DATA STORY- Q & A USERGROUP ANALYSER

A report submitted to

M S RAMAIAH INSTITUTE OF TECHNOLOGY

Bengaluru

for fulfilling the requirement of

Bachelor of Engineering (B.E) in Information Science and Engineering

by

CHANDAN VENKATESH (USN- 1MS11IS021)
DEEKSHITH KUMAR (USN- 1MS11IS025)
GANESH MADHAV R (USN- 1MS11IS031)
SANJAY T (USN- 1MS11IS122)

under the guidance of

Dr. Mydhili K. Nair

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

M S RAMAIAH INSTITUTE OF TECHNOLOGY

May 2015

Department of Information Science and Engineering
M S Ramaiah Institute of Technology
Bengaluru - 54

CERTIFICATE

This is to certify that **Chandan Venkatesh (USN- 1MS11IS021), Deekshith Kumar (USN- 1MS11IS025), Ganesh Madhav R (USN- 1MS11IS031)** and **Sanjay T (USN- 1MS11IS122)** who were working for their B.E project under my guidance, have completed the work as per my satisfaction with the topic **DATA STORY- Q & A USERGROUP ANALYZER**. To the best of my understanding the work to be submitted in dissertation does not contain any work, which has been previously carried out by others and submitted by the candidates for themselves for the award of any degree anywhere.

(Guide)
Dr. Mydhili K. Nair
Assistant Professor, Dept. of ISE

(Head of the Department)
Dr. Vijaya Kumar B P
Professor & Head, Dept. of ISE

(Examiner 1)

Name

Signature

(Examiner 2)

Department of Information Science and Engineering
M S Ramaiah Institute of Technology
Bengaluru - 54

DECLARATION

We hereby declare that the entire work embodied in this B.E. Project report has been carried out by us at M S Ramaiah Institute of Technology under the supervision of Dr. Mydhili K. Nair. This Project report has not been submitted in part or full for the award of any diploma or degree of this or any other University.

Chandan Venkatesh (USN- 1MS11IS021)
Deekshith Kumar (USN- 1MS11IS025)
Ganesh Madhav R (USN- 1MS11IS031)
Sanjay T (USN- 1MS11IS122)

Acknowledgements

We would like to express our gratitude to our parents for their undying support.This project would not have been possible without the invaluable guidance and support of our guide, Dr. Mydhili K Nair, Assistant Professor, Department of Information Science and Engineering M.S. Ramaiah Institute Of Technology. We thank the Head of our department, Dr. Vijaykumar B.P. for providing encouragement and invaluable support. We are grateful to the lab staff for being extremely patient with us and allowing us to utilize the lab resources. We are indebted to the support provided to us by Dr. S. Y. Kulkarni, Principal, MSRIT. Lastly, we thank the Visvesvaraya Technological Univeristy for providing us with the opportunity to take part in this project, to learn and grow rich with experience.

Abstract

In the present Digital Space a lot of Internet users try to come up with solutions to a particular problem by suggesting solutions that are pre-existing on the Internet. This brings down the originality of posts and we, in our project, try to overcome this problem by applying prediction models on data sets. It is important for a user to come up with original ideas to gain upvotes which in turn represent the quality of a post. Due to the huge influx of data at every moment, the need for big data analytics becomes essential and hence the use of an open source framework like Hadoop is imperative.

The present systems make use of a much more constrained environment that implement MySQL framework. With our setup, we take advantage of distributed computing, effectively removing constraints set up by MySQL and its other counterparts. To achieve the desired results, we implement frameworks like Hive and Mahout on top of Hadoop. Here, Hive acts as a query processing and managing software, Mahout provides us the important prediction models and Hadoop forms the base for these two frameworks.

In our project, we are using different algorithms from the Mahout framework like Co-occurrence and Log-Likelihood to analyze data sets. These models provide us different measures to compare user similarities and dissimilarities. In one of the instances we have applied Co-occurrence model to suggest questions to a user based on his/her answers while comparing him/her with other users.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Prologue

In our project, titled DataStory: Q & A Usergroup Analyser, we have successfully integrated a collection of frameworks and systems to create an optimal prediction model. The core of our project is Hadoop which acts as a base for other frameworks to act upon. On the whole, the system layout consists of Hadoop, Hive and Mahout to process the user input. In our setup, we are using the data dumps provided by the Stack OverFlow community to suggest questions to a user based on the Co-occurance similarity and log-likelihood module available in the Mahout Framework. In the following paragraphs, I shall explain the overall working of our setup.

First, we download the data dumps provided by the StackOverFlow community and use our custom built python code to extract data from the xml format into a more appropriate csv format. The python code that we have created requires the input file name, output file name and attributes to convert the given xml format data to csv format. Next, we implement the Hive framework to use its 'BULK LOADING' feature to transfer huge amounts of data present in the csv files into a tabular format for a higher query processing efficiency. Next we implement the user defined hive queries to process the data. Hive queries are put through a certain plan prepared by the hive framework which converts the jobs into map-reduce jobs. Map-reduce increases the efficiency of query process and gives results faster. The output from the Hive module is passed onto the Mahout framework for further processing.

Through Mahout, we use Item Recommender module to predict the questions that a user might be interested in answering but hasn't answered yet. This is based on a model which analyses the answers given by other users and maps them into certain relations. For example, consider two users A and B. A has answered a set of 10 questions, say 1 to 10 and B has answered 7 questions from the same set. Our project has been successful in suggesting the 3 questions to B that he had left out. This prediction model can be used on a larger scale to predict a user's interest in a particular field and suggest him questions

based on it. The other model that we have inherited from the Mahout framework is the User Recommender module which tells us how similar two users are, in a range of -1 to 1. Two users, considered as a series of numbers, are compared with each other based on certain attributes. By using Pearson correlation, we can measure the tendency of the numbers to move together proportionally, such that there's roughly linear relationship between the values in one series and the other. When this tendency is high, the correlation value is closer to 1 and it is near 0 when there appears to be very little relation among the two users. It is closer to -1 when the users appears to have negative relations.

1.2 Intent

The intent of the project work is to make data analyses that can be evaluated on the basis of several criterions which are mentioned underneath.

- Efficiency (how long does it take to compute the answer, how much memory does it need?)
- Consistency (will it converge on the same answer repeatedly, if each time given different data for the same model problem?)
- Robustness (does it cope well with violations of the assumptions of the underlying model?)
- Falsifiability (does it alert us when it is not good to use, i.e. when assumptions are violated?

1.3 Goal

The goal of our project is to recommend useful and appropriate content to a user based on data analytics using Hadoop Data clusters.The proposed methodology should give a performance enhancement when prediction modules are run in the given setup. The advocated system effectively improves the scalability, reliability and efficiency of the recommendation process.

1.4 Problem Statement

Stack Exchange is an online forum with a number of users and even larger number of questions and answers. These users are responsible for a huge amount of data that is generated everyday on Internet forums. This calls for an effective approach to provide relevant recommendation to users. These recommendations may be in terms of suggesting

questions to users most appropriate to answer based on their previous preference, most relevant answers to questions searched by users, eliminating duplicity in terms of questions and answers or even duplicate accounts in the website.
In our project, we provide such a recommender system that generates recommendation of relevant questions to users and recommendation of users themselves to form an online community to improve post qualities.

Chapter 2

Literature Review

2.1 MiQs: Characterization and Prediction of Migrated Questions on Stack Exchange

It is seen frequently that a posted question and the website domain it is present in, are a mismatch. These are considered off-topic questions and they need to be transferred to a more appropriate site on the SE network. Currently, this problem is tackled by moderators and expert users through a voting process. This procedure is called Question Migration and due to the high amount of overhead and manual labour involved, this technique is undesirable.It also causes delay to the user seeking the answer.
We have an in-depth characterization study on numerous migrated questions retrieved from the datasets provided by SE. The results of temporal distribution, community structure of Q & A websites, owner reputation, amount of discussion, and popularity of migrated questions are also present. It also provides several distinguishing features of migrated questions and proposes a machine learning based framework to predict migrating questions. Results prove its higher efficiency on comparison with the manual technique. [4]

2.2 An Automatic Approach for Identifying Topical Near-Duplicate Relations between Questions from Social Media Q/A Sites

With such huge amount of posts present on the network, the chances of a user asking a question that has already been posted/answered is high. Here the focus is on identifying such topical near-duplicate questions on social media platforms. Often times, questions are explicitly marked as possible duplicates of other entries. This manual markup provides are higher quality dataset for the sparse information provided by the SE data sump.

This obstacle is addressed as a classification task: given a pair of documents, a supervised classifier decides if two questions are topical nearduplicates or not. The result of this experiment shows high performance.[5]

2.3 Predict Closed Questions on StackOverflow

With millions of new questions being posted everyday, it is vital for the moderators to keep a check on the quality and quantity of questions. If a question is an exact duplicate or if it is off topic question that belongs to another site or if the question is vague/too broad/primary opinion based it has to be "closed". According to SE, a closed question is one to which no additional answers can be posted, answers maybe edited but no added. It can be reopened though, with the right privileges. Due to the enormous number of questions being posted, it is important to remove as many unnecessary questions as possible by closing them.
Here the focus is on developing a classifier that predicts whether or not a question will be closed. The input expected by this classifier is the question as it was submitted and the reason it was closed.[1]

2.4 Recommendations in StackOverflow

In social media discussion forums, user generated contents are organized with the help of "tags". This tagging process should be as accurate as possible to maintain stability and precision in the forum. This procedure can be facilitated by recommending tags to users based on the content they generate. He we employ an algorithm NetTagCombine to improve upon existing machine learning methods for tag recommendation.[8]

2.5 Eliciting Answers on StackOverflow

It is important for a user posting a question to know which features of a question make it most likely to be answered. Features were initially limited to the ones within the question itself and those which were known at the time the question was asked. To improve classification results feature set is expanded to include the definition of what it means to be an answered question. Random Forests ,Logistic Regression, K-Nearest Neighbors, and Support Vector Machines were the different algorithms used to make classifications.[2]

2.6 Gender, Representation and Online Participation: A Quantitative Study of StackOverflow

It is interesting to find out patterns relating location, gender and time of user activity. It helps companies look into a particular community more closely; this helps them in their recruitment process. Oftentimes it is said that women are under-represented on such forums. A quantitative study is performed to assess the representation of gender in SE. These findings can be helpful in helping women take up such opportunities more frequently. The result of this paper was unsurprising, male dominance in this forum was clearly seen. Another interesting pattern observed was that men engage in "game" more than women do and tend to garner more reputation.[9]

2.7 Geo-locating the knowledge transfer in StackOverflow

The data obtained from StackExchange includes a location stamp and a timestamp. By analysing this data we get an idea of the way in which different locations of the world contribute to the forums represented by the website. It is implicated that Europe and north America are the top contributors while Asia, mostly represented by Indians, stands third. This information can be very helpful for organizations that depend on outsourcing.[7]

2.8 Hive

The size of data sets being collected and analyzed in the industry for business intelligence is growing rapidly, making traditional warehousing solutions prohibitively expensive. Hadoop is a popular open-source map-reduce implementation which is being used as an alternative to store and process extremely large data sets on commodity hardware. However, the map-reduce programming model is very low level and requires developers to write custom programs which are hard to maintain and reuse. The Hive framework effectively improves the efficiency of the Map-Reduce paradigm.
The primary reason for moving data between SQL stores as well as Hadoop is usually to take advantage of the massive storage and processing capabilities to process quantities of data larger than you could hope to cope with in SQL alone. Increasing popularity of Hadoop as data platform of choice for many organizations, HIVE becomes must-have supplement to provide greater usability as well as connectivity within the organization by introducing high level language support known as HiveQL.[3]

2.9 Mahout

Mahout offers tools to evaluate the prediction quality of a recommender on a random split of the data. For explicit feedback data, a RecommenderEvaluator can compute the mean average error as well as the root mean squared error. In the case of implicit feedback data, the RecommenderIRStatsEvaluator computes statistics such as precision,recall, normalized discounted cumulative gain and related measures.[6]

Chapter 3

System Design

System design is the process of defining the elements of a system such as the architecture, modules and components, the different interfaces of those components and the data that goes through that system. It is meant to satisfy specification's needs and requirements of a business or organization through the engineering of a coherent and well-running system. Systems design implies a systematic approach to the design of a system. It may take a bottom-up or top-down approach, but either way the process is systematic wherein it takes into account all related variables of the system that needs to be created from the architecture, to the required hardware and software, right down to the data and how it travels and transforms throughout its travel through the system. Systems design then overlaps with systems analysis, systems engineering and systems architecture.
The systems design approach first appeared right before World War II, when engineers were trying to solve complex control and communications problems. They needed to be able to standardize their work into a formal discipline with proper methods, especially for new fields like information theory, operations research and computer science in general.

3.1 Architectures

3.1.1 Hadoop

Hadoop consists of the Hadoop Common package, which provides filesystem and OS level abstractions, a MapReduce engine (either MapReduce/MR1 or YARN/MR2) and the Hadoop Distributed File System (HDFS). The Hadoop Common package contains the necessary Java ARchive (JAR) files and scripts needed to start Hadoop. The package also provides source code, documentation, and a contribution section that includes projects from the Hadoop Community.

A small Hadoop cluster includes a single master and multiple worker nodes. The master node consists of a JobTracker, TaskTracker, NameNode and DataNode. A slave or worker node acts as both a DataNode and TaskTracker, though it is possible to have

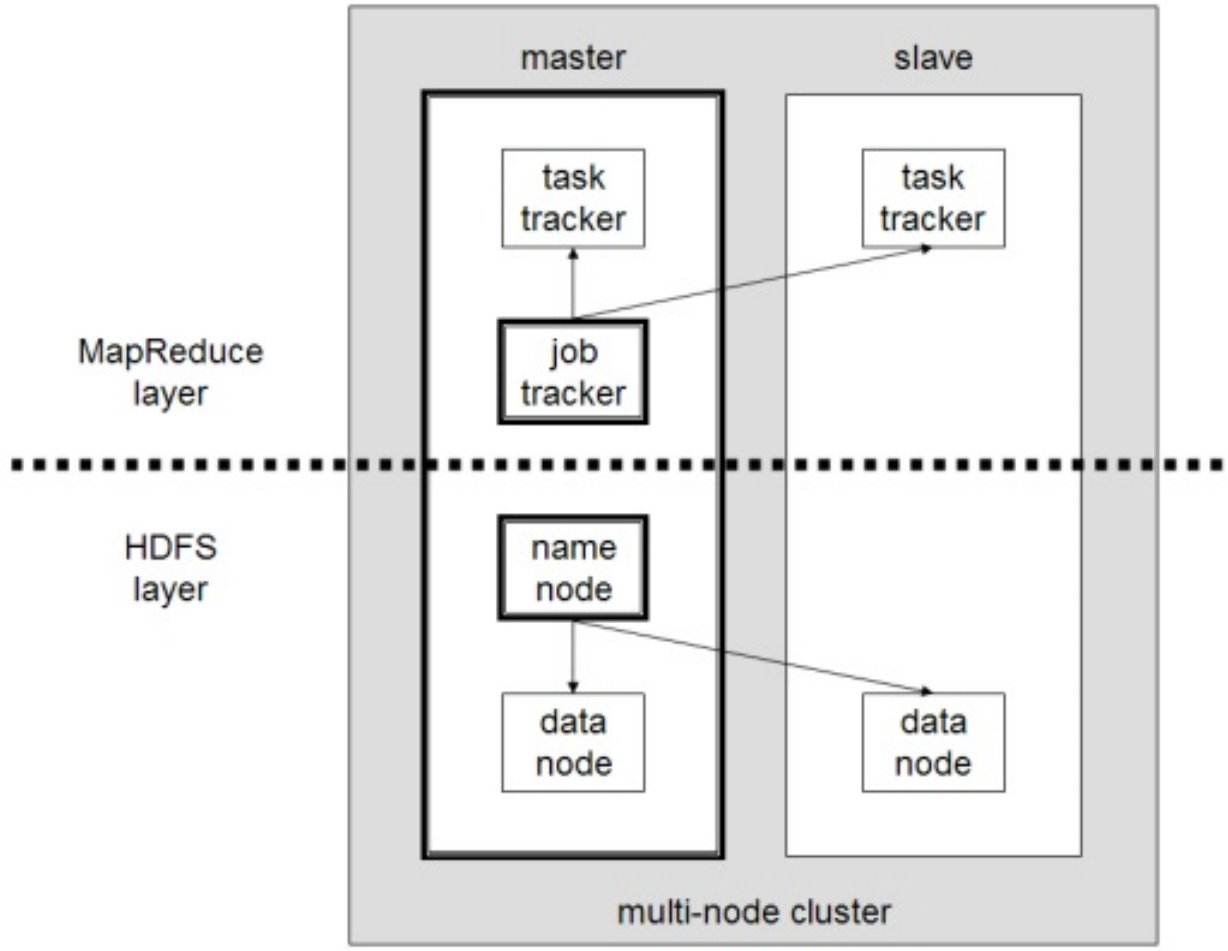

Figure 3.1: A multi-node hadoop cluster

data-only worker nodes and compute-only worker nodes. These are normally used only in nonstandard applications.
In a larger cluster, the HDFS is managed through a dedicated NameNode server to host the file system index, and a secondary NameNode that can generate snapshots of the namenode's memory structures, thus preventing file-system corruption and reducing loss of data. Similarly, a standalone JobTracker server can manage job scheduling. In clusters where the Hadoop MapReduce engine is deployed against an alternate file system, the NameNode, secondary NameNode, and DataNode architecture of HDFS are replaced by the file-system-specific equivalents.

3.1.2 Hadoop Mapreduce

Hadoop MapReduce is a software framework for easier writing of applications which process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner.
A MapReduce job usually splits the input data-set into independent chunks which are

processed by the map tasks in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the reduce tasks. Typically both the input and the output of the job are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks.
Typically the compute nodes and the storage nodes are the same, that is, the MapReduce framework and the Hadoop Distributed File System are running on the same set of nodes. This configuration allows the framework to effectively schedule tasks on the nodes where data is already present, resulting in very high aggregate bandwidth across the cluster.
The MapReduce framework consists of a single master JobTracker and one slave TaskTracker per cluster-node. The master is responsible for scheduling the jobs' component tasks on the slaves, monitoring them and re-executing the failed tasks. The slaves execute the tasks as directed by the master.
Minimally, applications specify the input/output locations and supply map and reduce functions via implementations of appropriate interfaces and/or abstract-classes. These, and other job parameters, comprise the job configuration. The Hadoop job client then submits the job (jar/executable etc.) and configuration to the JobTrackerwhich then assumes the responsibility of distributing the software/configuration to the slaves, scheduling tasks and monitoring them, providing status and diagnostic information to the job-client.

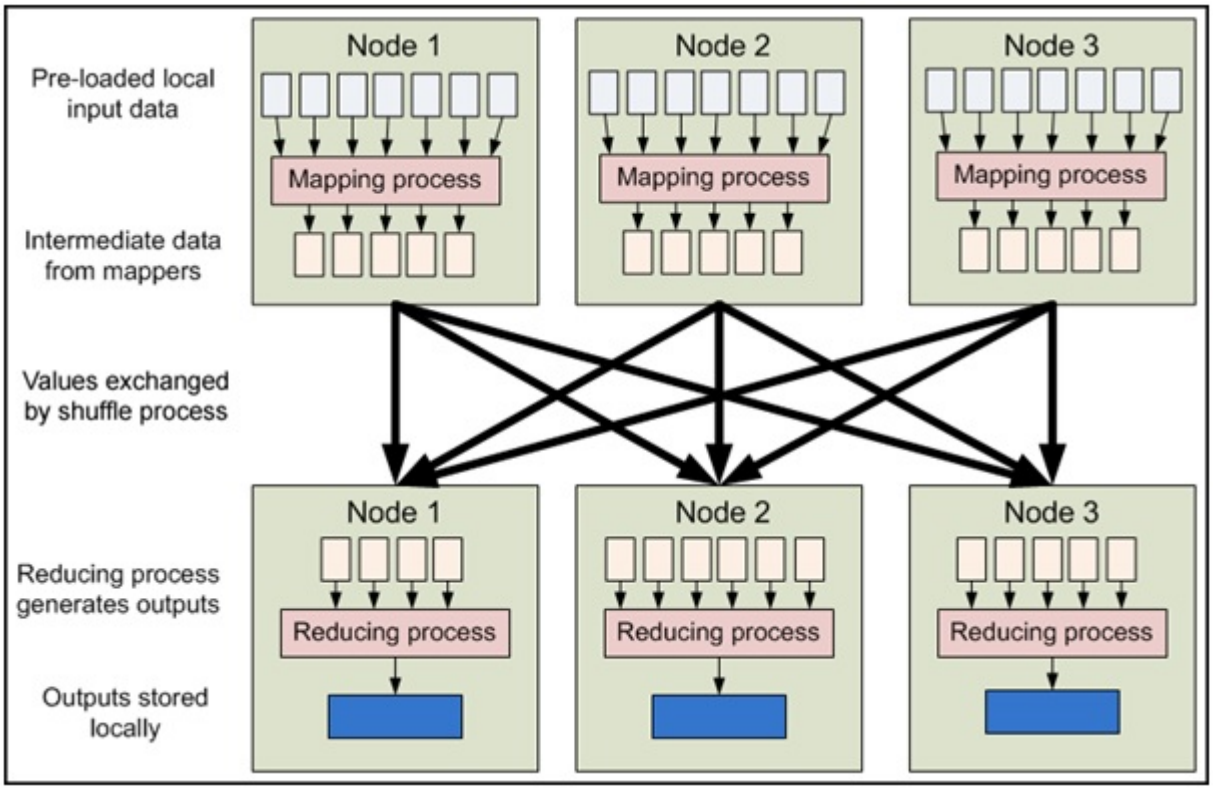

Figure 3.2: A map-reduce job

3.1.3 HDFS Architecture

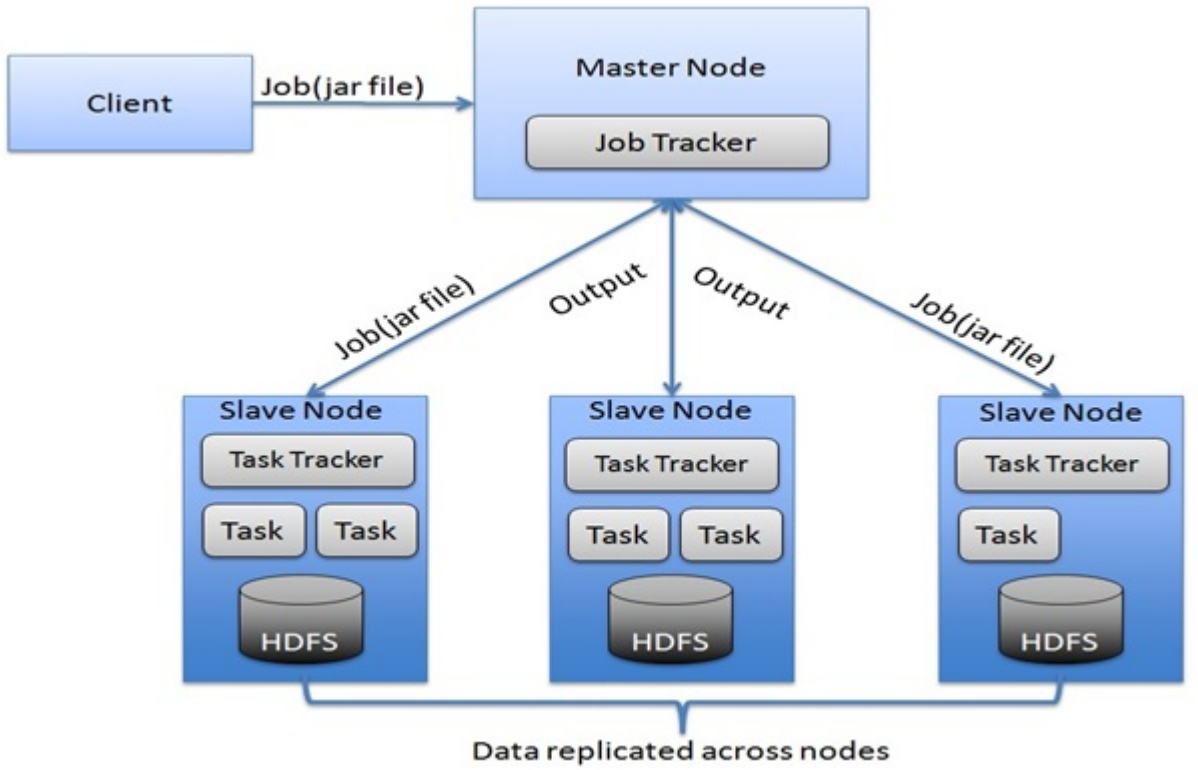

Figure 3.3: Given above is the architecture of a Hadoop File System

HDFS follows the master-slave architecture and it has the following elements.

Name Node

The namenode is the commodity hardware that contains the GNU/Linux operating system and the namenode software. It is a software that can be run on commodity hardware. The system having the namenode acts as the master server and it does the following tasks:

- Manages the file system namespace.
- Regulates client's access to files.
- It also executes file system operations such as renaming, closing, and opening files and directories.

Data Node

The datanode is a commodity hardware having the GNU/Linux operating system and datanode software. For every node (Commodity hardware/System) in a cluster, there will be a datanode. These nodes manage the data storage of their system.

- Datanodes perform read-write operations on the file systems, as per client request.
- They also perform operations such as block creation, deletion, and replication according to the instructions of the namenode.

Job Tracker
The JobTracker is the service within Hadoop that farms out MapReduce tasks to specific nodes in the cluster, ideally the nodes that have the data, or at least are in the same rack.

- Client applications submit jobs to the Job tracker.
- The JobTracker talks to the NameNode to determine the location of the data
- The JobTracker locates TaskTracker nodes with available slots at or near the data
- The JobTracker submits the work to the chosen TaskTracker nodes.
- The TaskTracker nodes are monitored. If they do not submit heartbeat signals often enough, they are deemed to have failed and the work is scheduled on a different TaskTracker.
- A TaskTracker will notify the JobTracker when a task fails. The JobTracker decides what to do then: it may resubmit the job elsewhere, it may mark that specific record as something to avoid, and it may may even blacklist the TaskTracker as unreliable.
- When the work is completed, the JobTracker updates its status.
- Client applications can poll the JobTracker for information.

Task Tracker
A TaskTracker is a node in the cluster that accepts tasks - Map, Reduce and Shuffle operations - from a JobTracker.
Every TaskTracker is configured with a set of slots, these indicate the number of tasks that it can accept. When the JobTracker tries to find somewhere to schedule a task within theMapReduce operations, it first looks for an empty slot on the same server that hosts the DataNode containing the data, and if not, it looks for an empty slot on a machine in the same rack.
The TaskTracker spawns a separate JVM processes to do the actual work; this is to ensure that process failure does not take down the task tracker. The TaskTracker monitors these spawned processes, capturing the output and exit codes. When the process finishes, successfully or not, the tracker notifies the JobTracker. The TaskTrackers also send out heartbeat messages to the JobTracker, usually every few minutes, to reassure the JobTracker that it is still alive. These message also inform the JobTracker of the number of available slots, so theJobTracker can stay up to date with where in the cluster work can be delegated.
Block
Generally the user data is stored in the files of HDFS. The file in a file system will be divided into one or more segments and/or stored in individual data nodes. These file

segments are called as blocks. In other words, the minimum amount of data that HDFS can read or write is called a Block. The default block size is 64MB, but it can be increased as per the need to change in HDFS configuration.

3.1.4 HIVE

The Apache Hive data warehouse software facilitates querying and managing large datasets residing in distributed storage. Hive provides a mechanism to project structure onto this data and query the data using a SQL-like language called HiveQL. At the same time this language also allows traditional map/reduce programmers to plug in their custom mappers and reducers when it is inconvenient or inefficient to express this logic in HiveQL. The main components of Hive are:

- UI – The user interface for users to submit queries and other operations to the system. As of 2011 the system had a command line interface and a web based GUI was being developed.
- Driver – The component which receives the queries. This component implements the notion of session handles and provides execute and fetch APIs modeled on JDBC/ODBC interfaces.
- Compiler – The component that parses the query, does semantic analysis on the different query blocks and query expressions and eventually generates an execution plan with the help of the table and partition metadata looked up from the metastore.
- Metastore – The component that stores all the structure information of the various tables and partitions in the warehouse including column and column type information, the serializers and deserializers necessary to read and write data and the corresponding HDFS files where the data is stored.
- Execution Engine – The component which executes the execution plan created by the compiler. The plan is a DAG of stages. The execution engine manages the dependencies between these different stages of the plan and executes these stages on the appropriate system components.

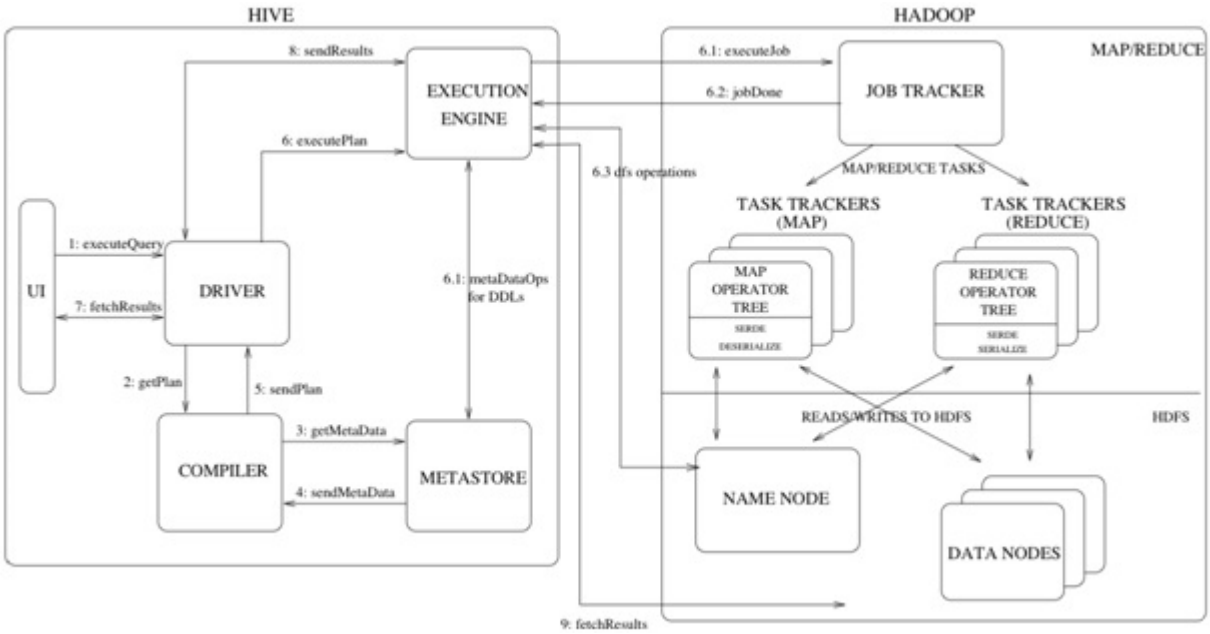

Figure 3.4: Architecture of hive

3.1.5 Mahout

Apache Mahout is a project of the Apache Software Foundation to produce free implementations of distributed or otherwise scalable machine learning algorithms focused primarily in the areas of collaborative filtering, clustering and classification. Many of the implementations use the Apache Hadoop platform. Mahout also provides Java libraries for common maths operations (focused on linear algebra and statistics) and primitive Java collections. Mahout is a work in progress; the number of implemented algorithms has grown quickly.
A Mahout-based collaborative filtering engine takes users' preferences for items ("tastes") and returns estimated preferences for other items.
Mahout provides a rich set of components from which you can construct a customized recommender system from a selection of algorithms. Mahout is designed to be enterprise-ready; it's designed for performance, scalability and flexibility.
Top-level packages define the Mahout interfaces to these key abstractions:

- DataModel
- UserSimilarity
- ItemSimilarity
- UserNeighborhood
- Recommender

Subpackages of org.apache.mahout.cf.taste.impl hold implementations of these interfaces. These are the pieces from which you will build your own recommendation engine.

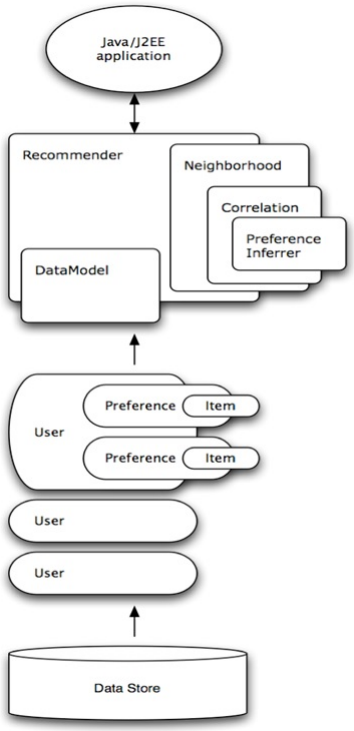

Figure 3.5: Architecture of Mahout

3.2 Data Format

3.2.1 Input Format

MapReduce

The MapReduce framework operates exclusively on <key, value> pairs, that is, the framework views the input to the job as a set of <key, value> pairs and produces a set of <key, value> pairs as the output of the job, conceivably of different types.

The key and value classes have to be serializable by the framework and hence need to implement the Writable interface. Additionally, the key classes have to implement the WritableComparable interface to facilitate sorting by the framework.

Input and Output types of a MapReduce job:

(input) <k1, v1> -> map -> <k2, v2> -> combine -> <k2, v2> -> reduce -> <k3, v3> (output)

HIVE

In this project we are using CSV file as Input to HIVE , A comma-separated values (CSV) (also sometimes called character-separated values) file stores tabular data (numbers and

text) in plain-text form. Plain text means that the file is a sequence of characters, with no data that has to be interpreted as binary numbers. It consists of number of records, separated by line breaks of some kind; each record consists of fields, separated by some other character or string, most commonly a literal comma or tab records have an identical sequence of fields.we are extracting that data from the archive.org where stack overflow website will host there dumps in 7z format, 7z is a compressed archive file format that supports several different data compression, encryption and pre-processing algorithms. The 7z format initially appeared as implemented by the 7-Zip archiver.

Mahout

Mahout will get its input from hive by running selected queries by pipelining the output from hive to mahout to run the job ,Mahout will get the input in CSV Format with each record consists of fields, separated by some other character or string, most commonly a literal comma or tab records have an identical sequence of fields.

3.2.2 Output Format

In this project Output will be parsed from two different modules

Hive:

In hive by executing the hive queries we can get the output in the desired format we are taking outputin CSV format where each record consists of fields, separated by some other character or string, most commonly a literal comma or tab records have an identical sequence of fields.

Mahout:

Mahout will output the resulting output in CSV Format ,while that will be parsed with help of Java Modules and it will be pipelined to the Java Servlet on the Backend for some processing and the output will be displayed in the HTML file as a final result.

3.3 Modules

3.3.1 Hadoop Distributed File System (HDFS)

Hadoop File System was developed using distributed file system design. It is run on commodity hardware. Unlike other distributed systems, HDFS is highly faulttolerant and designed using low-cost hardware.

HDFS holds very large amount of data and provides easier access. To store such huge data, the files are stored across multiple machines. These files are stored in redundant fashion to rescue the system from possible data losses in case of failure. HDFS also makes applications available to parallel processing. **Features of HDFS**

- It is suitable for the distributed storage and processing.

- Hadoop provides a command interface to interact with HDFS.
- The built-in servers of namenode and datanode help users to easily check the status of cluster.
- Streaming access to file system data.
- HDFS provides file permissions and authentication.

3.3.2 Hadoop MapReduce

It is a software framework for easily writing applications which process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner.
A MapReduce job usually splits the input data-set into independent chunks which are processed by the map tasks in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the reduce tasks. Typically both the input and the output of the job are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks.Typically the compute nodes and the storage nodes are the same, that is, the MapReduce framework and the Hadoop Distributed File System are running on the same set of nodes. This configuration allows the framework to effectively schedule tasks on the nodes where data is already present, resulting in very high aggregate bandwidth across the cluster.
The MapReduce framework consists of a single master JobTracker and one slave TaskTracker per cluster-node. The master is responsible for scheduling the jobs' component tasks on the slaves, monitoring them and re-executing the failed tasks. The slaves execute the tasks as directed by the master.
Minimally, applications specify the input/output locations and supply map and reduce functions via implementations of appropriate interfaces and/or abstract-classes. These, and other job parameters, comprise the job configuration. The Hadoop job client then submits the job (jar/executable etc.) and configuration to the JobTrackerwhich then assumes the responsibility of distributing the software/configuration to the slaves, scheduling tasks and monitoring them, providing status and diagnostic information to the job-client.
Although the Hadoop framework is implemented in JavaTM, MapReduce applications need not be written in Java.c
Payload
Applications typically implement the Mapper and Reducer interfaces to provide the map and reduce methods. These form the core of the job.
Mapper
Mapper maps input key/value pairs to a set of intermediate key/value pairs.
Maps are the individual tasks that transform input records into intermediate records.

The transformed intermediate records do not need to be of the same type as the input records. A given input pair may map to zero or many output pairs.
The Hadoop MapReduce framework spawns one map task for each InputSplit generated by the InputFormat for the job.Overall, Mapper implementations are passed the JobConf for the job via the JobConfigurable.configure(JobConf) method and override it to initialize themselves. The framework then calls map(WritableComparable, Writable, OutputCollector, Reporter) for each key/value pair in the InputSplit for that task. Applications can then override the Closeable.close() method to perform any required cleanup.Output pairs do not need to be of the same types as input pairs. A given input pair may map to zero or many output pairs. Output pairs are collected with calls toOutputCollector.collect(WritableComparable,Writable).
Applications can use the Reporter to report progress, set application-level status messages and update Counters, or just indicate that they are alive.All intermediate values associated with a given output key are subsequently grouped by the framework, and passed to the Reducer(s) to determine the final output. Users can control the grouping by specifying a Comparator via JobConf.setOutputKeyComparatorClass(Class).
The Mapper outputs are sorted and then partitioned per Reducer. The total number of partitions is the same as the number of reduce tasks for the job. Users can control which keys (and hence records) go to which Reducer by implementing a custom Partitioner.
Users can optionally specify a combiner, via JobConf.setCombinerClass(Class),to perform local aggregation of the intermediate outputs, which helps to cut down the amount of data transferred from the Mapper to the Reducer.The intermediate, sorted outputs are always stored in a simple (key-len, key, value-len, value) format. Applications can control if,and how, the intermediate outputs are to be compressed and the CompressionCodec to be used via the JobConf.

Reducer

Reducer reduces a set of intermediate values which share a key to a smaller set of values.The number of reduces for the job is set by the user via JobConf.setNumReduceTasks(int).
Overall, Reducer implementations are passed the JobConf for the job via the JobConfigurable.configure(JobConf) method and can override it to initialize themselves. The framework then calls reduce(WritableComparable, Iterator, OutputCollector, Reporter) method for each <key, (list of values)> pair in the grouped inputs. Applications can then override the Closeable.close() method to perform any required cleanup.
Reducer has 3 primary phases: shuffle, sort and reduce.

Shuffle

Input to the Reducer is the sorted output of the mappers. In this phase the framework fetches the relevant partition of the output of all the mappers, via HTTP.

Reduce

In this phase the reduce(WritableComparable, Iterator, OutputCollector, Reporter) method is called for each <key, (list of values)> pair in the grouped inputs.

The output of the reduce task is typically written to the FileSystem via OutputCollector.collect(WritableComparable, Writable).
Applications can use the Reporter to report progress, set application-level status messages and update Counters, or just indicate that they are alive.
The output of the Reducer is not sorted.

Partitioner

Partitioner partitions the key space.
Partitioner controls the partitioning of the keys of the intermediate map-outputs. The key (or a subset of the key) is used to derive the partition, typically by a hash function. The total number of partitions is the same as the number of reduce tasks for the job. Hence this controls which of the m reduce tasks the intermediate key (and hence the record) is sent to for reduction.
HashPartitioner is the default Partitioner.

OutputCollector

OutputCollector is a generalization of the facility provided by the MapReduce framework to collect data output by the Mapper or the Reducer (either the intermediate outputs or the output of the job).
Hadoop MapReduce comes bundled with a library of generally useful mappers, reducers, and partitioners.

3.3.3 Python

xml.etree

XML is an inherently hierarchical data format, and the most natural way to represent it is with a tree. ET has two classes for this purpose -ElementTree represents the whole XML document as a tree, and Element represents a single node in this tree. Interactions with the whole document (reading and writing to/from files) are usually done on the ElementTree level. Interactions with a single XML element and its sub-elements are done on the Element level.

argparse

The argparse module makes it easy to write user friendly command line interfaces.The program defines what arguments it requires, and argparse will figure out how to parse those out of sys.argv. The argparse module also automatically generates help and usage messages and issues errors when users give the program invalid arguments.

3.3.4 Sequence Diagram

A Sequence diagram is an interaction diagram that shows how processes operate with one another and what is their order. It is a construct of a Message Sequence Chart. A sequence diagram shows object interactions arranged in time sequence. It depicts the

objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called event diagrams or event scenarios.

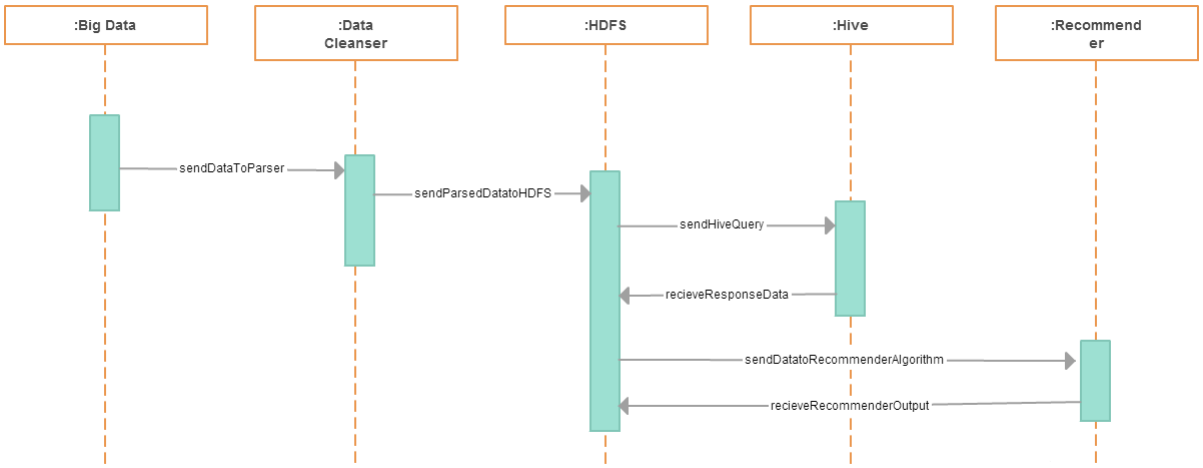

Figure 3.6: Sequence Diagram of QA Analyzer

3.3.5 Deployment Diagram

A deployment diagram in the Unified Modeling Language models the physical deployment of artifacts on nodes. The nodes appear as boxes, and the artifacts allocated to each node appear as rectangles within the boxes.Nodes may have sub-nodes, which appear as nested boxes. A single node in a deployment diagram may conceptually represent multiple physical nodes.Device nodes are physical computing resources with processing memory and services to execute software, such as typical computers or mobile phones.An execution environment node (EEN) is a software computing resource

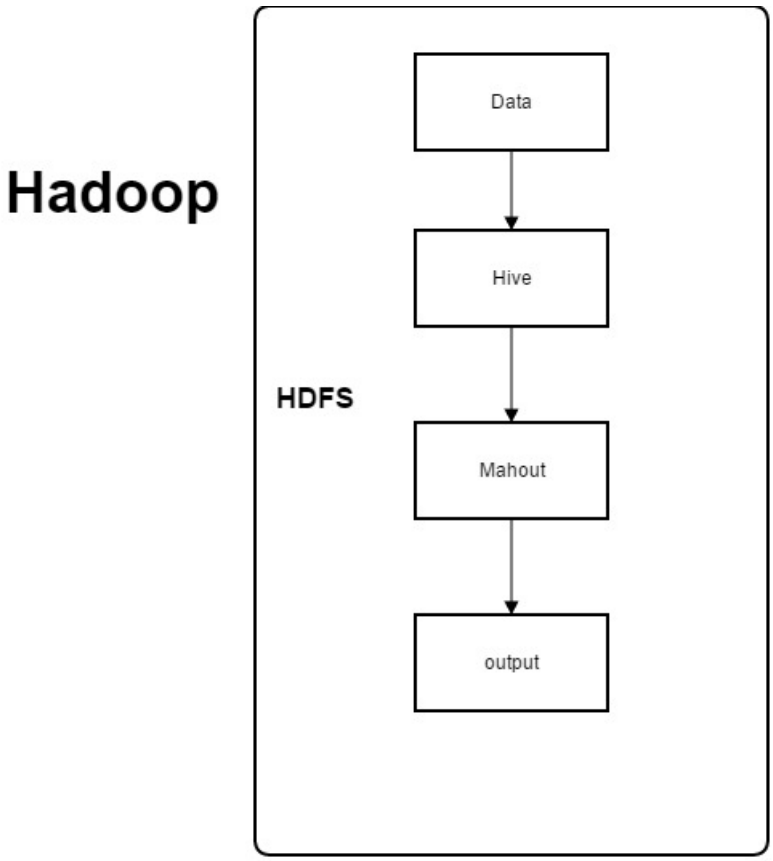

Figure 3.7: Deployment Diagram of QA Analyzer

3.3.6 Use Case Diagram

A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well. An oval circle represents the use case.The Use Case Diagram is as shown in figure

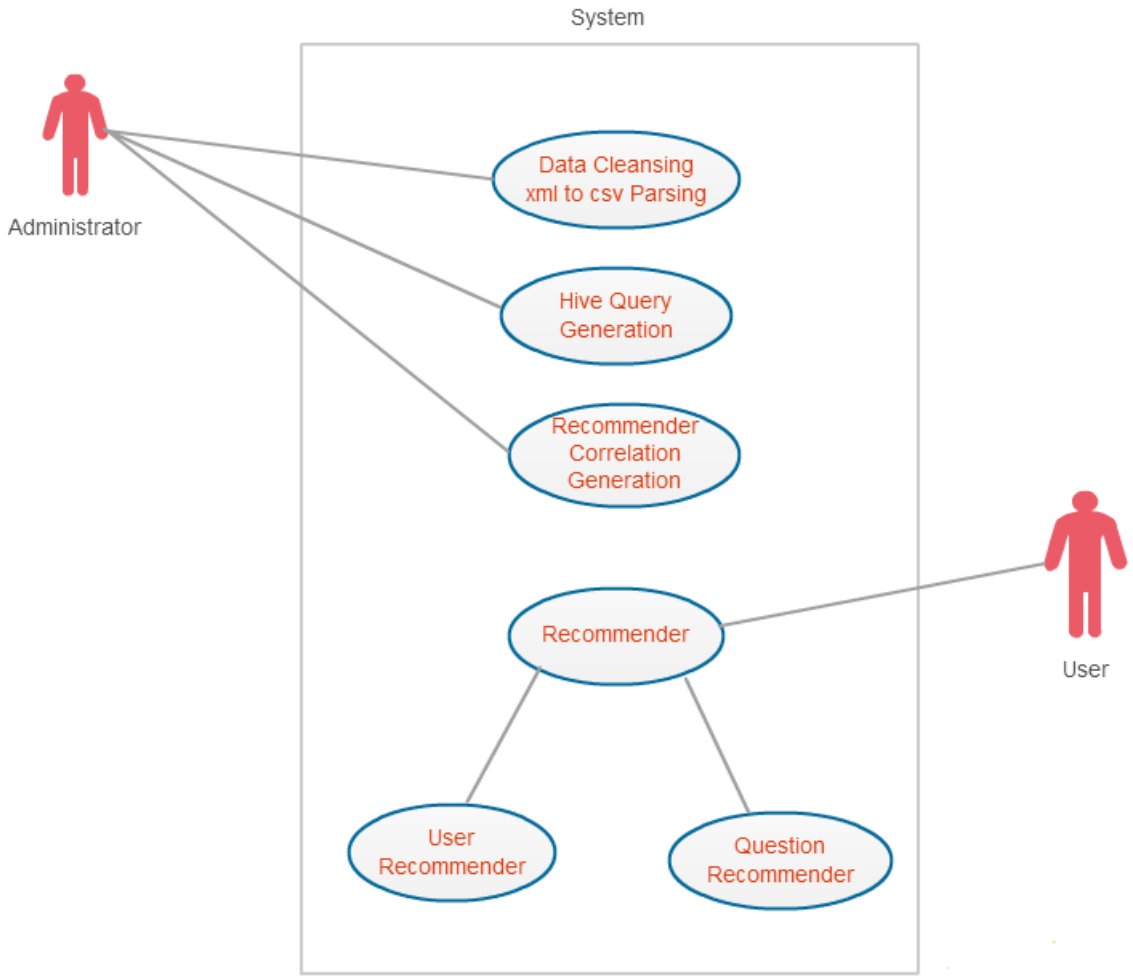

Figure 3.8: Use Case Diagram of QA Analyzer

3.4 Software Requirement Specification

Hadoop requires Java Runtime Environment (JRE) 1.6 or higher. The standard startup and shutdown scripts require that Secure Shell (ssh) be set up between nodes in the cluster.

3.4.1 Overall Description

The project is implemented on Linux platform. The programming languages used for the core of the project are Python and Java, as for the front-end we have used CSS, HTML and JSP. It fulfils the following requirements:

- Data aggregation.
- Data cleansing.
- MapReduce job implementation.

- Prediction models.
- Graphical interface.

Project Perspective

This project is built to achieve two goals, to ascertain the level of similarity between two users and to predict questions that a user might be interested in answering. It takes into consideration the user profiles from online forums like StackExchange and processes the data through three steps in our setup. The input is cleansed to reach the appropriate format, managed and processed by a framework and finally analyzed by prediction models. The final output would be the extent of similarity between two users on a scale of -1 to 1 and questions that a user might be interested in answering but hasn't answered yet.

User interfaces

This project is built to achieve two goals, to ascertain the level of similarity between two users and to predict questions that a user might be interested in answering. It takes into consideration the user profiles from online forums like StackExchange and processes the data through three steps in our setup. The input is cleansed to reach the appropriate format, managed and processed by a framework and finally analyzed by prediction models. The final output would be the extent of similarity between two users on a scale of -1 to 1 and questions that a user might be interested in answering but hasn't answered yet.
Most of the jobs are done using Command Line Interface. HDFS slows a web client to allow users to access its contents through a graphical layout. Hive and Mahout User interfaces are limited to commands through the terminal.
The final output is displayed on a web client, it has an option to enter the user id and access the predictions made by the setup. The web layout allows user to reach the predicted questions and users in real time, provided there is a working internet connection.

Hardware interfaces

- Mouse is required for clicking on the web layout to access the final output.
- Keyboard is required to input new information and to run the commands.
- Multiple Computer systems are required for a multi cluster node.
- A high speed working internet connection.
- 1TB+ of hard disk space on all systems.
- 4GB+ of Ram on all systems, Master node should preferably have more.

Software interfaces

This project runs on three levels.

- Hadoop: The core of the project is based on the Hadoop Framework that allows us to use the advantages of distributed computing. Hadoop Distributed File System and MapReduce framework help in improving scalability, flexibility and efficiency of the setup.

- Hive: Hive is a data warehouse infrastructure built on top of Hadoop for providing data summarization, query, and analysis. In our project, we make use of the bulk loading feature to transfer huge amounts of data onto the HDFS using Hive and we use Hive QL to implement our queries.

- Mahout: Mahout Framework is responsible for delivering us the prediction models. The Item based recommender and User based Recommender systems are derived from this framework.

The final output is displayed on a web client and it makes use of HTML, CSS and JSP frameworks to achieve it.

Communication interfaces

Most of the communication aspect of the system is done manually as of now. But, the data storage and processing is taken care of by the HDFS module. Name node and data node are capable of handling huge volumes of data.

3.4.2 Product Functions

Our setup possesses the following functions:

- Data formatting: Our python script helps user to convert raw data into appropriate format for further processing. The output from this step can be transferred onto the HDFS with much more ease.

- Data management: Hive Framework is responsible for data transfer, data query and data analysis. Its powerful features improve the efficiency of the system overall.

- Recommender system: Mahout Framework brings in the User and Item recommendation models which help the project in predicting user trends and suggesting questions and users.

3.4.3 User characteristics:

All users should have the following characteristics:

- Sound knowledge of how to read and interpret the information from Microsoft Excel .csv files.

- Should be able to read and understand English language.

- Should be able to perform the required functions from the GUI(Graphical user interface).
- Should have knowledge on working of the Hadoop Framework.

Potential users of our project:

- Educational Institutions: The user recommender part of our project can help institutions classify student based on various parameters.
- Recruiters: Companies looking to recruit candidates adept at certain fields can use the question recommender part of our project to list out the people capable of answering questions related to those fields.

3.4.4 Constraints:

General constraints regarding design/implementation are:

- Hadoop, Hive and Mahout version compatibility issues.
- CSV files should be compatible.
- Output from the prediction models will have too many values and is hard to understand.
- Only a few of the predictions are displayed on the Web Layout.

3.4.5 Assumptions and Dependencies:

Data dumps are provided by StackExchange every three months. So analysis should be conducted once every three months and not on a daily or monthly basis.

3.5 Specific Requirements

3.5.1 Performance requirements:

The performance of the application depends on the hardware. But, the beauty of our project lies here. Our setup is capable of running on commodity hardware and has no specific set of hardware requirements. But it is advisable to have the following hardware components in to have appreciable performance levels:

- Intel i5 processor 2.5 Ghz.
- 4GB of RAM.

- 1TB hard disk space.
- 10mbps internet connection.

3.6 Software Requirements

3.6.1 Reliability:

- Regular coding practises shall be utilised.
- Simple GUI that can be used with minimal system understanding.
- Efficient handling of memory resources.

3.6.2 Maintainability:

- Regular coding practices shall be documented and utilized.
- Codes/commands are organized and easy to read.
- Codes/commands are well documented.
- Modules must be loosely coupled.

3.6.3 Design considerations:

There are many design considerations that we have to be aware of before designing a complete solution for the system. The upcoming sections consist of dependencies/assumptions of the software, the architectural strategy and the development method. It also includes an overall system design which gives an insight on a high level overview of the software functionality.

3.6.4 Assumptions and Dependencies:

Data dumps are provided by StackExchange every three months. So analysis should be conducted once every three months and not on a daily or monthly basis.

3.6.5 General constraints:

General design/implementation constraints include:

- The application will run on a system which has a Python compiler and Java development environment.
- CSV files should be compatible.
- Report generated will be stored in HDFS.

3.6.6 Product specific constraints:

The constraints specific to the product are specified below:

- Hadoop, Hive and Mahout of right compatibility have to be installed on the system.
- JRE and Python compiler needs to be installed before starting the application.

3.6.7 Goals and Guidelines:

The proposed system will meet the following design rules:

- Ease of use: Users with basic computer knowledge will be able to use the front-end due to its simplicity.
- Data load: It can handle huge amounts of data while delivering fast processing time.
- Portability: Data can be transferred easily between computers/systems which are situated away from each other.
- Extensibility: The application is designed to allow addition of new functionality easily such as new parameters. The addition of new modules should not cause any unwanted side effects.
- Scalability: With the power of Hadoop, scalability concerns are brushed aside. As the amount of data increases, it is wise to increase the number of nodes in the cluster too.

3.7 Architectural Strategies:

3.7.1 Programming Language

Python is a widely used general-purpose, high-level programming language that supports multiple programming paradigms, including object-oriented, imperative and functional programming or procedural styles. In our project, we use python script to convert the raw semi-structured/unformatted data into a more efficient structured (csv) format data. Java is indeed one of the most powerful object oriented programming languages. The MapReduce part of the Mahout Recommender system is implemented using the Java language.
The front-end of the project is developed using Eclipse integrated development environment. It contains a base workspace and an extensible plug-in system for customizing the environment. It uses plug-ins to provide all the functionality within and on top of the runtime system.

3.7.2 Data Storage Mechanism:

The input data from StackExchange dumps are in XML format which is then converted to csv format using the custom built python script. This formatted data is then taken onto the Hadoop Framework with the help of Hive Bulk Loading feature. This data is stored in the Hive tables and is accessible for querying and analysis through Hive QL.

3.7.3 Communication Mechanism:

Communication aspects of the system are handled by the HDFS. Name nodes and Data nodes are responsible for storing data and processing them in slave nodes. The inbuilt mechanisms are powerful enough to handle huge volumes of data easily.

Chapter 4

Software Implementation

4.1 Hadoop Installation

Step 1:
We download Hadoop from the Apache Download Mirrors and extract the contents of the Hadoop package to a location of your choice. We picked /usr/local/hadoop. We then changed the owner of all the files to the hduser user and hadoop group, for example:
$cd /usr/local
$sudo tar xzf hadoop-1.2.1.tar.gz
$sudo mv hadoop-1.2.1 hadoop
$sudo chown -R hduser:hadoop hadoop
Step 2:
Configure Java HOME in hadoop-env.sh
The only required environment variable we have to configure for Hadoop in this tutorial is JAVAHOME. Open conf/hadoop-env.sh in the editor of your choice (if you used the installation path in this tutorial, the full path is /usr/local/hadoop/conf/hadoop-env.sh) and set the JAVAHOME environment variable to the Sun JDK/JRE 6 directory.
Step 3:
Configure conf/*-site.xml
In this step, we will configure the directory where Hadoop will store its data files, the network ports it listens to, etc. Our setup will use Hadoop's Distributed File System, HDFS, even though our little "cluster" only contains our single local machine.
We can leave the settings below "as is" with the exception of the hadoop.tmp.dir parameter – this parameter you must change to a directory of your choice. We will use the directory /app/hadoop/tmp in this tutorial. Hadoop's default configurations use hadoop.tmp.dir as the base temporary directory both for the local file system and HDFS, so don't be surprised if you see Hadoop creating the specified directory automatically on HDFS at some later point.
Step 4:

Formatting the HDFS filesystem via the NameNode
The first step to starting up your Hadoop installation is formatting the Hadoop filesystem which is implemented on top of the local filesystem of your "cluster" (which includes only your local machine if you followed this tutorial). You need to do this the first time you set up a Hadoop cluster.

Step 5:

Starting your single-node cluster
Run the command:
hduser@ubuntu: $ /usr/local/hadoop/bin/start-all.sh
whether the expected Hadoop processes are running is jps
2287 TaskTracker
2149 JobTracker
1938 DataNode
2085 SecondaryNameNode
2349 Jps
1788 NameNode

Step 6:

Stopping your single-node cluster
Run the command

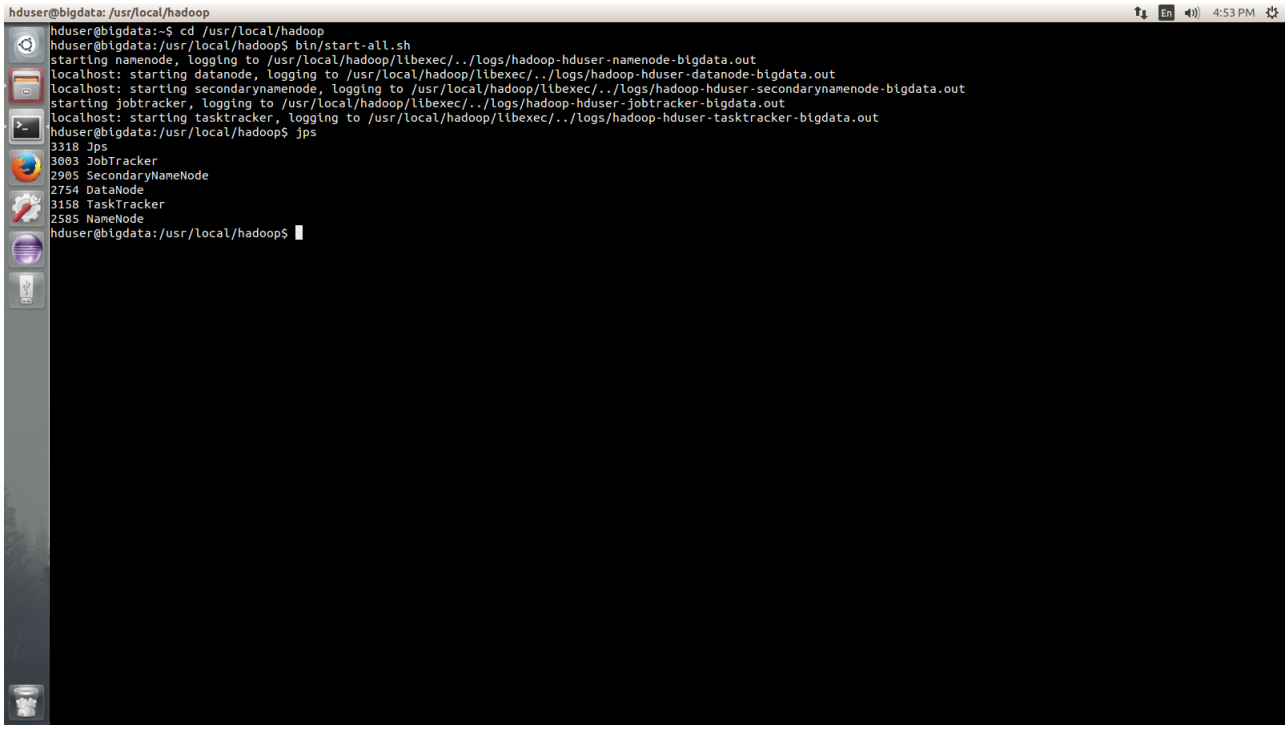

Figure 4.1: Starting Single Node Cluster

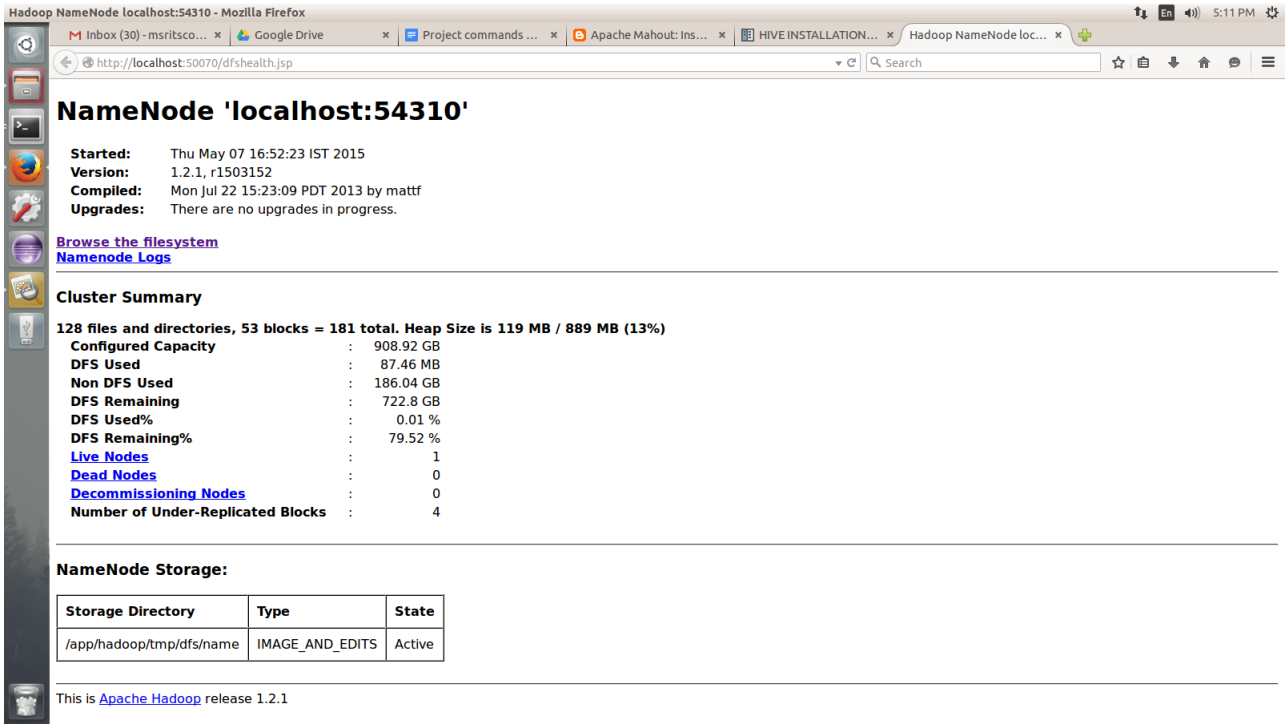

Figure 4.2: Web UI of the NameNode daemon

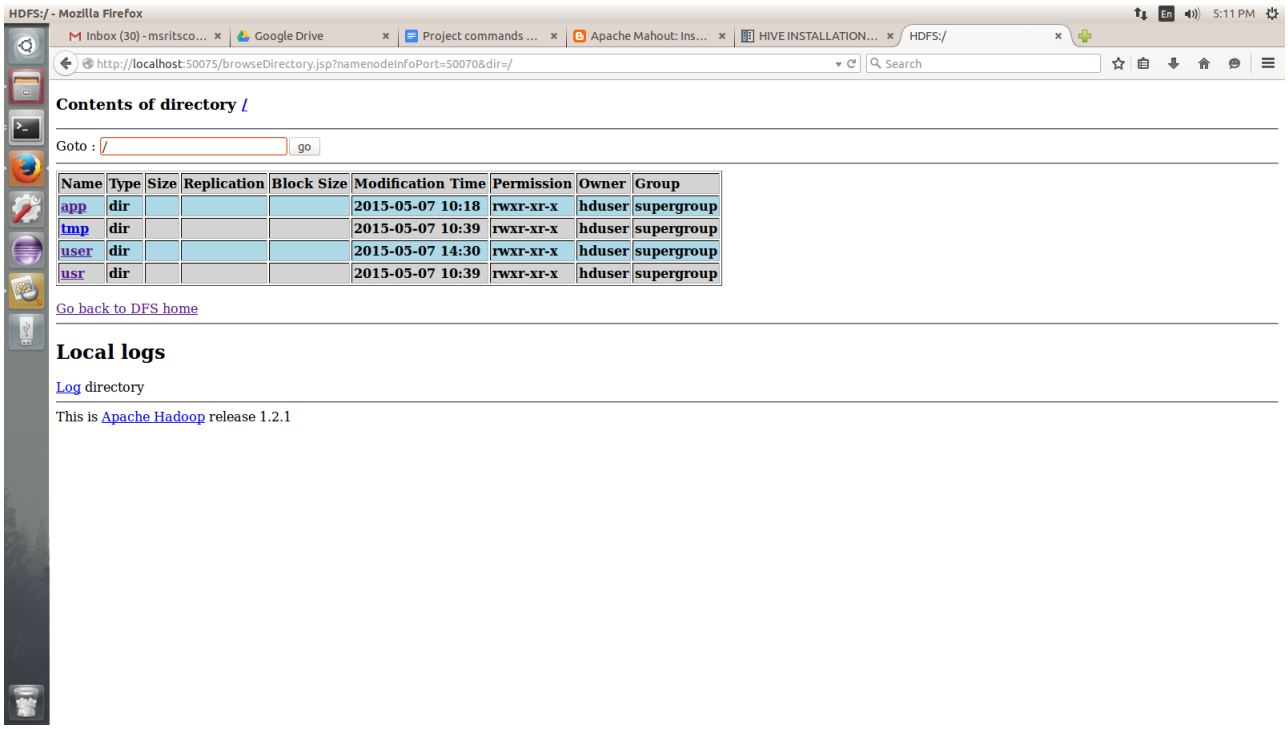

Figure 4.3: HDFS Directories

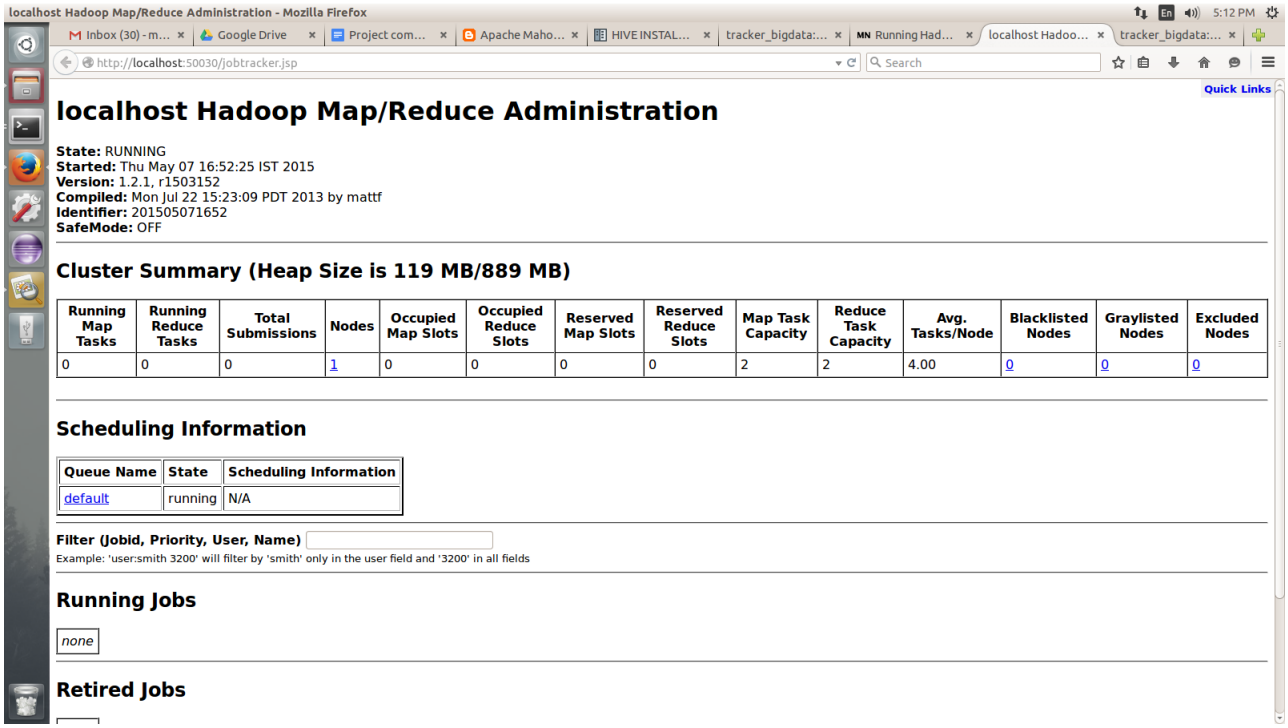

Figure 4.4: Web UI of the JobTracker daemon

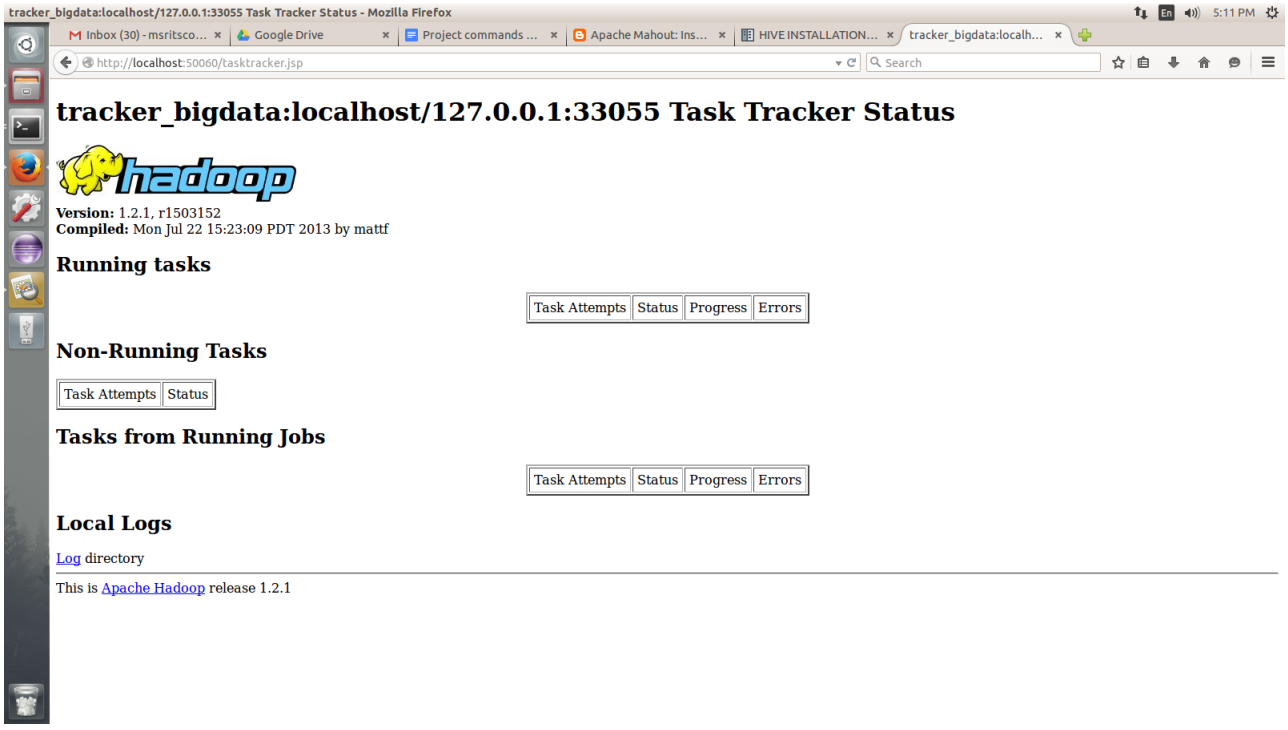

Figure 4.5: Web UI of the TaskTracker daemon

hduser@ubuntu: $ /usr/local/hadoop/bin/stop-all.sh

4.2 Hive Installation

step 1:
We download Hive from the Apache Download Mirrors and extract the contents of the Hadoop package to a location of your choice. We picked /usr/lib/hive.
hduser@hadoop: $ cd /usr/lib/
hduser@hadoop: $ sudo mkdir hive
hduser@hadoop: $ cd Downloads
hduser@hadoop: $ sudo mv apache-hive-0.13.0-bin /usr/lib/hive
step 2:
Setting Hive environment variable:
hduser@hadoop: $ cd
hduser@hadoop: $ sudo gedit /.bashrc
Copy and paste the following lines at end of the file
Set HIVE_ HOME
export HIVE_ HOME="/usr/lib/hive/apache-hive-0.13.0-bin"
PATH=$PATH:$HIVE_ HOME/bin
export PATH
step 3:
Setting HADOOP_ PATH in HIVE config.sh
hduser@hadoop: $ cd /usr/lib/hive/apache-hive-0.13.0-bin/bin
hduser@hadoop: $ sudo gedit hive-config.sh
Go to the line where the following statements are written
Allow alternate conf dir location.
HIVE_ CONF_ DIR="$HIVE_ CONF_DIR:-$HIVE_ HOME/conf"
export HIVE_ CONF_ DIR=$HIVE_CONF_DIR
export HIVE_ AUX_ JARS_ PATH=$HIVE_ AUX_ JARS_ PATH
Below this write the following:
export HADOOP_ HOME=/usr/local/hadoop (write the path where hadoop file is there)
Step 4:
Create Hive directories within HDFS
hduser@hadoop: $ hadoop fs -mkdir /usr/hive/warehouse
Setting READ/WRITE permission for table
hduser@hadoop: $ hadoop fs -chmod g+w /usr/hive/warehouse
Step 5:
HIVE launch
hduser@hadoop: $ hive
Hive shell will prompt:
Logging initialized using configuration in jar:file:/usr/lib/hive/apache-hive-0.13.0-bin/lib/hive-common-0.13.0.jar!/hive-log4j.properties

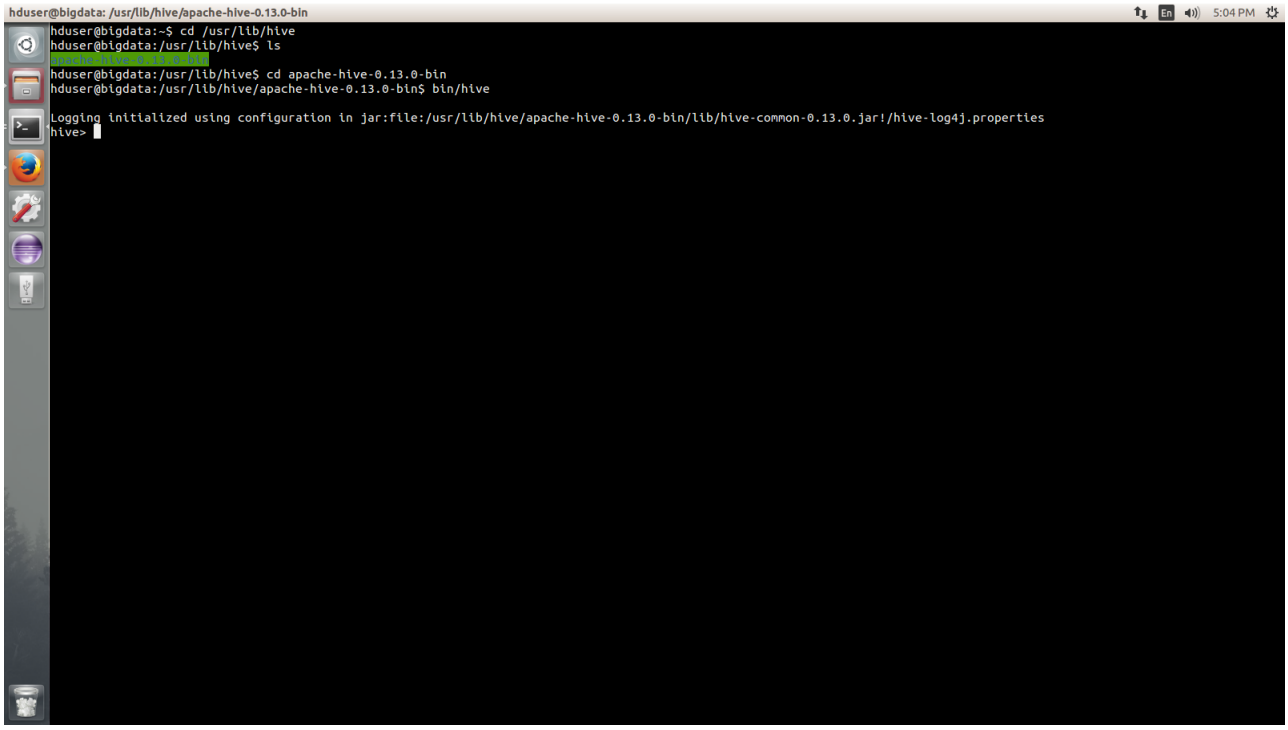

Figure 4.6: Launch Hive

4.3 Maven

One can install maven by running the command
sudo apt-get install maven
but here too we installed Apache maven 3.0.4 manually by following the procedure mentioned below

step 1:

Extract the distribution archive, i.e. apache-maven-3.0.4-bin.tar.gz to the directory you wish to install Maven 3.0.4. These instructions assume you chose /usr/local/apache-maven. The subdirectory apache-maven-3.0.4 will be created from the archive.

step 2:

Open the .bashrc file and add the following lines
- Apache-Maven -
export M2-HOME=/usr/local/apache-maven-3.0.4
export M2=$M2-HOME/bin
export PATH=$M2:$PATH
export JAVA-HOME=$HOME/programs/jdk

step :3

Make sure that JAVA-HOME is set to the location of your JDK, e.g. export JAVA-HOME=/usr/java/jdk1.5.0-02 and that

$JAVA-HOME/bin is in your PATH environment variable.

step4

Run below command to verify that it is correctly installed.

mvn –version

It will show version of installed maven

4.4 Mahout

step 1:

select one of the mahout version which are 0.4 , 0.5 ,0.6 ..we selected 0.9

and make sure that out of so many zipped files in there download the .src zipped file

step 2:

Unzip and name it as mahout and moved to /usr/local

step 3:

Run the commands

hduser@hadoop: $ cd /usr/local/mahout

hduser@hadoop:/usr/local/mahout$ mvn install

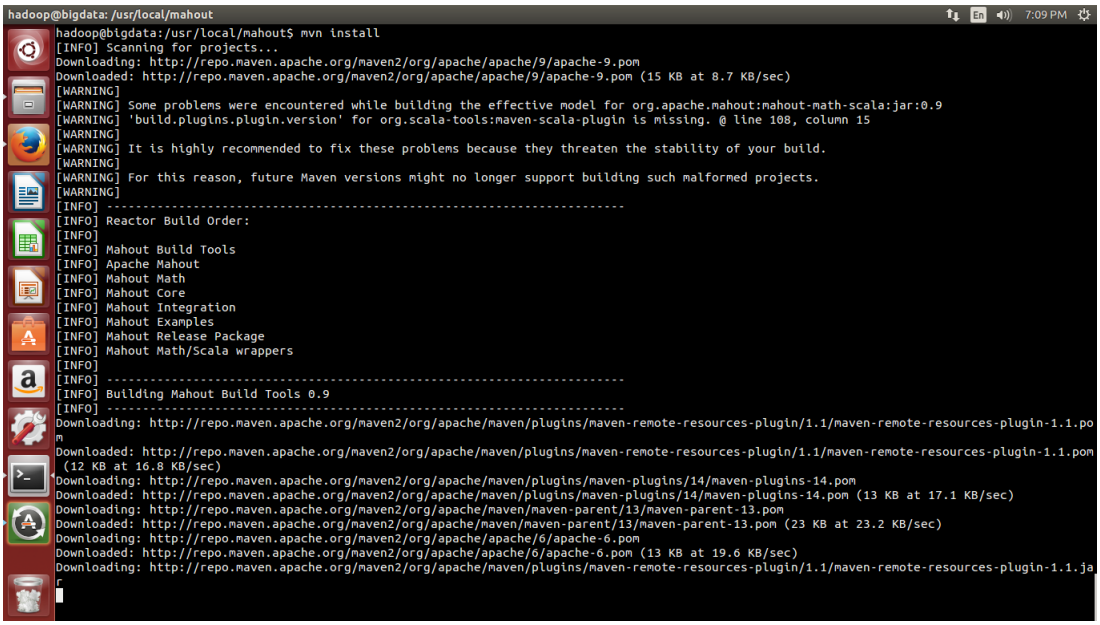

Figure 4.7: Build Mahout

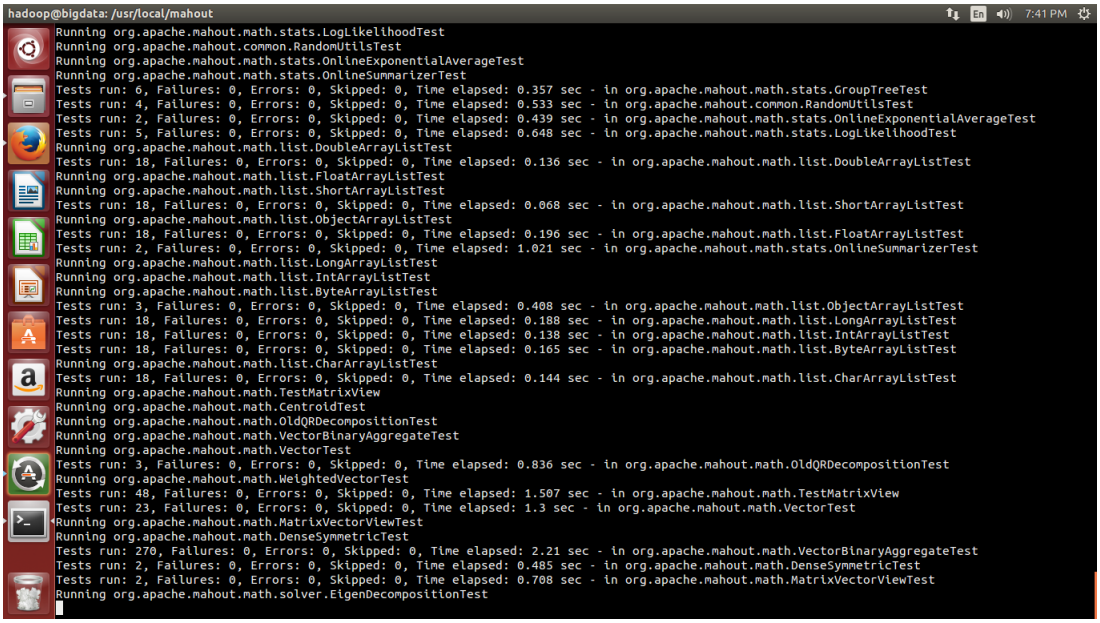

Figure 4.8: Perform Tests

Step 4:

Then it would perform the test , it's recommended that the complete tests should be done and the process should be completed for the first time installation.. later when we run the mvn install we use a command to skip the tests which is

mvn install -Dmaven.test.skip=true.

Step 5:

Once the tests are done and the mahout is built ..we get a success message as shown in the figure

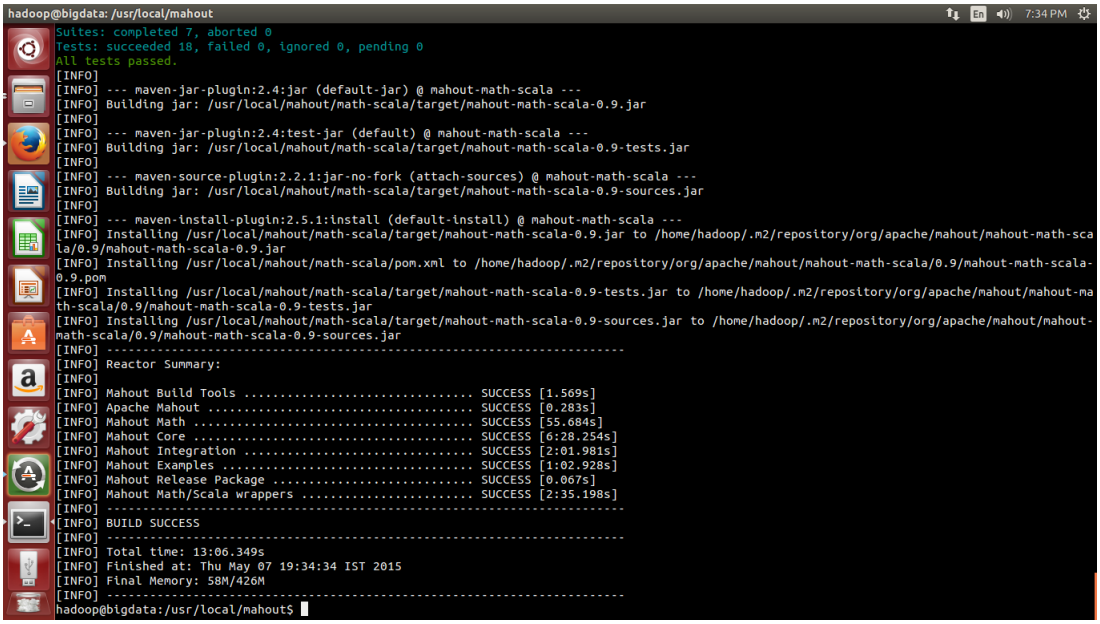

Figure 4.9: Built Mahout

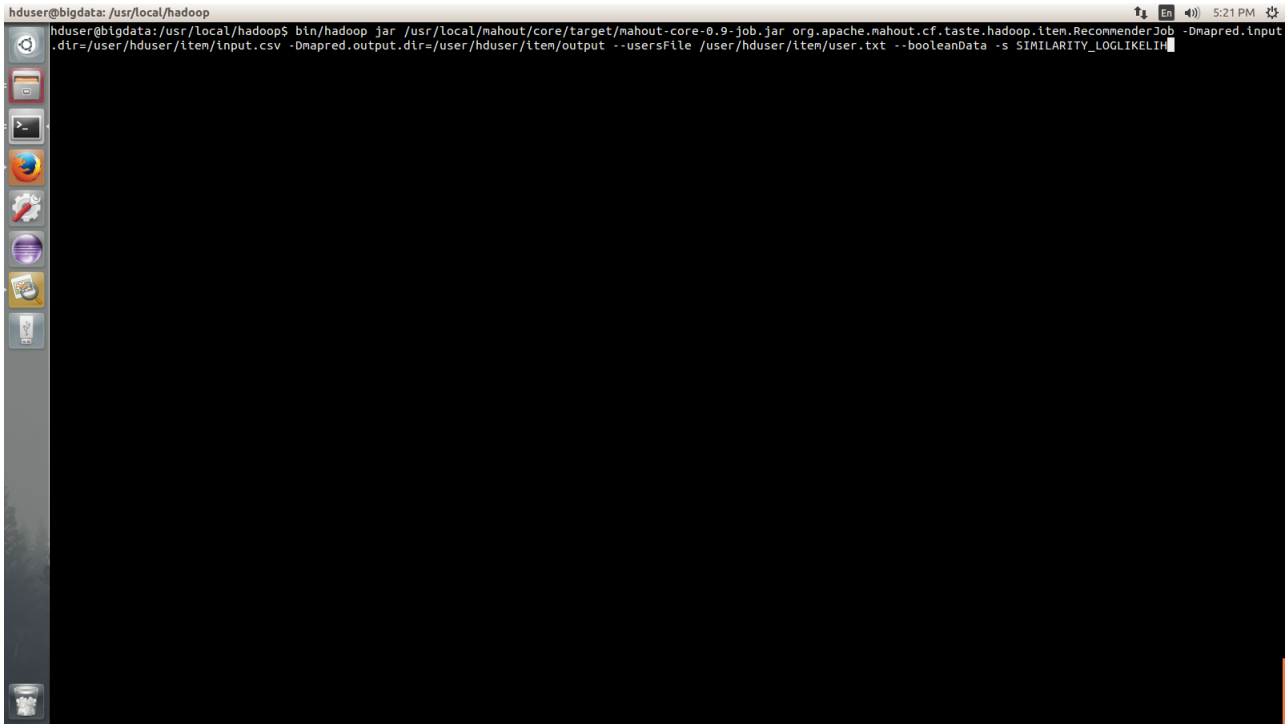

Figure 4.10: Command To Run Loglikelihood

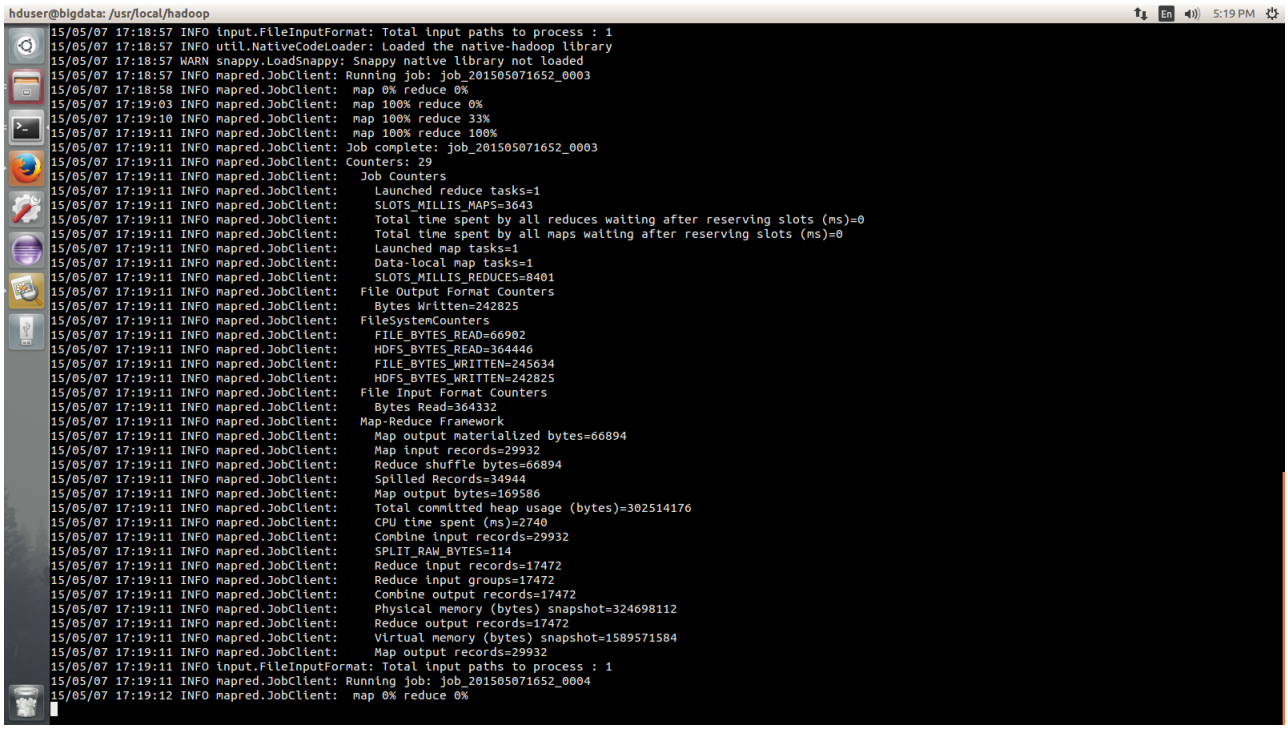

Figure 4.11: Job in Process

	Event A	Everything but A
Event B	A and B together (k11)	B, but not A (k12)
Everything but B	A without B (k21)	Neither A nor B (k22)

Table 4.1: LogLikelihood Table

4.4.1 Python Module Installation:

Argparse Installation: Run the commands given below
python setup.py install
easy_install argparse
pip install argparse
xml.etree Module Installation:
Run the commands
python setup.py install
easy_install xml.etree
pip install xml.etree

4.5 Algorithms

4.5.1 Log Likelihood

To compute LogLikelihood, we find the number of times the events occurred together (let's call this k11), the number of times each has occurred without the other (let's call these k12 and k21) and the number of times something has been observed that was neither of these events (let's call this k22). Here, row or column 1 is the event of interest and row or column 2 is everything else.

Once you have these counts, the hard part is over. Computing the log-likelihood ratio score (also known as G2) is very simple,

LLR = 2 sum(k) (H(k) - H(rowSums(k)) - H(colSums(k)))

where H is Shannon's entropy, computed as the sum of (kij / sum(k)) log (kij / sum(k)). In R, this function is defined as

H = function(k) N = sum(k) ; return (sum(k/N * log(k/N + (k==0))))

The trick with adding k==0 handles the case where some element of k is zero. Since we are multiplying by that same zero, we can drop those terms but it helps not to try to compute the log(0). The java or C versions of this (ask me for a copy) is almost as

simple, but not quite as pretty.
Method Details:
setPreferenceInferrer:
public void setPreferenceInferrer(PreferenceInferrer inferrer)
Description
Attaches a PreferenceInferrer to the implementation.
Specified by:
setPreferenceInferrer in interface UserSimilarity
Parameters:
inferrer - PreferenceInferrer(Implementations of this interface compute an inferred preference for a user and an item that the user has not expressed any preference for. This might be an average of other preferences scores from that user, for example. This technique is sometimes called "default voting".)
userSimilarity:
public double userSimilarity(long userID1, long userID2)throws TasteException
Description
Returns the degree of similarity, of two users, based on the their preferences.
Specified by:
userSimilarity in interface UserSimilarity
Parameters:
userID1 - first user ID
userID2 - second user ID
Returns:
similarity between the users, in [-1,1] or Double.NaN similarity is unknown
Throws:
NoSuchUserException - if either user is known to be non-existent in the data
TasteException- if an error occurs while accessing the data
itemSimilarity
public double itemSimilarity(long itemID1,long itemID2)throws TasteException
Description copied from interface: ItemSimilarity
Returns the degree of similarity, of two items, based on the preferences that users have expressed for the items.
Specified by:
itemSimilarity in interface ItemSimilarity
Parameters:
itemID1 - first item ID
itemID2 - second item ID
Returns:
similarity between the items, in [-1,1] or Double.NaN similarity is unknown
Throws:

NoSuchItemException - if either item is known to be non-existent in the data
TasteException - if an error occurs while accessing the data

itemSimilarities

public double[] itemSimilarities(long itemID1,long[] itemID2s) throws TasteException
Description copied from interface: ItemSimilarity
A bulk-get version of ItemSimilarity.itemSimilarity(long, long).
Specified by:
itemSimilarities in interface ItemSimilarity
Parameters:
itemID1 - first item ID
itemID2s - second item IDs to compute similarity with
Returns:
similarity between itemID1 and other items
Throws:
NoSuchItemException - if any item is known to be non-existent in the data
TasteException - if an error occurs while accessing the data

refresh

public void refresh(Collection¡Refreshable¿ alreadyRefreshed)
Description copied from interface: Refreshable
Triggers "refresh" – whatever that means – of the implementation. The general contract is that any should always leave itself in a consistent, operational state, and that the refresh atomically updates internal state from old to new.
Specified by:
refresh in interface Refreshable
Overrides:
refresh in class AbstractItemSimilarity
Parameters:
alreadyRefreshed - s that are known to have already been refreshed as a result of an initial call to a method on some object. This ensure that objects in a refresh dependency graph aren't refreshed twice needlessly.

toString

public String toString()
Overrides:
toString in class Object

4.5.2 Cooccurrence Similarity

Co-occurrence data is ubiquitous in modern data mining and machine learning applications as it provides a very rich signal source for inferring similarity between items, a common prediction task.

Method Details:

similarity:

public double similarity(double dots,double normA,double normB,int numberOfColumns)

consider

public boolean consider(int numNonZeroEntriesA,int numNonZeroEntriesB,double maxValueA,double maxValueB,double threshold)

Specified by:

consider in interface VectorSimilarityMeasure

Overrides:

consider in class CountbasedMeasure

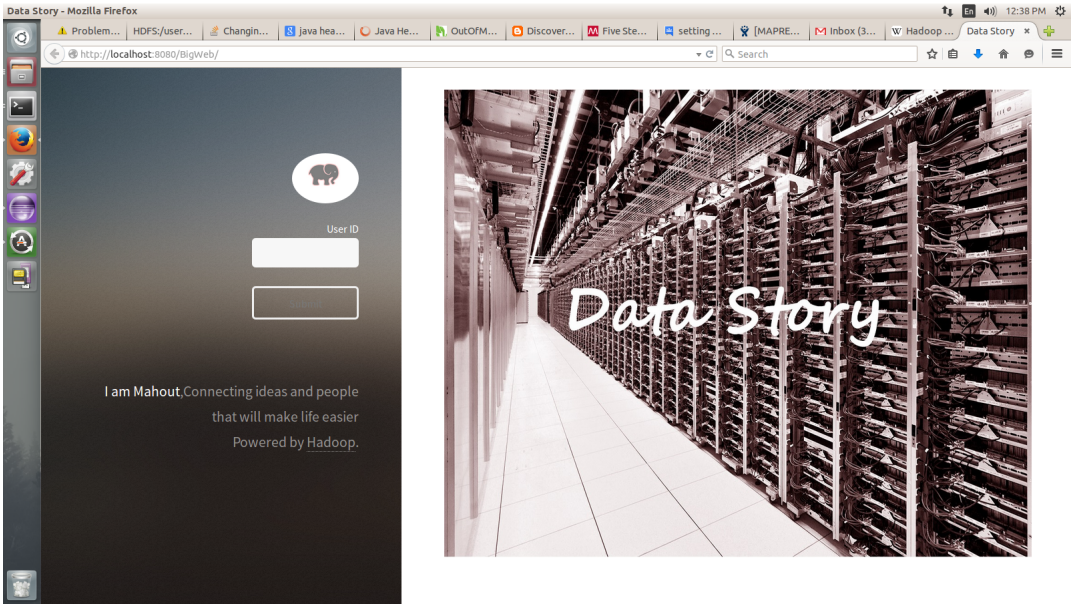

Figure 4.12: Home page

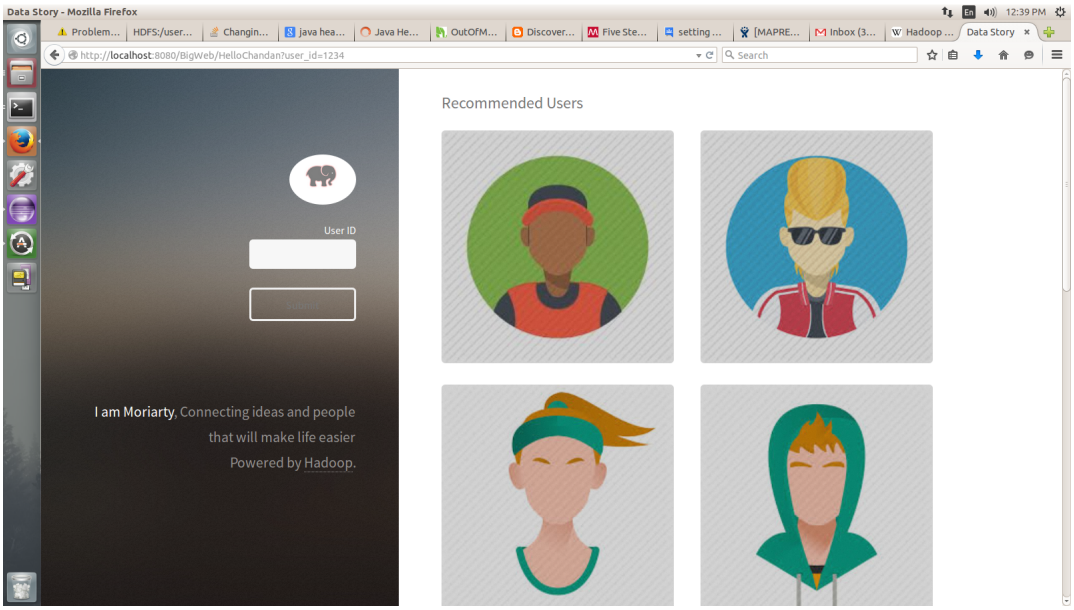

Figure 4.13: User Interface: user recommender

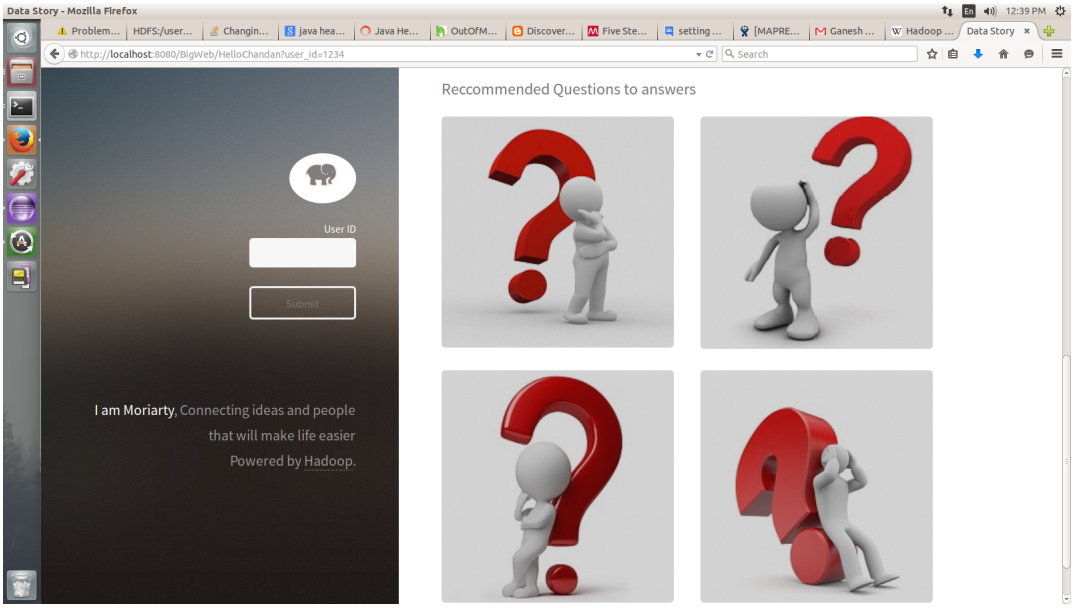

Figure 4.14: User Interface: question recommender

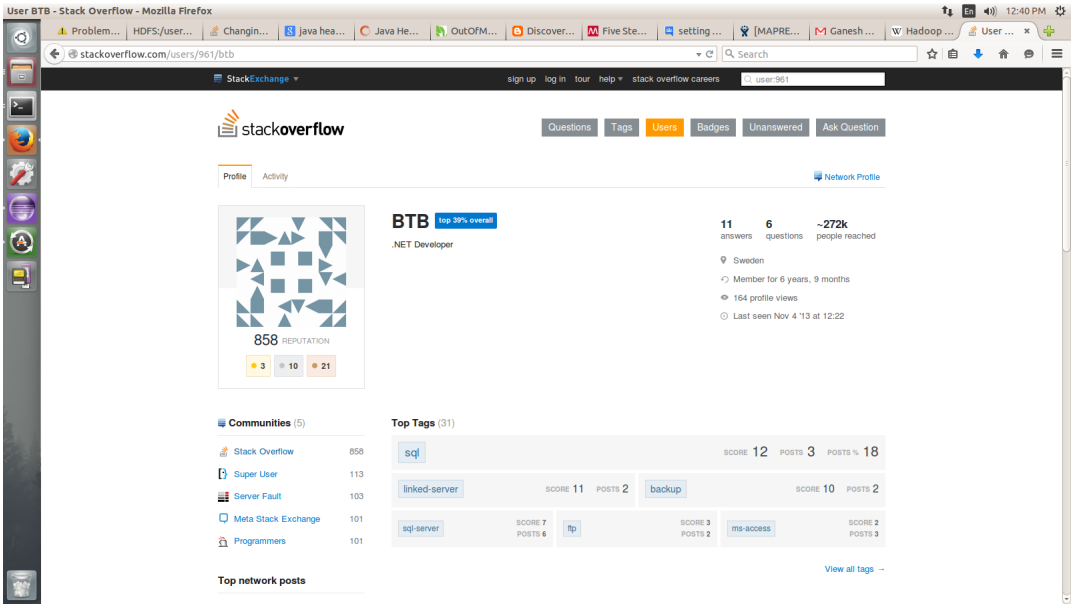

Figure 4.15: A recommended user

Chapter 5

Software Testing

1) Process: Continuous testing of the recommender, with bulk load proved successful from 6gb to upto 55gb.

2) All Life Cycle Activities: Testing was conducted throughout the Software Development Life Cycle (SDLC). Data was made available by parsing and in each step quality of data was tested against boundry scenerios to find any defect whatsoever. The testing also included documents such as the requirements and design specifications.

3) Static Testing: This testing includes reviewing of the documents (including source code) and static analysis. Daily reviews were conducted based on quality of code.

4) Dynamic Testing: In dynamic testing we executed the software code to demonstrate the results ran successfully. It was done during validation process. It included unit testing with JUnit, integration testing, system testing, etc.

5) Preparation: Tests were conducted on data query, parser cleansing quality of recommender output were also tested explicitly.

6) Evaluation: Various UserId were selected from user.csv and based on reputation their recommender algorithms were tested against real cases.

7) Software products testing: Requirement and design specifications were tested, also the related documents like operation, user and training material were also considered.

After having successfully setup the system, we test the accuracy of output generated. in one of the scenarios, we manually checked the recommendations provided by the system by comparing the interests of the user in real time with the suggestions provided by Data Story. The output from the Mahout framework contains the level of precision of the output and it is in descending order. So, the first recommendation is the strongest candidate for our recommender. With the help of our simple GUI, we can view the real time profile of the user and make sure the prediction provided by Data Story is correct. If no recommendation can be provided, we forward the system to a null page which displays a page not found error in the Stack Exchange website.

Chapter 6

Conclusion and Future Work

6.1 Conclusion and Future Work

Q & A User group is a vast and ever increasing community. The Q & A Usergroup Analyzer provides an interesting, alternative and efficient method in order to recommend appropriately. The two parts of this project include recommendation of questions to Users based on their previous history of answering. This recommendation follows from the Pearson Co-relation algorithm. Secondly, we recommend Users to other Users in order to build a global community based on common interest. This is achieved by log-likelihood algorithm.

The recommender system has the overload of processing huge data. Therefore, the use of hadoop framework is appropriate.

In the future, we would like to extend the processing ability to high number of processors through distributed operating system creating multi-node clusters. This would scale well with our project.

Bibliography

[1] E Lezina cG Galina and Artem M Kuznetsov. Predict closed questions on stackoverflow. 2013. 5

[2] Derrick Cheng, Michael Schiff, and Wei Wu. Eliciting answers on stackoverflow. 2013. 5

[3] Rakesh Kumar, Neha Gupta, Shilpi Charu, Somya Bansal, and Kusum Yadav. Comparison of sql with hiveql. *International Journal for Research in Technological Studies*, 1(9):2348–1439, 2014. 6

[4] Sangeeta Lal, Denzil Correa, and Ashish Sureka. Miqs: Characterization and prediction of migrated questions on stackexchange. 4

[5] Klemens Muthmann and Alina Petrova. An automatic approach for identifying topical near-duplicate relations between questions from social media q/a sites. 5

[6] Sebastian Schelter and Sean Owen. Collaborative filtering with apache mahout. *Proc. of ACM RecSys Challenge*, 2012. 7

[7] Dennis Schenk and Mircea Lungu. Geo-locating the knowledge transfer in stackoverflow. In *Proceedings of the 2013 International Workshop on Social Software Engineering*, pages 21–24. ACM, 2013. 6

[8] Logan Short, Christopher Wong, and David Zeng. Tag recommendations in stackoverflow. 5

[9] Bogdan Vasilescu, Andrea Capiluppi, and Alexander Serebrenik. Gender, representation and online participation: A quantitative study of stackoverflow. In *Social Informatics (SocialInformatics), 2012 International Conference on*, pages 332–338. IEEE, 2012. 6