52.007	88.02	116.88
Speedup over spark	8.17600123	4.48766512	2.96284935	2.84830595

Based on the same memory usage, Spark performs better than RDBMS and Hadoop. The reasons mainly result from the following factors:

1) Spark workloads have a higher number of disk accesses per second than Hadoop and RDBMS

2) Spark has better memory bandwidth utilization than other two

3) RDBMS have many read and write operations on disk while Mapreduce have many read but single write, in which Spark perform read and write operation on disk once a time.

Also, in Spark, task scheduling is based on an event-driven mode, but Hadoop employs heartbeat to tracking tasks, which periodically causes a few seconds delays. For some applications involved in the iterative algorithm, Hadoop is overwhelmed entirely by Spark because multiple jobs in Hadoop cannot share data and have to access HDFS frequently.

Next evaluation of the recommendation is done by the memory utilization. The observed result is presented in Figure 18.

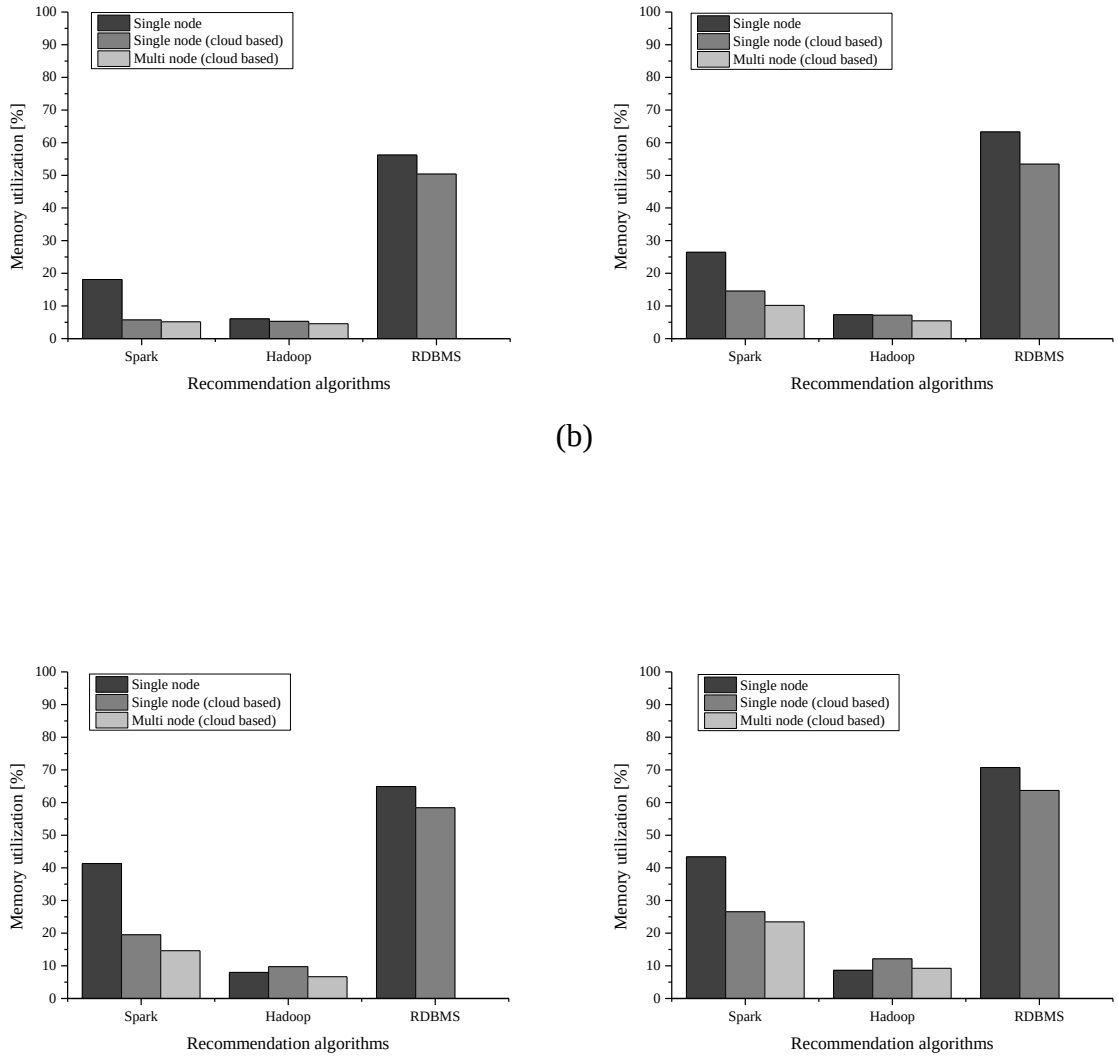


Figure 18 Percentage of memory utilization by various recommendation algorithms at different configurations of data networks with various sizes of datasets: (a) 100000 ratings, (b) 2000000 ratings, (c) 6231790 ratings, and (d) 16231790 ratings.

Here, Hadoop and RDBMS have less effect on memory utilization with an increase in the size of datasets where former and later approaches are best and least efficient among the three approaches concerning the memory utilization. However, memory utilization with Spark has been significantly increased with increase in the size of datasets. Because of more number of iterations, spark occupies more memory for newly created RDD, which effect spark's performance and also increase the CPU consumption as shown in Figure 19. At all the algorithms, a configuration with the cloud-based multi-node is best with memory

utilization as compared with non-cloud based because of its memory capacity and parallelism of task computation on different clusters.

The third evaluation of the recommendation algorithms in this study is done by percentage use of CPU during the operation. The obtained result is illustrated in the following Figure 19.

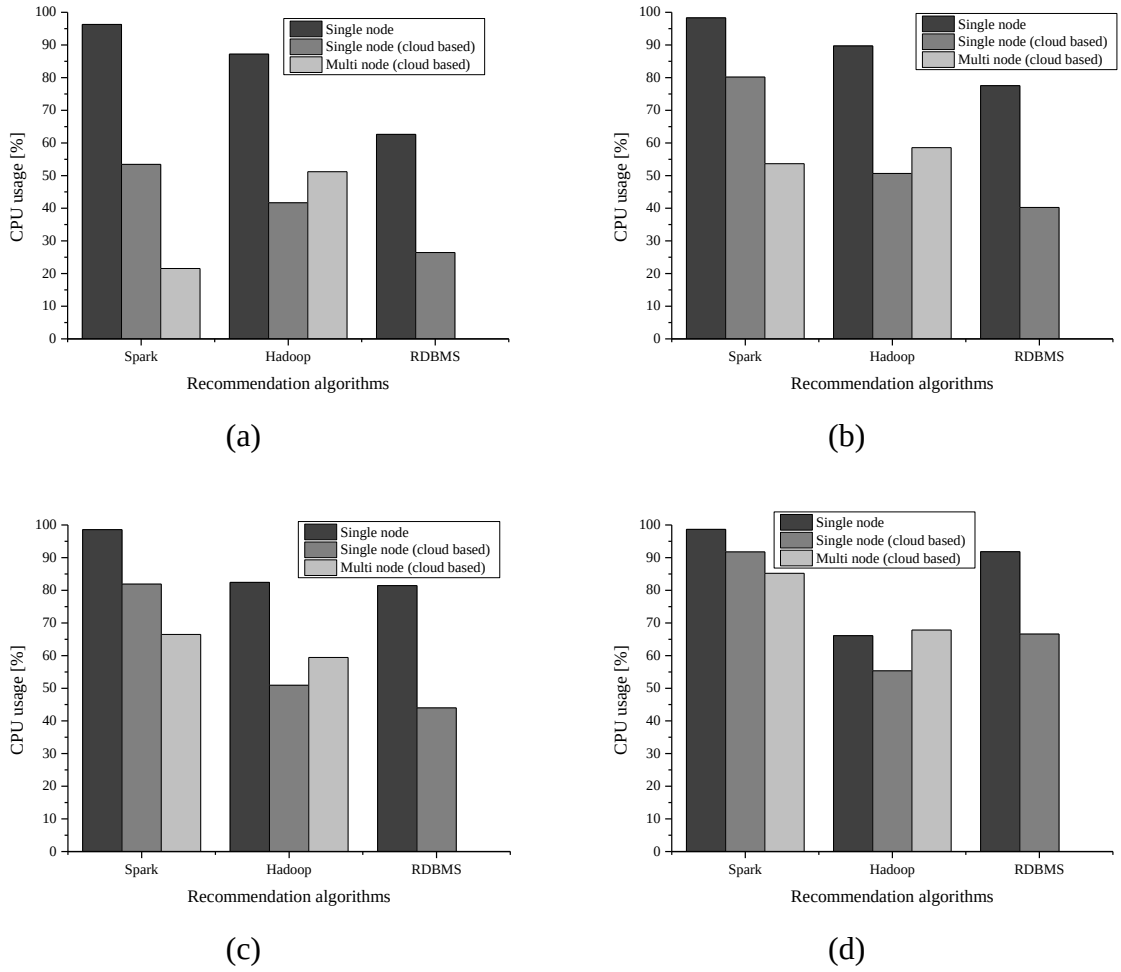


Figure 19 Percentage of CPU usage by various recommendation algorithms at different configurations of data networks with various sizes of datasets: (a) 100000 ratings, (b) 2000000 ratings, (c) 6231790 ratings, and (d) 16231790 ratings.

It is seen from Figure 19 that the amount of CPU usage is increased gradually with the increase in the size of datasets. The CPU usage of Hadoop with cloud-based single node is lesser compared to other configurations of it at all sizes of datasets. At Spark and RDBMS,

the use of cloud-based configuration has helped to reduce the amount of CPU usage at all the configurations. Overall, all the compared algorithms have mixed result with the change in network configurations and increase in size of datasets.

Lastly, the observation is done by disk I/O operations per second. It is done only on cloud environment after noticing cloud-based as more efficient than non-cloud based. This is done only between Hadoop and Spark, to compare their disk I/O for task completion. It is seen from the Figure 20 that, Spark has a higher number of disk accesses (read operation per second) per second than Hadoop on every experiment. However, write operation per second in Hadoop is slightly higher than that in spark. Overall, considering disk I/O operations as a performance metric, spark on multi-node cluster environment have faster access.

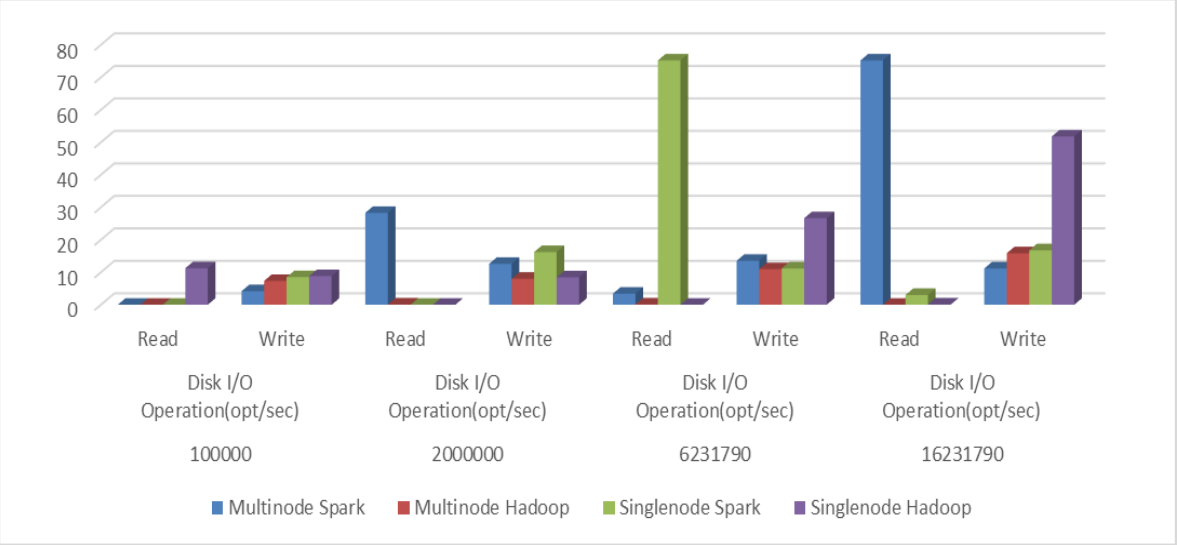


Figure 20 Graph plot of Disk I/O read and write by Spark and Hadoop on cloud-based single node and multi-node cluster with various sizes of datasets.

4.2 Accuracy of recommendation system

Experiments test whether the algorithms are useful for predicting relevant song for any item. For every item in the dataset, 10% song was randomly removed and the algorithms were used to try to recommend that removed song. This “Leave one out” methodology has been used in CF offline experiments.

The dataset is divided into training and test datasets at a 90% to 10% ratio. Ten different training and testing datasets were created for 10-fold cross validation. This method of experimentation has some limitations. These algorithms could recommend song that are very similar to or even better than the removed song and this might diminish the algorithms' performance, since these song will appear in the recommendation list before than the removed song. Even though this is a possibility, it is expected the removed song to be recommended. In order to do this, "Hit-percentage" is taken as a metric. "Hit-percentage" (HP) is defined as a metric to measure the percentage of the time the recommender algorithm correctly recommends the removed song. Here, Spark is taken as the base system for calculating accuracy of algorithm which is shown in Table 6 below.

Table 6 Accuracy of Recommendation System where hit-percentage is accuracy.

	round-1	round-2	round-3	round-4	round-5	round-6	round-7	round-8	round-9	round-10
Hit-percentage	89.09%	89.12%	90.18%	89.31%	89.20%	89.17%	89.86%	88.70%	88.79%	89.06%

As seen in Table 6, the minimum and maximum accuracy on a hit-percentage scale is 88.70% and 90.18%, respectively. At most of the times, nearly similar hit-percentage was observed with overall accuracy being 89.248%.

4.3 Statistical validation of the experimental results

To validate the reported result, Friedman as the ranking test [24] [25] and Holm [26] as the post-hoc test is performed. For the test, the null hypothesis (H_0) is set as:

a. Ranking: The means of the results of two or more algorithms are the same.
b. Post-hoc: The mean of the results of the control method and against each other groups is equal (compared in pairs).

The following tests are applied to the STAC web platform where it is assumed that the lower the result of an algorithm on a problem, the better of such algorithm [27] [28]. Here, a combination of a node configuration and network configuration in a recommendation method is assumed as an individual algorithm or a group. Hence there are eight algorithms for the ranking and post-hoc test. For example, a non-cloud based Spark with a single node is assumed an algorithm among the eight. For an evaluation measure, each algorithm has

three samples of output in one dataset. Hence, a number of the group for the test (k) is eight and number of samples (n) is 96 (12 samples per algorithm). The significance level (α) is assumed as 0.05 for all the tests.

- Computational time

  The null hypothesis for ranking was rejected by the Friedman test with p-value of 0.0021. In addition, the null hypothesis for the post-hoc test with multi node cloud based Spark as a control method yielded the following result.

  1. Spark (multi node/cloud based) vs. RDBMS (non-cloud): H_0 is rejected with p-value of 0.0012.
  2. Spark (multi node/cloud based) vs. RDBMS (cloud based): H_0 is rejected with p-value of 0.0054.
  3. Spark (multi node/cloud based) vs. Hadoop (single node/non-cloud): H_0 is rejected with p-value of 0.0013.
  4. Spark (multi node/cloud based) vs. Hadoop (single node/cloud based): H_0 is rejected with p-value of 0.00398.
  5. Spark (multi node/cloud based) vs. Hadoop (multi node/cloud based): H_0 is accepted with p-value of 0.08342.
  6. Spark (multi node/cloud based) vs. Spark (single node/non-cloud): H_0 is accepted with p-value of 0.35257.
  7. Spark (multi node/cloud based) vs. Spark (single node/cloud based): H_0 is accepted with p-value of 0.55098.

The result of the rank test suggests that the computational time required by every individual algorithm differs from each other. Moreover, post-hoc test signifies that the computational time required by multi-node cloud-based Hadoop is statistically similar to multi-node cloud-based Spark with p-value very close to the significance level (0.05). The post-hoc test for remaining Hadoop configurations and RDBMS is rejected.

- Memory utilization

  The null hypothesis for ranking was rejected by the Friedman test with p-value of 0.0021. Since, Hadoop seems efficient at memory utilization as shown in Figure 21, the multi node cloud based Hadoop is set as a control method in the null hypothesis

for the post-hoc test on memory utilization by different algorithms. The result of the post-hoc test is presented as follows:

1. Hadoop (multi node/cloud based) vs. RDBMS (non-cloud): H_0 is rejected with p-value of 0.0011.
2. Hadoop (multi node/cloud based) vs. RDBMS (cloud based): H_0 is rejected with p-value of 0.0012.
3. Hadoop (multi node/cloud based) vs. Spark (single node/non-cloud): H_0 is rejected with p-value of 0.0021.
4. Hadoop (multi node/cloud based) vs. Spark (single node/cloud based): H_0 is rejected with p-value of 0.0023.
5. Hadoop (multi node/cloud based) vs. Spark (multi node/cloud based): H_0 is accepted with p-value of 0.06829.
6. Hadoop (multi node/cloud based) vs. Hadoop (single node/non-cloud): H_0 is accepted with p-value of 0.41024.
7. Hadoop (multi node/cloud based) vs. Hadoop (single node/cloud based): H_0 is accepted with p-value of 0.23953.

Since the null hypothesis for the rank test is rejected, it is clear that the memory utilization by every individual algorithm is different. Furthermore, as in computational time, the post-host test compared with multi-node cloud-based Spark is accepted by the p-value very close to the significance level. Test with remaining Spark configurations and RDBMS is rejected.

- CPU usage

  Similar to the computational time and memory utilization, the Friedman ranking test is rejected with p-value 0.0020. In addition, as the CPU usage among the recommendation algorithms in Figure 22 shows the mixed result, cloud based RDBMS have set as a control method in the null hypothesis in the post-hoc test. The obtained statistical result is reported as follows:

  1. RDBMS (cloud-based) vs. Spark (single node/non-cloud): H_0 is rejected with p-value of 0.0050.

2. RDBMS (cloud based) vs. Spark (single node/cloud-based): H_0 is rejected with p-value of 0.00276.
3. RDBMS (cloud-based) vs. Spark (multi node/cloud-based): H_0 is accepted with p-value of 0.41024.
4. RDBMS (cloud-based) vs. Hadoop (single node/non-cloud): H_0 is rejected with p-value of 0.00303.
5. RDBMS (cloud-based) vs. Hadoop (single node/cloud-based): H_0 is rejected with p-value of 0.00150.
6. RDBMS (cloud-based) vs. Hadoop (multi node/cloud-based): H_0 is accepted with p-value of 0.40420.
7. RDBMS (cloud-based) vs. RDBMS (non-cloud): H_0 is rejected with p-value of 0.00243.

The Friedman rank test signifies that the CPU usages by various algorithms are different. Moreover, the post-hoc test suggests that the CPU usage of cloud-based RDBMS is statistically similar to multi-node cloud based Spark/Hadoop.

4.4 Analysis of the Result

As from all the observations, the performance of the spark technology is best among all. The Hadoop lies on second and RDBMS technology on third. The characteristics of dividing up of the single task into multiple tasks and executing those tasks from individual clusters make the performance of the system faster, which can be seen from the performance of Spark and Mapreduce in every experiment, and this is not found in RDBMS as it is not its property. Similarly the processing of a task on memory rather than on a disk again improves the performance, which is proven by the performance of Spark that is not available on Hadoop, where Hadoop have number of data in and out from disk while processing data causing a delay. RDBMS having property of processing data on memory taking from disk is not much significant because of its higher disk I/O overhead, memory utilization and CPU usage, making the process slower for huge data.

The use of cloud has also a prominent impact on the performance of the system, which can be seen from the cloud-based output in every graph of experimentally observed. The result

value (time in second) from all the system for the cloud-based environment is least among them all. Moreover, from memory usage, multi-node cluster in the cloud have the best result for every algorithm.

5 CONCLUSION

By comparing the experimental results for running different case studies, it is found that processing of data in-memory led spark to outperform Hadoop as Hadoop MapReduce persists back to the disk after a map or reduce action. Nonetheless, Spark needs much memory. The memory in the Spark cluster should be at least as large as the amount of data that need to process because the data has to fit into the memory for optimal performance. So, if it is to process really Big Data, Hadoop will definitely be the cheaper option since hard disk space comes at a much lower rate than memory space. On the other hand, considering Spark's benchmarks, it should be more cost-effective since less hardware can perform the same tasks much faster, especially on the cloud where compute power is paid per use.

Overall, if there is no sufficient memory and the speed is not a demanding requirement, Hadoop is a better choice. In addition, if the speed is not an important constraint and data volume is less and structured in nature then RDBMS is the best choice. For those applications, which are, time sensitive or involved in iterative algorithms and there is abundant memory available, Spark sure to be a best fit. Moreover, Spark running on cloud system having high compute power and pay per use property will be the most requirement fulfilling system.

As for future work, setting up of recommendation system on a bigger cluster with large volume of data to test the scalability of each platform will be the first priority. Also, increasing the memory capacity of the clusters to explore the influence of memory restriction on running time of Spark will be the other work. And planned to design an intelligent system that can help to choose a platform and the configuration parameters based on the applications and input data sizes to get optimized performance.

REFERENCES

[1] Manekar and A. a. Pradeepini, "A Review on Cloud -based Big Data Analytics," *ICSES Journal on Computer Networks and Communication (IJCNC,* 2015.

[2] Assuncao, M. D., Calheiros, R. N., Bianchi, S. a. Netto and M. A. S., "Big Data Computing and Clouds," *: Trends and Future Directions. J. Parallel Distrib. Computing,* 2015.

[3] R. Raphael1 and R. Kumar T2 , "Big Data, RDBMS and HADOOP - A Comparative Study," *International Journal of Science and Research (IJSR) ,* 2014.

[4] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker and a. I. Stoica, "Spark: cluster computing with working sets," in *Proceedings of the 2nd USENIX Conference on Hot Topics in cloud computing*, Berkeley, 2010.

[5] " Apache Software Foundation.," june 2014. [Online]. Available: http://hadoop.apache.org/hadoop. [Accessed june 2014].

[6] K. Shvachko, K. Hairong, S. Radia and a. R. Chansler, "The Hadoop Distributed File System," *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies,* 2010.

[7] H. Karau, A. Konwinski, P. Wendell and a. M. Zaharia, Learning Spark, Sebastopol: O'Reilly Media, 2015.

[8] N. Islam, S. Sharmin, M. Wasi-ur-Rahman, X. Lu, D. Shankar and D. K. Panda, "Performance characterization and acceleration of in-memory file systems for Hadoop and Spark applications on HPC clusters," in *IEEE International Conference on Big Data*, 2015.

[9] "Apache Spark: core concepts, architecture and internals," 03 03 2016. [Online]. Available: http://datastrophic.io/core-concepts-architecture-and-internals-of-apache-spark/. [Accessed 03 03 2016].

[10] H. Yu and D. Wang, "Mass Log Data Processing and Mining Based on Hadoop and Cloud Computing," in The 7th International conference on Computer Science and Education, Melbourne, Australia, 2012.

[11] Xiangsheng and Kong, "Scientific data mining and processing using MapReduce in cloud," Journal of Chemical and Pharmaceutical Research, 2014.

[12] Patil and Viki, "Study of Data Mining algorithm in cloud computing using MapReduce Framework," Journal of Engineering, Computers & Applied Sciences, 2013.

[13] U. Gov, "minerals.usgs.gov," [Online]. Available: minerals.usgs.gov.

[14] Wu, X., Zhu, X., Wu and G. a. Ding, "Data Mining with Big Data," School of Computer Science and Information Engineering, Hefei University of Technology, China, 2013.

[15] X. Wu, X. Zhu, G.-Q. Wu and a. W. Ding, "Data mining with big data," IEEE Transaction on Knowledge and Engineering, 2014.

[16] Arora and S. G. a. Rohan, "Comparing Apache Spark and Map Reduce with Performance Analysis using K-Means," International Journal of Computer Applications (0975 – 8887) Volume 113 – No. 1, 2015.

[17] J. Shi, Y. Qiu, U. F. Minhas, L. Jiao, C. Wang, B. Reinwald and a. F. O.¨ zcan, "MapReduce vs. Spark for Large Scale Data Analytics," Proceedings of the VLDB Endowment, Vol. 8, No. 13, 2015.

[18] S. Landset, T. M. Khoshgoftaar and A. N. R. a. T. Hasanin, "A survey of open source tools for machine learning with big data in the Hadoop ecosystem," Landset et al. Journal of Big Data, 2015.

[19] J. P. Verma, B. Patel and a. A. Patel, "Big Data Analysis: Recommendation System with Hadoop Framework," 2015 IEEE International Conference on Computational Intelligence &

Communication Technology, 2015.

[20] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker and a. I. Stoica, "Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing," in 9th USENIX Conference on Networked Systems Design and Implementation, Berkeley, 2012.

[21] "Spark Overview," 2015. [Online]. Available: http://spark.apache.org/.

[22] J. A. Scott, Getting Started with Apache Spark, First Edition ed., United States of America: MapR Technologies, Inc., 350 Holger Way, San Jose, CA 95134, 2015.

[23] "Cloud DataProc," [Online]. Available: https://cloud.google.com/.

[24] M. Friedman, "The Use of Ranks to Avoid the Assumption of Normality Implicit in the Analysis of Variance," 1937, p. 675–701.

[25] J. Hodges and E. Lehmann, "Rank Methods for Combination of Independent Experiments in Analysis of Variance," in The Annals of Mathematical Statistics, vol. 33, Institute of Mathematical Statistics, p. pp. 482–497..

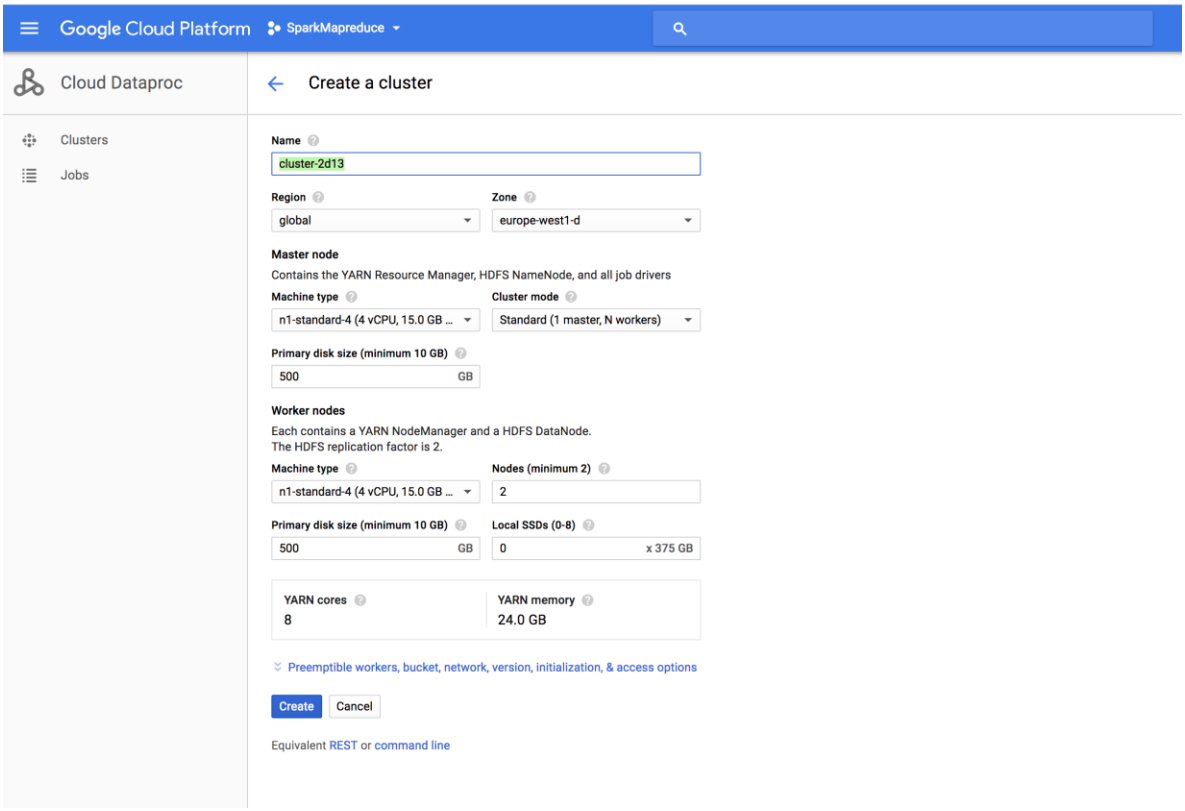
[26] S. Holm, "A Simple Sequentially Rejective Multiple Test Procedure," in Scandinavian Journal of Statistics, vol. 6, WileyBoard of the Foundation of the Scandinavian Journal of Statistics, 1979, p. pp. 65–70.

[27] I. Rodriguez-Fdez, A. Canosa, M. Mucientes and A. Bugarin, "STAC: A web platform for the comparison of algorithms using statistical tests," in IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), 2015.

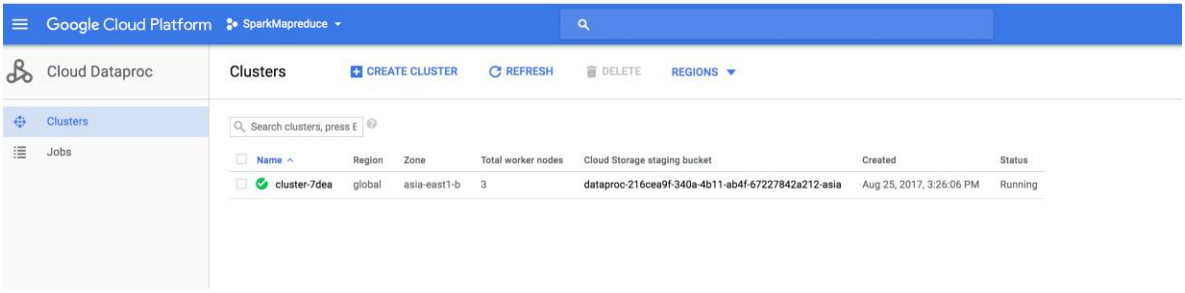
[28] "STAC Web Platform.," [Online]. Available: http://tec.citius.usc.es/stac/index.html.. [Accessed 2 11 2017].

APPENDICES

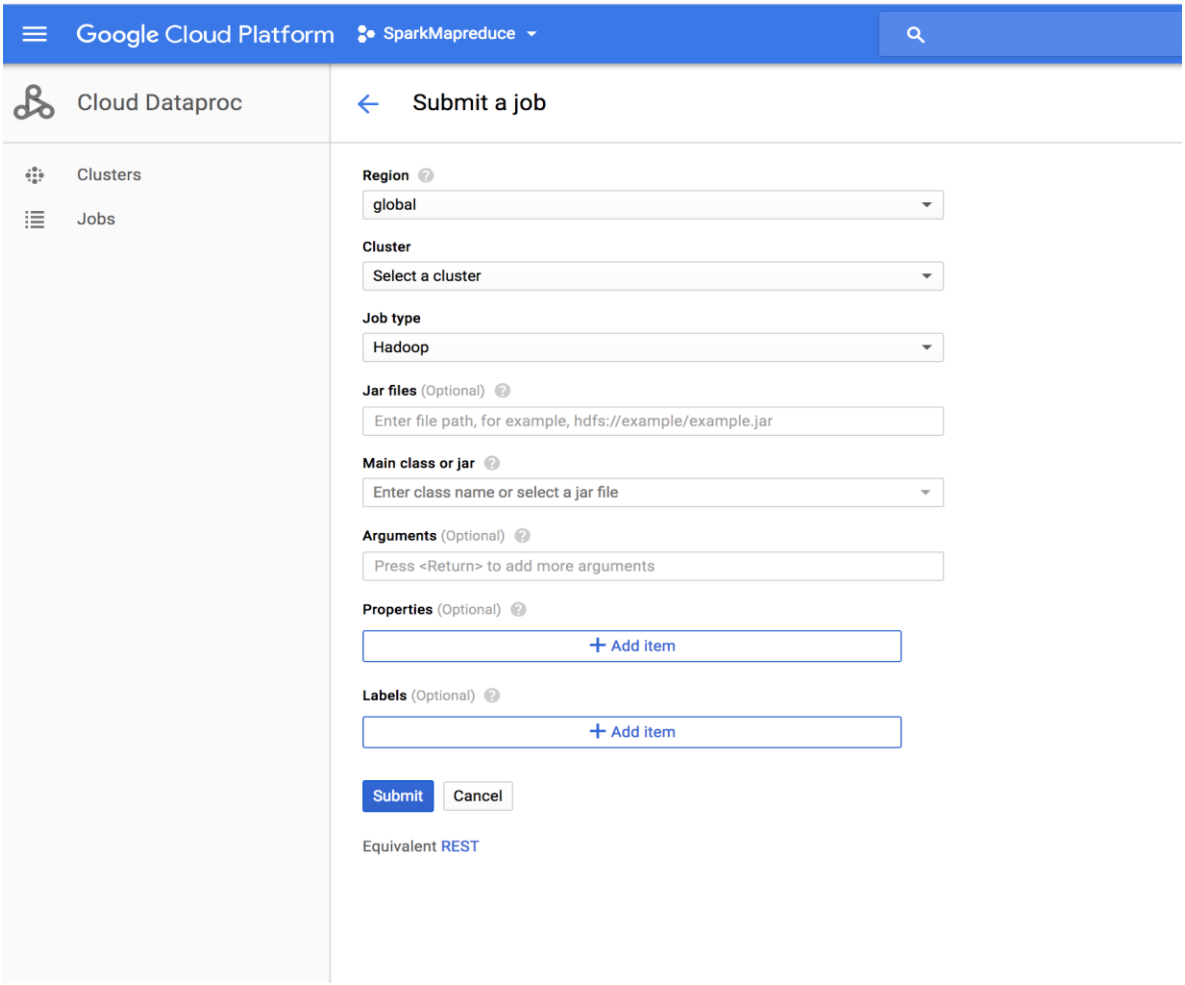
Appendix A - Creating a cluster in google cloud



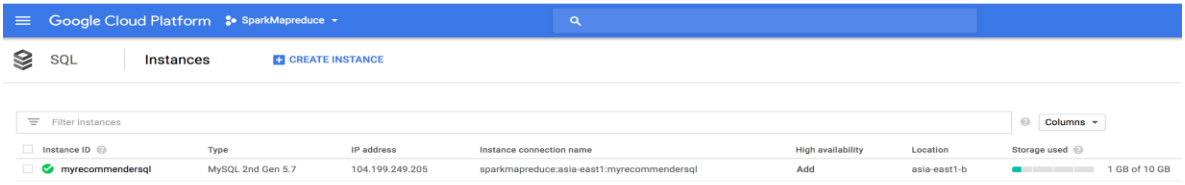
Appendix B - List of clusters



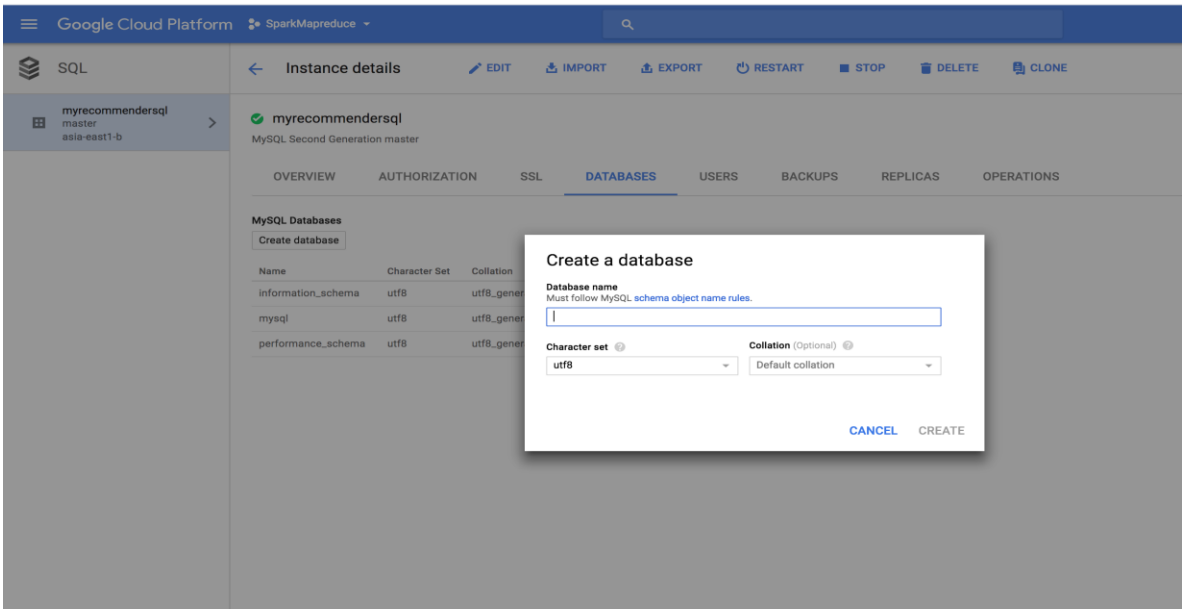
Appendix C - Running a Job in google cloud



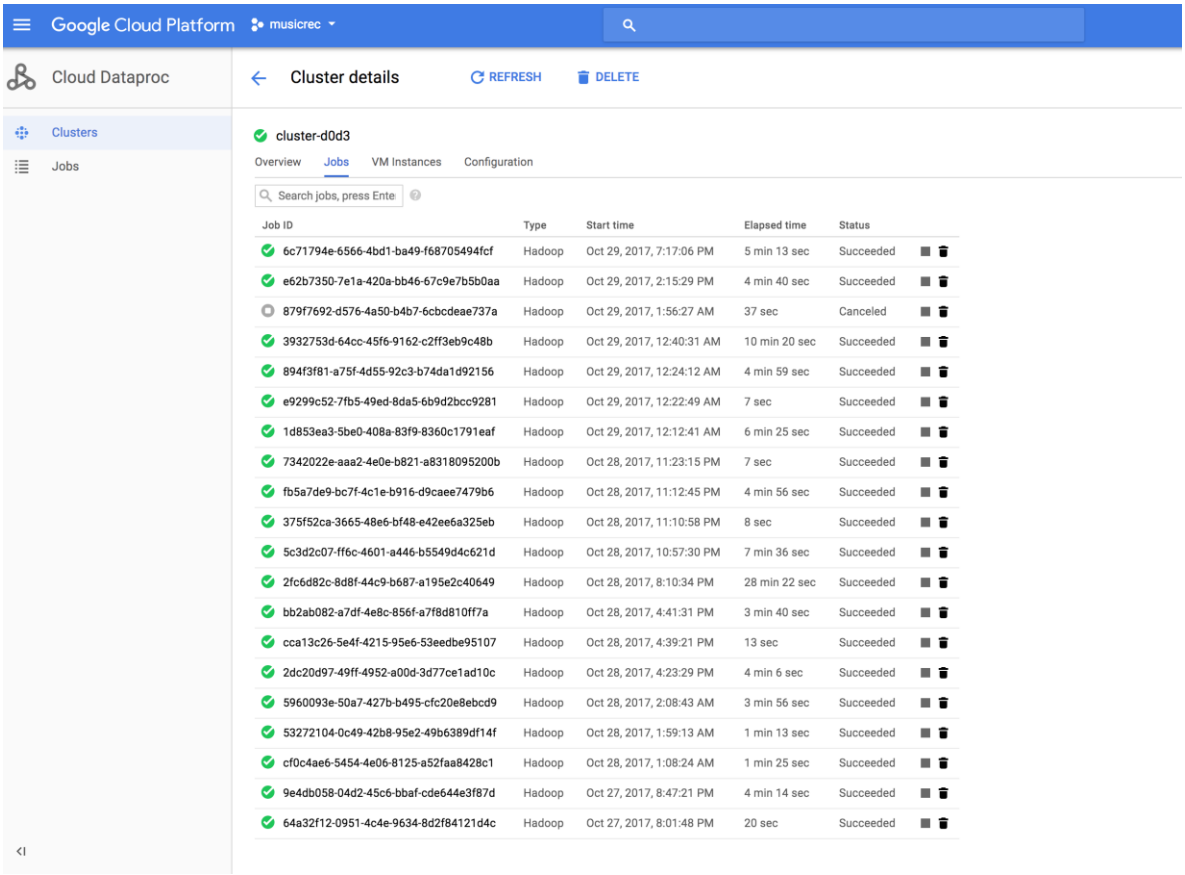
Appendix D - List of MySQL Instance

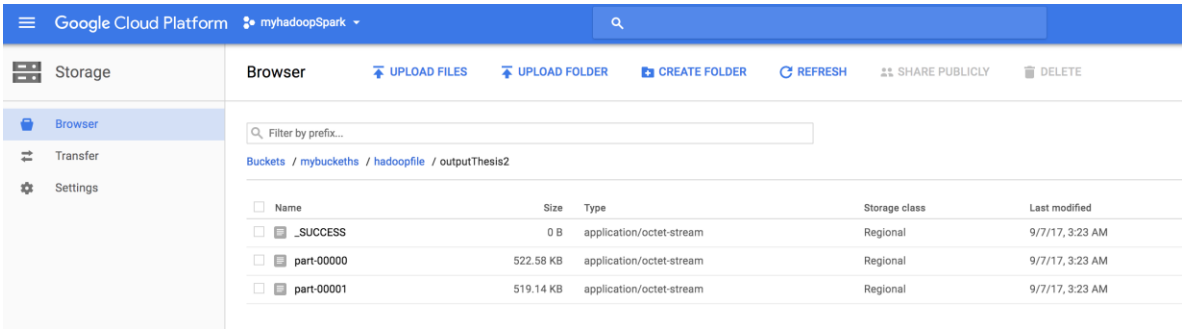


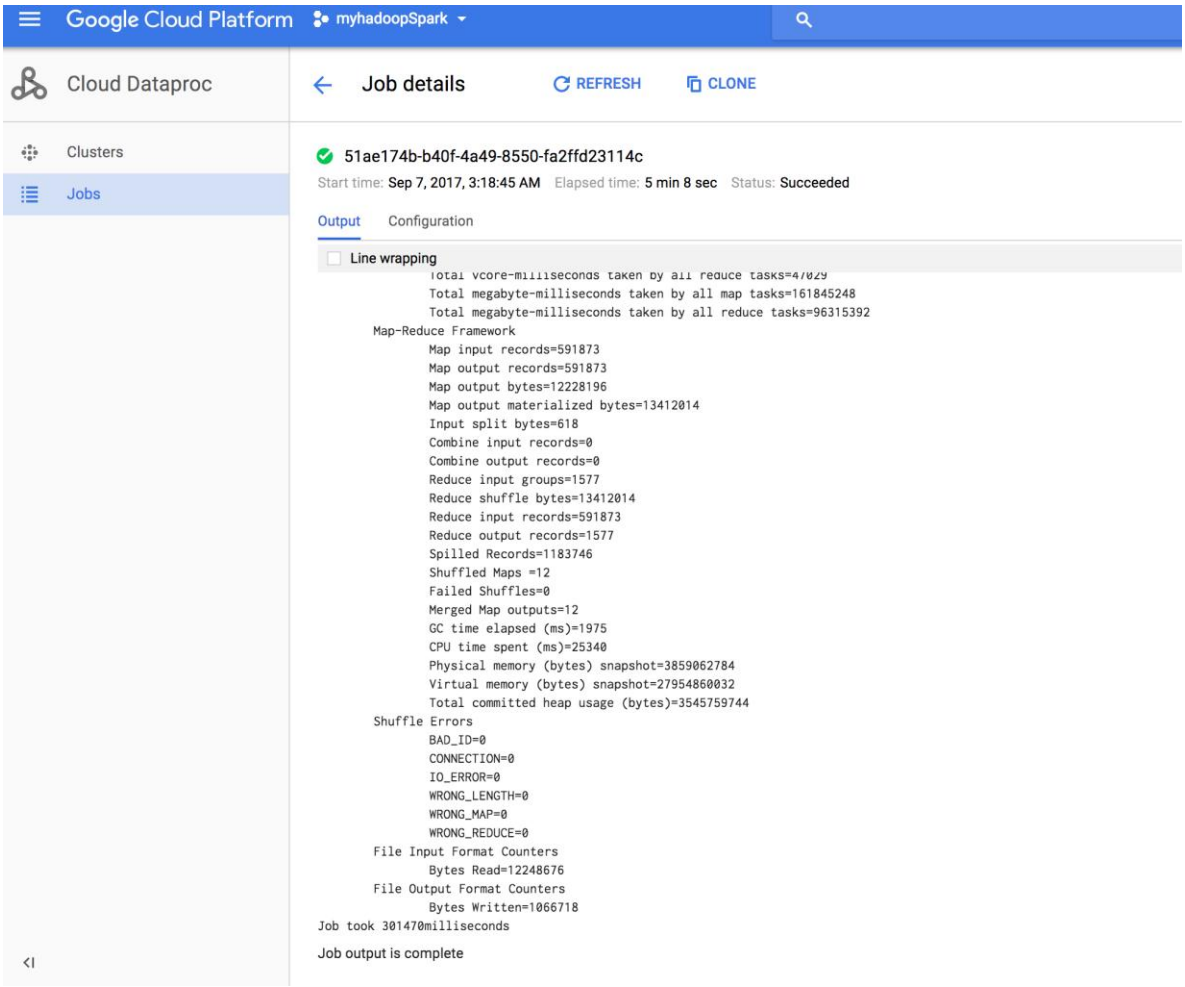
Appendix E - Creating a database in MySQL in Google cloud



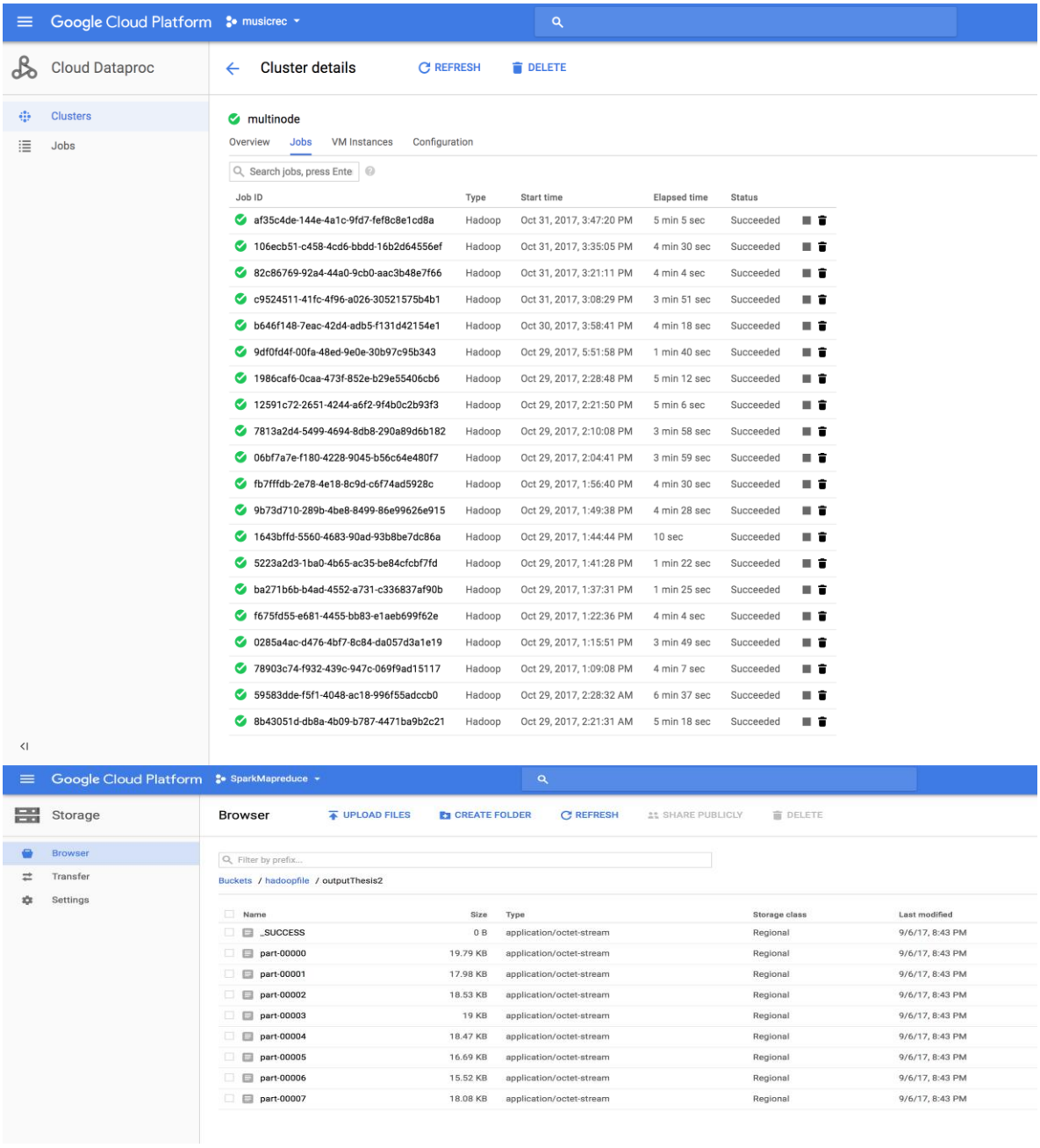
Appendix F - Mapreduce on cloud with single cluster

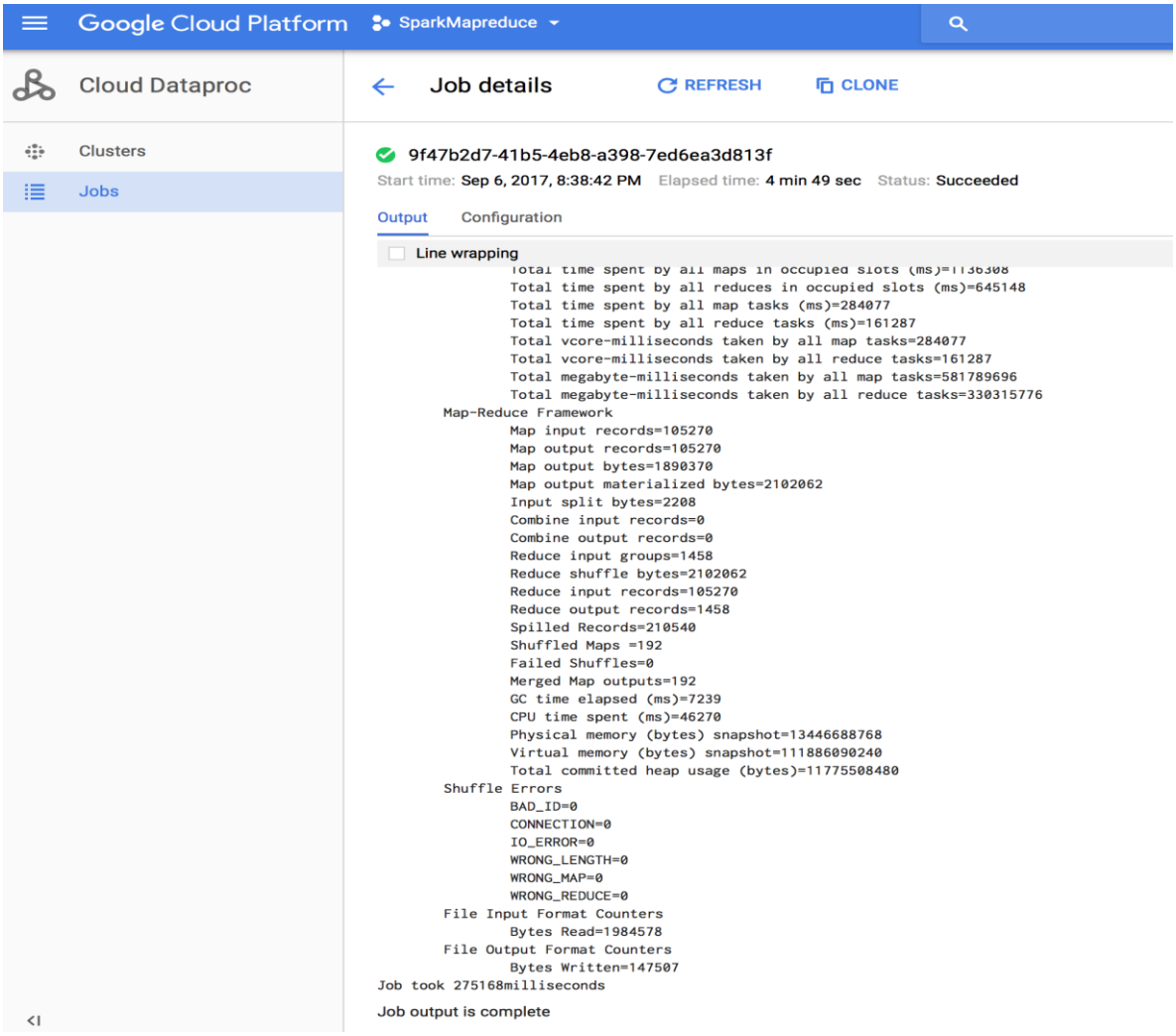



Google Cloud Platform
myhadoopSpark
Storage
Browser
UPLOAD FILES
UPLOAD FOLDER
CREATE FOLDER
REFRESH
SHARE PUBLICLY
DELETE
Browser
Transfer
Settings
Filter by prefix...
Buckets / mybucketh / hadoopfile / outputThesis2
Name
Size
Type
Storage class
Last modified
_SUCCESS
0 B
application/octet-stream
Regional
9/7/17, 3:23 AM
part-00000
522.58 KB
application/octet-stream
Regional
9/7/17, 3:23 AM
part-00001
519.14 KB
application/octet-stream
Regional
9/7/17, 3:23 AM

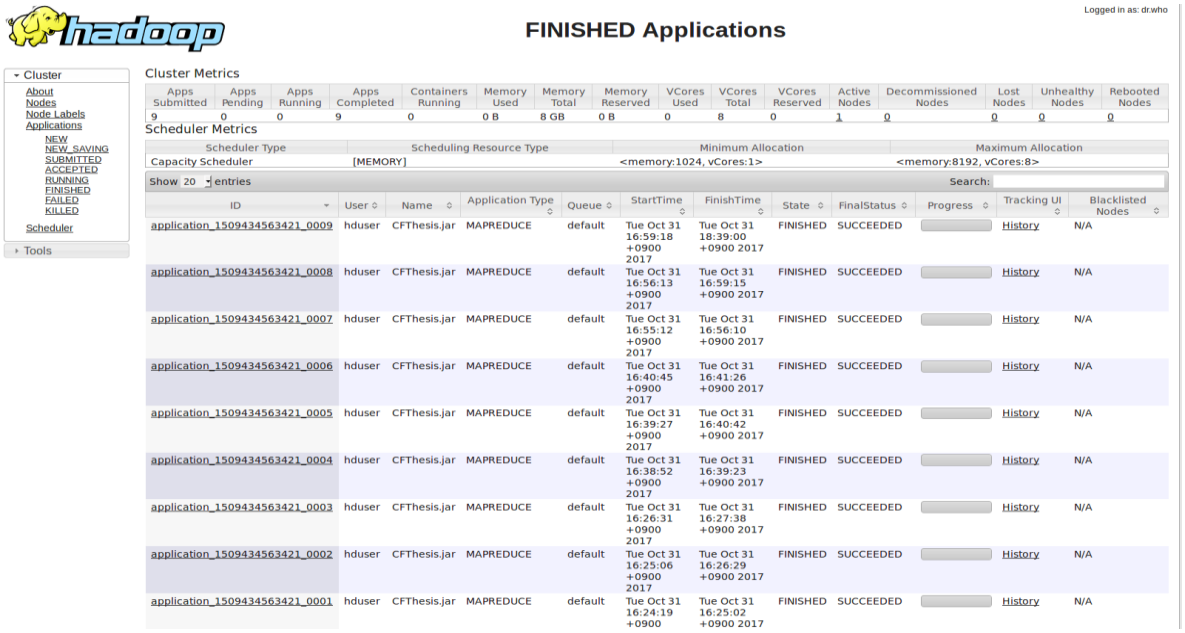

Google Cloud Platform
myhadoopSpark
Cloud Dataproc
Job details
REFRESH
CLONE
Clusters
Jobs
51ae174b-b40f-4a49-8550-fa2ffd23114c
Start time: Sep 7, 2017, 3:18:45 AM
Elapsed time: 5 min 8 sec
Status: Succeeded
Output
Configuration
Line wrapping
Total megabyte-milliseconds taken by all map tasks=161845248
Total megabyte-milliseconds taken by all reduce tasks=96315392
Map-Reduce Framework
Map input records=591873
Map output records=591873
Map output bytes=12228196
Map output materialized bytes=13412014
Input split bytes=618
Combine input records=0
Combine output records=0
Reduce input groups=1577
Reduce shuffle bytes=13412014
Reduce input records=591873
Reduce output records=1577
Spilled Records=1183746
Shuffled Maps =12
Failed Shuffles=0
Merged Map outputs=12
GC time elapsed (ms)=1975
CPU time spent (ms)=25340
Physical memory (bytes) snapshot=3859062784
Virtual memory (bytes) snapshot=27954860032
Total committed heap usage (bytes)=3545759744
Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Input Format Counters
Bytes Read=12248676
File Output Format Counters
Bytes Written=1066718
Job took 301470milliseconds
Job output is complete

Appendix G - Mapreduce on cloud with multi-node cluster



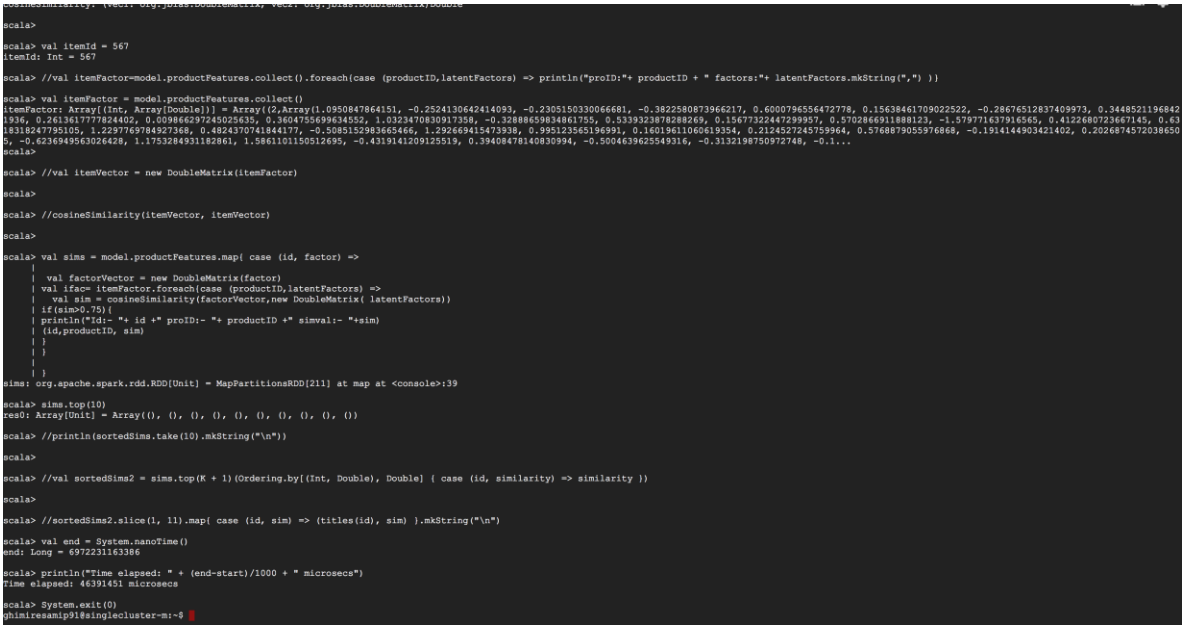


Appendix H - Mapreduce on Non-cloud based system

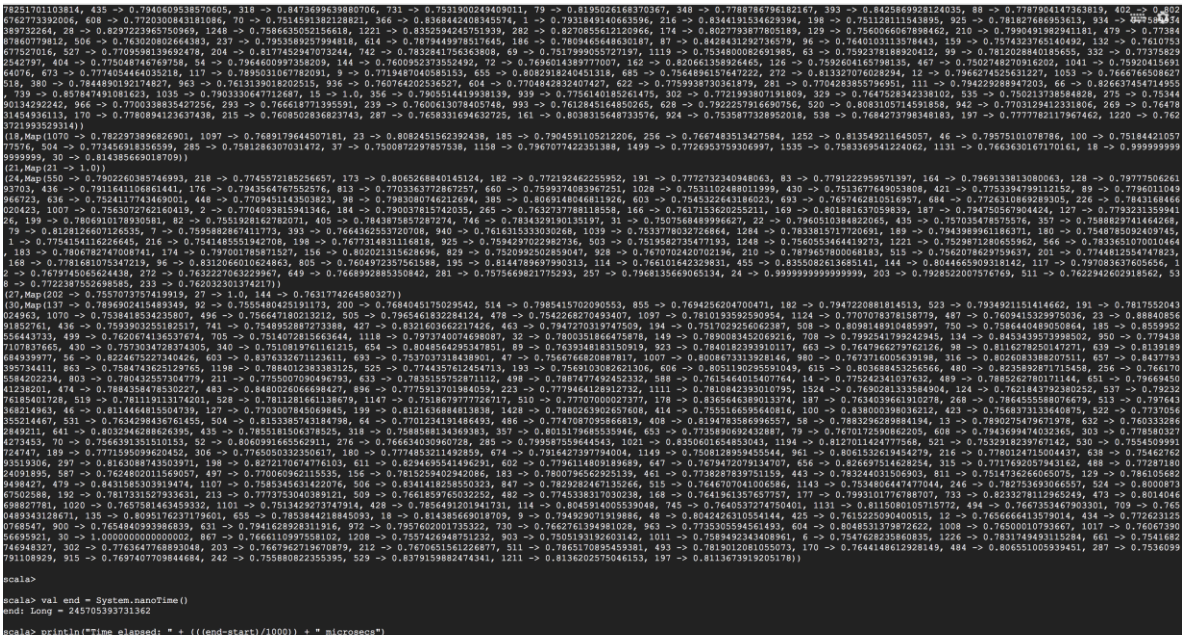


```
        File System Counters
                FILE: Number of bytes read=37689932
                FILE: Number of bytes written=75736089
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=34543390
                HDFS: Number of bytes written=3207509
                HDFS: Number of read operations=9
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Launched map tasks=2
                Launched reduce tasks=1
                Data-local map tasks=2
                Total time spent by all maps in occupied slots (ms)=30828
                Total time spent by all reduces in occupied slots (ms)=10657
                Total time spent by all map tasks (ms)=30828
                Total time spent by all reduce tasks (ms)=10657
                Total vcore-milliseconds taken by all map tasks=30828
                Total vcore-milliseconds taken by all reduce tasks=10657
                Total megabyte-milliseconds taken by all map tasks=31567872
                Total megabyte-milliseconds taken by all reduce tasks=10912768
        Map-Reduce Framework
                Map input records=1575414
                Map output records=1575414
                Map output bytes=34539098
                Map output materialized bytes=37689938
                Input split bytes=196
                Combine input records=0
                Combine output records=0
                Reduce input groups=1682
                Reduce shuffle bytes=37689938
                Reduce input records=1575414
                Reduce output records=1682
                Spilled Records=3150828
                Shuffled Maps =2
                Failed Shuffles=0
                Merged Map outputs=2
                GC time elapsed (ms)=900
                CPU time spent (ms)=10830
                Physical memory (bytes) snapshot=591147008
                Virtual memory (bytes) snapshot=5814398976
                Total committed heap usage (bytes)=392306688
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=34543194
        File Output Format Counters
                Bytes Written=3207509
Job took 452208milliseconds
hduser@master:~/ThesisCode/CFMapreduce$
```

Appendix I - Spark on cloud



Appendix J - Spark on non-cloud



Appendix K – Output recommendation list from all the methods

```
1 1 0.9999999999999999
1 2 0.9487373941786248
1 3 0.9132997212006829
1 4 0.9429068878509639
1 5 0.9613638498709224
1 6 0.9551193973874768
1 7 0.948915528102488
1 8 0.9600459451420741
1 9 0.9387445181517423
1 10 0.943039377274702
1 11 0.9573860685968381
1 12 0.960866623892328
1 13 0.9265809768445075
1 14 0.9373408420349089
1 15 0.9456641485556644
1 16 0.9446651742249216
1 17 0.940529134897858
1 18 0.8837209302325583
1 19 0.9333110679927804
1 20 0.9062803873487921
1 21 0.9254588973428826
1 22 0.9675425840658004
1 23 0.9593277108297823
1 24 0.9445706635952916
1 25 0.9396933787226742
1 26 0.9473091158522402
1 27 0.9191908352058211
1 28 0.9723951205383821
1 29 0.9352010898538022
1 30 0.9424204132773988
1 31 0.9643669449985898
1 32 0.9223941966146801
1 33 0.9378785237170711
1 34 0.8601793595076359
1 35 0.8918244238879707
1 36 0.9395834997845066
1 37 0.9133691346759317
1 38 0.9515646322979401
1 39 0.9527600899802233
1 40 0.9313248452425006
1 41 0.9471477966965145
1 42 0.9386701957784107
1 43 0.9287386430320962
1 44 0.9518082739429662
1 45 0.9652445408881114
```

1	767	1337	680	752	253	688	369	1009	1067	413	
453	1407	1021	574	1135	1029	299	769	547	585	743	
644	543	1001	624	370	312	325	262	20	985	305	
260	1157	734	473	989	1034	149	1061	950	572	826	
774	424	400	1540	1227	394	976	103	406	444	877	
1229	974	1210	422	455	930	365	53	231	396	91	
738	1074	1267	309	1128	17	697	886	107	145	1245	
758	345	13	563	284	554	464	1160	338	240	565	
1335	93	475	307	552	873	721	143	679	787	109	
387	871	864	428	763	895	259	462	1045	343	19	
323	84	235	682	271	242	824	1115	783	541	268	
301	118	185	347	220	51	433	286	219	741	491	
354	451	40	550	408	1012	477	592	415	431	411	
1047	1090	279	62	33	42	288	295	662	761	640	
1005	1263	239	468	749	880	1283	567	419	363	675	
402	147	844	1209	66	48	266	790	123	1014	620	
466	521	39	756	244	248	121	538	808	116	1315	24
	246	590	556	1070	213	789	86	840	745	255	
105	1326	233	794	15	934	1063	1065	990	73	167	
693	952	280	316	518	60	200	358	727	941	277	
967	596	792	457	943	859	932	815	202	525	820	68
	170	293	1098	321	378	125	718	637	1007	1016	
569	26	846	165	332	1089	833	376	127	356	1052	8
	772	4	660	190	226	112	956	134	55	613	
1069	628	237	110	642	420	194	619	583	921	879	
381	1397	695	505	211	1113	765	725	11	866	392	
189	448	527	831	310	150	1188	217	88	516	509	
1117	334	181	291	486	949	514	709	215	196	204	
101	558	99	77	82	484	1036	549	163	501	156	
1221	1203	493	154	578	510	97	273	714	222	631	


-> 0.8090413253827367, 531 -> 0.8087972481705905, 84 -> 0.8011577158993727, 154 -> 0.7811781022817301, 1024 -> 0.760
288622431127, 408 -> 0.7914045983688249, 58 -> 0.7848975687375069, 381 -> 0.7543206569452798, 1018 -> 0.78486395834
75568, 13 -> 0.7600139661565923, 532 -> 0.7689357489766557, 482 -> 0.7660735089573577, 435 -> 0.7623936674402745, 65
9 -> 0.772666000771112, 1045 -> 0.7726988605751524, 318 -> 0.8061970493139112, 357 -> 0.8070528011226854, 707 -> 0.
7572647156381923, 303 -> 0.7696325916447639, 70 -> 0.8135436005667673, 52 -> 0.7780005230777707, 276 -> 0.7938638098
265711, 285 -> 0.8006720434917748, 258 -> 0.7723556907285852, 61 -> 0.8130192086960605, 958 -> 0.7827201659505338,
10 -> 0.8089132978768347, 521 -> 0.7735213955498777, 557 -> 0.7592830232350399, 306 -> 0.8201430487433734, 180 -> 0.
8125982972076534, 1149 -> 0.7647343339359898, 207 -> 0.7943736924371112, 961 -> 0.7567029321463963, 216 -> 0.825244
9374343346, 979 -> 0.7564659094344198, 629 -> 0.7502731138516606, 297 -> 0.7869103385132307, 1176 -> 0.7525533743076
0287, 503 -> 0.7959937582155585, 198 -> 0.7654610783839005, 19 -> 0.7528782494043782, 174 -> 0.7534846773299275, 488
-> 0.7718426778713182, 811 -> 0.7649728173864354, 443 -> 0.7609321956919479, 479 -> 0.8125977952386205, 847 -> 0.7
6988782491449779, 165 -> 0.7857305792536085, 506 -> 0.7756078769762506, 515 -> 0.7584379340964338, 246 -> 0.75661525
52733741, 533 -> 0.7716357651097562, 614 -> 0.8028806746805696, 60 -> 0.8117980268976488, 213 -> 0.8291553122745675
, 509 -> 0.8291953744820587, 482 -> 0.7869955408400057, 805 -> 0.7610080175863592, 464 -> 0.8052621288205609, 132 ->
0.7600426614938056, 1020 -> 0.8154537499398342, 428 -> 0.8129973660541596, 114 -> 0.7629682613048648, 527 -> 0.821
6394395841515, 536 -> 0.7888107269325119, 1119 -> 0.7585474990515547, 99 -> 0.7798503761930446, 45 -> 0.77303109614
94525, 467 -> 0.7787473906774438, 153 -> 0.7674660671482294, 162 -> 0.8215022574374232, 709 -> 0.8241810240741273,
135 -> 0.8230880911865998, 655 -> 0.8045787536261211, 9 -> 0.8100041421515208, 57 -> 0.7694673075833424, 272 -> 0.7
8038864953682O3, 317 -> 0.8054717366541969, 658 -> 0.7575087688334654, 12 -> 0.7961867392708858, 93 -> 0.756525665 2
489624, 972 -> 0.7922687865361493, 649 -> 0.7581163449244299, 730 -> 0.7953907711817917, 963 -> 0.7963414256582477,
936 -> 0.7674579196303087, 604 -> 0.7538116003978476, 712 -> 0.7683798752563826, 111 -> 0.7786387238409457, 694 ->
0.8044023163333538, 903 -> 0.7564975923432696, 1011 -> 0.7514095737248484, 661 -> 0.8133158955014714, 203 -> 0.755
2497323286871, 275 -> 0.7973700575704558, 311 -> 0.7813245965449357, 1172 -> 0.7635907079486552, 1558 -> 0.77160492
7085448A, 511 -> 0.8152748428051116, 493 -> 0.8179318272617693, 269 -> 0.7904131934607534, 170 -> 0.835533709920023
8, 484 -> 0.7778845164631955, 242 -> 0.7956701494739348, 251 -> 0.8129973961540444, 529 -> 0.7690075300694777, 1211
-> 0.7573808576192448, 197 -> 0.8290108592294474))
(16,Map(173 -> 0.7511018553087307, 191 -> 0.8165731472416369, 508 -> 0.7675200466730039, 176 -> 0.7526915837497289,
194 -> 0.7528772340031784, 813 -> 0.7581091860251045, 367 -> 0.7682996349973019, 1067 -> 0.7535873306844653, 1312
-> 0.7552007156509528, 558 -> 0.7501410811332082, 2 -> 0.7903801995793256, 657 -> 0.7636967236595478, 498 -> 0.7654
852915940747, 507 -> 0.7667888996631107, 202 -> 0.7551205567519B5, 151 -> 0.7664062489537757, 142 -> 0.773940407039
0836, 223 -> 0.7710301826518194, 483 -> 0.7936860517538249, 115 -> 0.7659698911478373, 163 -> 0.7654747342973657, 4
59 -> 0.7850303205602405, 100 -> 0.7500370255713487, 208 -> 0.7610311710872063, 199 -> 0.7581343764422144, 531 -> 0
.7538580633625643, 845 -> 0.7556043545467276, 73 -> 0.7607778937132317, 55 -> 0.7666512684948495, 154 -> 0.75014711
06939661, 58 -> 0.7560433596419733, 471 -> 0.7554866618394723, 417 -> 0.7674526108704592, 7 -> 0.7543095788698128, 1
6 -> 1.0000000000000002, 198 -> 0.8063633830569915, 165 -> 0.7612493968149493, 1137 -> 0.7676934861428360, 1119 -> 0
.7617566575811636, 12 -> 0.7873627437209133, 425 -> 0.7719869560080513, 631 -> 0.7877700908101716, 1011 -> 0.7509332
815910866, 248 -> 0.7734060248958260, 275 -> 0.7504409048530l, 293 -> 0.7723834644196014, 251 -> 0.7567794948672650)
)
(18,Map(182 -> 0.7547918099157225, 1070 -> 0.7619887979785259, 23 -> 0.8016790389237631, 185 -> 0.795112796819694,
32 -> 0.7882236113301O9, 134 -> 0.7714115505986221, 480 -> 0.7687016036820646, 1252 -> 0.7639458769896117, 489 ->
0.7522578999577885, 956 -> 0.7802721719317217, 483 -> 0.7686914606429848, 474 -> 0.7697504869140854, 127 -> 0.75018
16990987038, 46 -> 0.8448326289662, 1428 -> 0.7567483314701412, 504 -> 0.7588288798108346, 13 -> 0.7520663022377O1
2, 1395 -> 0.7648321269494249, 285 -> 0.7628507787504732, 1158 -> 0.7846832295495565, 1499 -> 0.763778757054614S, 1
535 -> 0.7643738281922836, 647 -> 0.7527487116902561, 1107 -> 0.7517017801219024, 479 -> 0.7630987770135954, 18 ->
1.0, 48 -> 0.7554248734713385, 30 -> 0.8123507021201638))
(20,Map(137 -> 0.7775866919787162, 505 -> 0.7516529628181328, 23 -> 0.7594156628888814, 427 -> 0.7569277179859291,
89 -> 0.8057100164194334, 654 -> 0.7654558056544509, 116 -> 0.8048058748858193, 663 -> 0.7548000603306254, 20 -> 0.9
999999999999999, 657 -> 0.7571395115212952, 606 -> 0.799184919819459, 480 -> 0.7665877293433887, 193 -> 0.754261501
5555897, 498 -> 0.7579000432322447, 14 -> 0.7833049375274114, 238 -> 0.7796313882508408, 489 -> 0.7553199477921501,
474 -> 0.7565180573059255, 124 -> 0.7856542694592608, 483 -> 0.7860907179657445, 178 -> 0.7577834926031982, 127 ->
0.7566375007183O3, 513 -> 0.7646199011921B2, 199 -> 0.7545058404754401, 608 -> 0.7600028101295586, 52 -> 0.7640187
110235299, 70 -> 0.7626707875349649, 303 -> 0.7721223587967163, 306 -> 0.7619712928140791, 297 -> 0.7834818428374400
8, 961 -> 0.7720674329949033, 647 -> 0.7931522729095353, 811 -> 0.7578605028813945, 506 -> 0.7523070615371866, 479
-> 0.7798578884906626, 246 -> 0.7510716231922958, 464 -> 0.7643173408688932, 473 -> 0.7904718782690748, 1020 -> 0.75
9188599667151G, 114 -> 0.7555844853778493, 428 -> 0.7641522021556l, 135 -> 0.7523312545412488, 93 -> 0.755381690965
98863, 694 -> 0.7525187605009847, 170 -> 0.7678015265487772, 197 -> 0.7903129410313946, 529 -> 0.7618259508214974))

scala>

scala> val end = System.nanoTime()
end: Long = 385215683406233

scala> println("Time elapsed: " + (((end-start)/1000)) + " microsecs")
Time elapsed: 31608942 microsecs

scala> System.exit(0)
ghimiresamip91@singlecluster-m:~$
ghimiresamip91@singlecluster-m:~$