CMPE 281: CLOUD TECHNOLOGIES

FALL 2013

Project Report on

Comparing Amazon EMR and Local Hadoop Map Reduce Instance

Submitted By

Vaibhav Bhor (009313434)
Sumant Murke (009303879)
Abhiram Uppaluri (008622380)
Prachi Gotkhindikar (008630947)
Srividya Reddy Thota (008656206)

Submitted To

Prof. Mike Larkin

Abstract

Big data analytics is all about collecting, storing, processing, and visualizing massive amounts of data so that companies can distill knowledge from it, derive valuable business insights from that knowledge, and make better business decisions, all as quickly as possible. The main challenges in operating data analysis platforms include installation and operational management, dynamically allocating data processing capacity to accommodate for variable load, and aggregating data from multiple sources for holistic analysis. The Open Source Apache Hadoop and its ecosystem of tools help solve these problems because Hadoop can expand horizontally to accommodate growing data volume and can process unstructured and structured data in the same environment. Setting up an on premise Hadoop cluster involves buying new hardware, racking them up and configuring them for optimal performance. This involves a lot of capital expenditure as well as dedicated expertise to maintain all. Amazon Web Services (AWS) accelerates the process of big data analytics or any other data -intensive activities through its Elastic MapReduce service. Amazon Elastic MapReduce (EMR) is one such service that provides fully managed hosted Hadoop framework on top of Amazon Elastic Compute Cloud (EC2). EMR allows focusing on analytics instead of infrastructure as it provides instant scalability and elasticity. In this project we try to analyze how a local instance of Apache Hadoop compares to Amazon's EMR service. The report highlights the in depth study conducted taking into account several factors such elasticity, cost effectiveness, reliability , security, flexibility to claim that using Amazon EMR offers impressive gains over deploying an Hadoop cluster locally.

Contents

State of Art

Title: Comparing Apache Hadoop and Amazon EMR with an example of sentiment analysis.

The constant influx of digital information fueled by billions of mobile phones, tens of billions of posts over social media and an ever expanding array of networked sensors from cars, shop-floor equipments, point of sale terminals, utility meters and many other sources has been the main motivation behind Apache Hadoop that presents a cost effective way to analyze all this unstructured data. This is usually called as Big data which is distributed over multiple machines as storage. Hadoop is based on massively scalable, parallel processing framework called MapReduce. Most of the Hadoop jobs involve a large set of input files that generate a large set of output files. In order to queue these jobs to run in batch organizations need to process large amounts of data. The input to the Map/Reduce is to split the input data into independent chunks that are processed in a parallel manner. It then sorts the output of the maps which then serves as an input to the reduce tasks. Amazon EMR (Amazon's elastic Map Reduce) enhances Hadoop by backing it up with the Amazon's web services. EMR allows distributing the computational work across clusters of virtual servers running in Amazon Cloud.

The principal aim of this research is an in-depth analysis of how a local instance of Apache Hadoop compares to Amazon's EMR service. Taking the use case of 'Sentiment analysis', that is used to determine whether a given text conveys a positive, negative or neutral sentiment, we try to analyze the effectiveness of Amazon EMR. With task such as reviewing sentiments that involves gleaning insights from structured and unstructured data we try to claim that using Amazon's EMR can provide impressive gains over deploying on Hadoop cluster locally.

Initial study about key concepts:

Apache Hadoop and MapReduce:

Apache Hadoop is an open source software framework for storing large set of the data on clusters. It consists of the various modules, one of which is Hadoop MapReduce which is programming module for parallel processing of the large data. The module is built on the basic assumption of the failure of commodity software which in turn causes critical data loss. MapReduce is key algorithm that the MapReduce engine users used to distribute work around 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. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks.

Algorithm:

1. Unstructured data is given input to map function which splits the input data into independent chunks which process vast amounts of data in parallel on large clusters of commodity hardware in a reliable, fault-tolerant manner.
2. Parallel processing by map task is done whose output is mapped by framework as an input to reduce task.
3. Output is reduced with help of keys used during map function. So final output can be combined results from reduce.

MapReduce is great for loosely coupled parallelization tasks. It is not well suited to situations where tight coupling (e.g. message passing or shared memory, or large graph processing algorithms). It also does better where the amount of computation required is significantly more than any one computer

Amazon EMR (Elastic Map Reduce)

Amazon EMR is a web service that provides businesses, researchers, data analysts, and developers to easily and cost-effectively process vast amounts of data. It uses a Hadoop framework that runs over infrastructure of Amazon Elastic Compute Cloud (Amazon EC2) and Amazon Simple Storage Service (Amazon S3). It provides Hadoop as service where it is integrated with AWS services.

Amazon EMR significantly reduces the complexity of the time-consuming set-up, management and tuning of Hadoop clusters or the compute capacity upon which they sit. We can instantly create large number of Hadoop clusters which will eventually start processing within few clicks and minutes. When your cluster finishes its processing, it will be automatically torn down so you stop paying. Using this service we can quickly perform various data analytics tasks for applications such as web indexing, data mining, log file analysis, machine learning, financial analysis, scientific simulation, and bioinformatics research.

Example as Sentiment Analysis:

Sentiment Analysis on Apache Map Reduce and AWS elastic Map Reduce using a real time data and running map reduce program. Some of our goals are

1. Build a map reduce program for the real time data to get a clear understanding of the Hadoop map reduce systems in the AWS and systems setup locally.
2. Compare the time, cost and effort taken to setup hadoop environment in AWS EC2 using EMR service and locally using Apache Hadoop.
3. Run a map reduce program on the systems and check the performance (time taken to successfully completed a map reduce job)
4. Check the sustainability with the help of giving a large size of input and running the map reduce program.

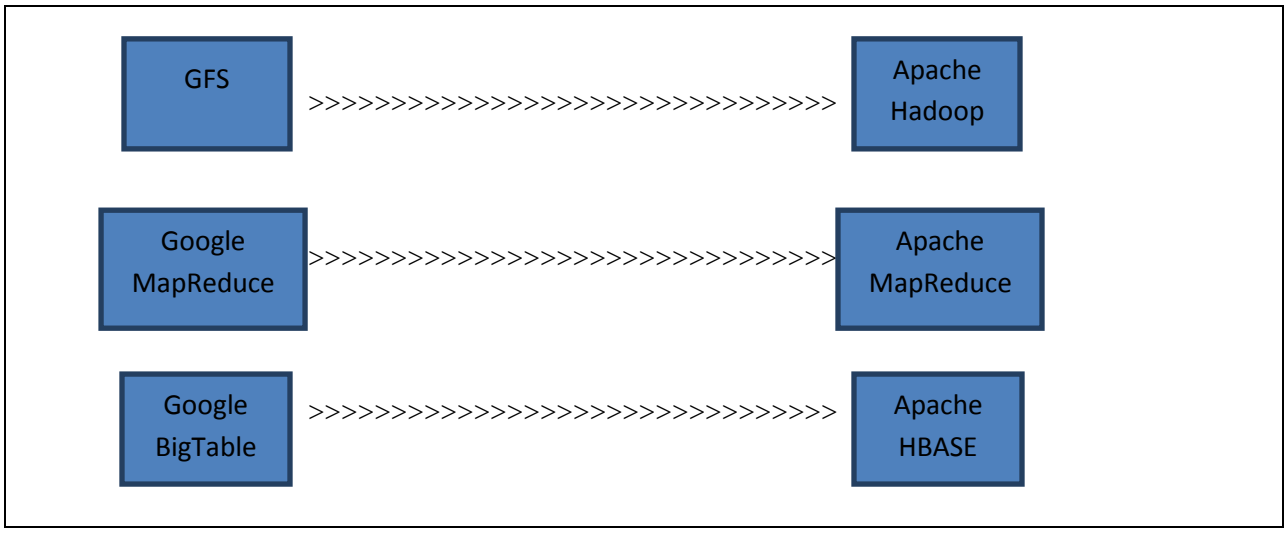
Claim: On the basic of study done for implementation of Apache Hadoop Map reduce in compared to Amazon ECR service usage for running sentiment analysis of social networking site, we think that using Amazon EMR is better than creating Hadoop structure in the own infrastructure as far as scaling and configuration perspectives are concerned.

Rough Answer: On the basic analysis of the key concepts like Apache map reduce and Amazon's EMR, we should be able to prove our claim about having edge of Amazon's EMR over Apache Hadoop map reduce because:

1. We could instantly provision capacity for performing data-intensive tasks.
2. It lets us focus on core work of data analysis rather than time-consuming set-up, and management of Hadoop clusters.
3. We could customize clusters according to our needs.
4. Ease in configuration and usage of the functionality than local infrastructure.

1. Introduction

Hadoop origin comes from Google's white papers. Google file system, Google map reduce and Google big table which become Apache HDFS, Apache MapReduce and Apache HBASE respectively. About 95% of the architecture described in Google's white papers is faithfully implemented in Apache Projects.



The Apache Hadoop Project works on the development of open-source software tools for scalable, reliable, distributing computing.

The Apache Hadoop software library is a frame work which is built for storage and processing of large amount of data in a clustered environment using simple programming models. It can scale up easily to thousands of machines. Each machine is capable of storage and processing of data. The framework takes care of handling defects at application layer rather than depending on hardware. The Project includes these modules

1. Hadoop Commons
2. Hadoop Distributed file system (HDFS)
3. Hadoop YARN
4. Hadoop MapReduce

2. Architecture

The Hadoop commons package provides file system and OS abstraction levels. It also contains the JAR files and scripts needed to start Hadoop. A small Hadoop cluster includes a master node and multiple slave nodes. The master node contains Data Node, Name Node, Job Tracker, Task tracker. A slave or worker node acts as data node and task tracker.

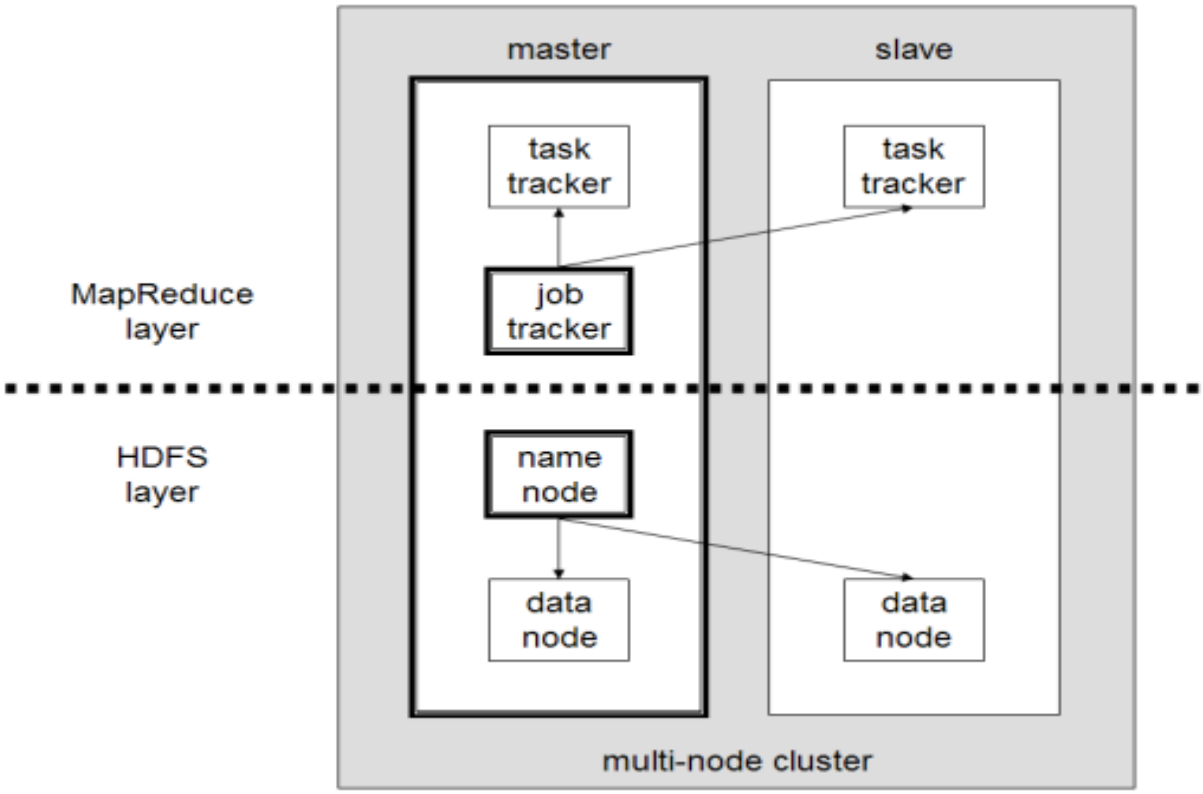


Figure: Hadoop Architecture

Hadoop Distributed File System:

The HDFS is a scalable, distributed and portable file system written in Java for the Hadoop framework. The file systems uses TCP/IP layer for communication. The data is divided in to blocks and gets stored in the multiple data nodes. The default size of a block is 64 MB. HDFS also replicates data on multiple hosts and hence it doesn't require any RAID storage on hosts. By default, the replication factor is three, two in the same rack and one in the different rack.

HDFS also has a secondary name node which may mislead people thinking that when the primary name node goes down, the secondary name node takes over. The secondary name node takes check point

images from the primary name node. These check point images are used for restart a failed primary name node.

An Advantage of HDFS is the data awareness between job tracker and task tracker.

HDFS is designed for processing the data in bulk and may not be suitable for systems with concurrent IO operations.

Map Reduce Engine

On top of HDFS, There is a map reduce engine, which consists of one job tracker to which client submits the MR jobs. The Job tracker pushes out the work to available task tracker nodes in the cluster. With the data awareness of the job tracker, it strives to keep data as close as possible to data. If in some cases, the actual node where the data resides cannot take up the work, then the preferences goes to a node in the same rack. A heartbeat is sent to the task tracker from job tracker in a certain time interval to check the status. The job is not marked as complete until all the tasks in a job are completed.

The task tracker can take up assigned number of slots and hence there is no consideration for the system load. Another limitation is one slow task tracker can slow down the entire processing. However, a single task can be executed on multiple nodes.

How MapReduce Works?

There are four high level tasks to be completed. These are

- Job Submission: The Client Submits the MapReduce Job
- Job Initialization: The job tracker which coordinates the task receives a call to submitJob() method, it puts into an internal queue, from where the job scheduler picks up and initialize it
- Task Assignment: Task tracker runs a loop periodically which sends the heartbeat to the job tracker. The heartbeat tells whether the task tracker is ready to take up the new job and based on that the job is assigned to a task tracker. The same job can be assigned to multiple task trackers.
- Task Execution: Firstly, it copies the jar to the local HDFS file system. Secondly, it creates a directory and unzips all the files; thirdly, it creates an instance of TaskRunner to run the task. The task is executed in two phases, Mapper and Reducer.

3. Hadoop Installation and Task Implementation on local machine:

A. Download and install Java:

```
$ sudo apt-get install openjdk-7-jdk
$ java -version
java version "1.7.0_25"
OpenJDK Runtime Environment (IcedTea 2.3.12) (7u25-2.3.12-4ubuntu3)
OpenJDK 64-Bit Server VM (build 23.7-b01, mixed mode)
$ cd /usr/lib/jvm
$ ln -s java-7-openjdk-amd64 jdk

$ sudo apt-get install openssh-server
```

B. Add Hadoop Group and User:

```
$ sudo addgroup hadoop
$ sudo adduser --ingroup hadoop hduser
$ sudo adduser hduser sudo
```

C. Setup SSH Certificate

```
$ ssh-keygen -t rsa -P ''
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
$ ssh localhost
```

D. Download Hadoop

```
$ cd ~
$ wget http://www.trieuvan.com/apache/hadoop/common/hadoop-
2.2.0/hadoop-2.2.0.tar.gz$ sudo tar vxzf hadoop-2.2.0.tar.gz -C
/usr/local
$ cd /usr/local
$ sudo mv hadoop-2.2.0 hadoop
$ sudo chown -R hduser:hadoop hadoop
```

E. Set up Hadoop Environment Variable:

a. Append following at the end of `.bashrc` file in vi editor:

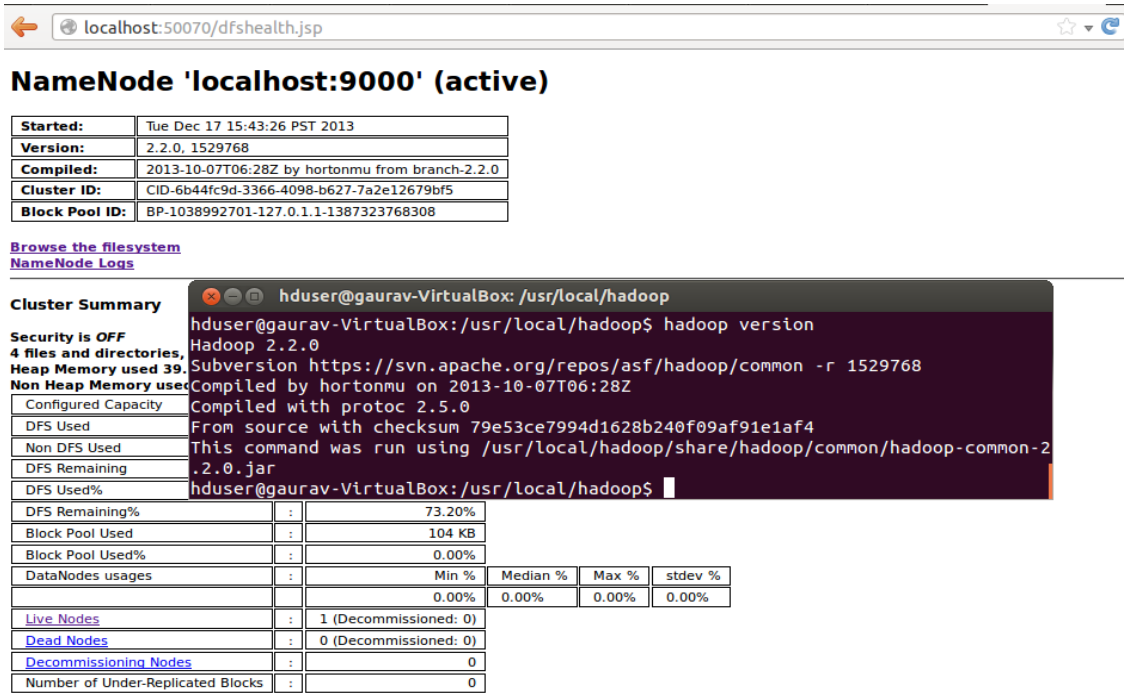
```
#Hadoop variables
export JAVA_HOME=/usr/lib/jvm/jdk/
export HADOOP_INSTALL=/usr/local/hadoop
export PATH=$PATH:$HADOOP_INSTALL/bin
export PATH=$PATH:$HADOOP_INSTALL/sbin
export HADOOP_MAPRED_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_HOME=$HADOOP_INSTALL
export HADOOP_HDFS_HOME=$HADOOP_INSTALL
export YARN_HOME=$HADOOP_INSTALL
###end of paste
```

b. `$ cd /usr/local/hadoop/etc/hadoop`

c. `$ vi hadoop-env.sh`

```
// Modify hadoop-env.sh file with modification of JAVA_HOME
#modify JAVA_HOME
export JAVA_HOME=/usr/lib/jvm/jdk/
```

F. Relogin again to linux machine and check if Hadoop installed and version.



So, we have installed Hadoop successfully.

G. Configure hadoop

a. `core-site.xml changes:`

```
#Paste following between <configuration>

<property>
   <name>fs.default.name</name>
   <value>hdfs://localhost:9000</value>
</property>
```

b. `yarn-site.xml changes:`

```
#Paste following between <configuration>

<property>
   <name>yarn.nodemanager.aux-services</name>
   <value>mapreduce_shuffle</value>
</property>
<property>
   <name>yarn.nodemanager.aux-
services.mapreduce.shuffle.class</name>
   <value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>
```

c. $ mv mapred-site.xml.template mapred-site.xml

d. Changes for mapred-site.xml
#Paste following between <configuration>

```
<property>
   <name>mapreduce.framework.name</name>
   <value>yarn</value>
</property>
```

e. Make directories for name and data nodes:

```
$ cd ~
$ mkdir -p mydata/hdfs/namenode
$ mkdir -p mydata/hdfs/datanode
$ cd /usr/local/hadoop/etc/hadoop
```

f. Changes for hdfs-site.xml
#Paste following between <configuration> tag

```
<property>
   <name>dfs.replication</name>
   <value>1</value>
 </property>
 <property>
   <name>dfs.namenode.name.dir</name>
   <value>file:/home/hduser/mydata/hdfs/namenode</value>
 </property>
 <property>
   <name>dfs.datanode.data.dir</name>
   <value>file:/home/hduser/mydata/hdfs/datanode</value>
 </property>
```

H. Format NameNode:

```
$ hdfs namenode –format
```

I. Start Hadoop Service:

```
$ start-dfs.sh
$ start-yarn.sh
```

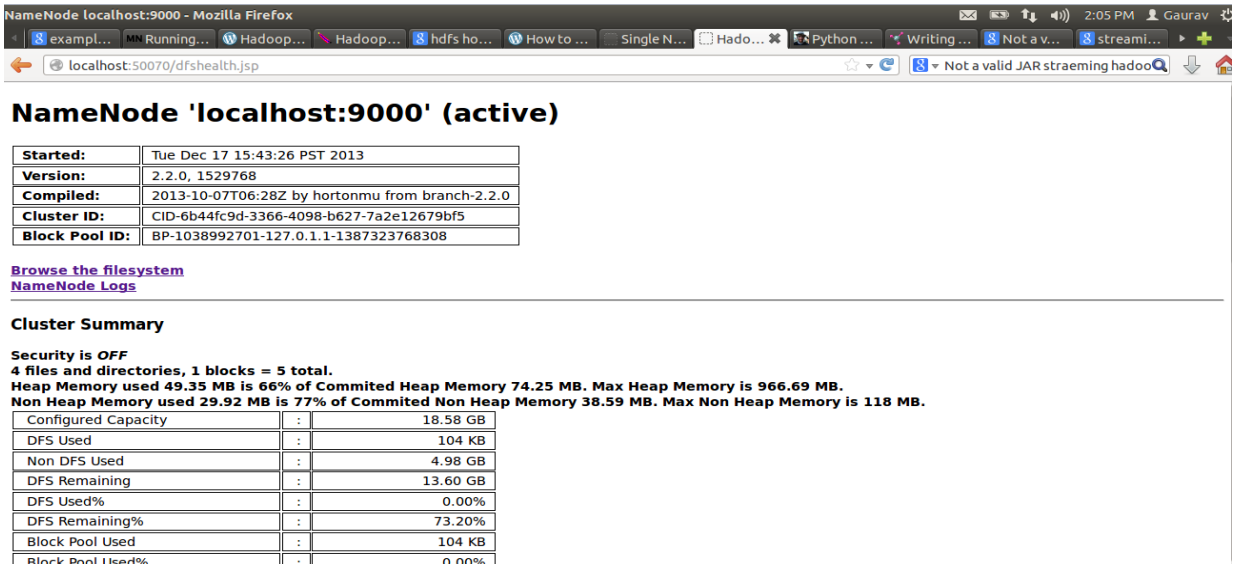
```
hduser@gaurav-VirtualBox: /usr/local/hadoop
hduser@gaurav-VirtualBox:/usr/local/hadoop$ start-dfs.sh
Starting namenodes on [localhost]
hduser@localhost's password:
localhost: starting namenode, logging to /usr/local/hadoop/logs
/hadoop-hduser-namenode-gaurav-VirtualBox.out
hduser@localhost's password:
localhost: starting datanode, logging to /usr/local/hadoop/logs
/hadoop-hduser-datanode-gaurav-VirtualBox.out
Starting secondary namenodes [0.0.0.0]
hduser@0.0.0.0's password:
0.0.0.0: starting secondarynamenode, logging to /usr/local/hado
op/logs/hadoop-hduser-secondarynamenode-gaurav-VirtualBox.out
hduser@gaurav-VirtualBox:/usr/local/hadoop$ start-yarn.sh
starting yarn daemons
starting resourcemanager, logging to /usr/local/hadoop/logs/yar
n-hduser-resourcemanager-gaurav-VirtualBox.out
hduser@localhost's password:
localhost: starting nodemanager, logging to /usr/local/hadoop/l
ogs/yarn-hduser-nodemanager-gaurav-VirtualBox.out
```

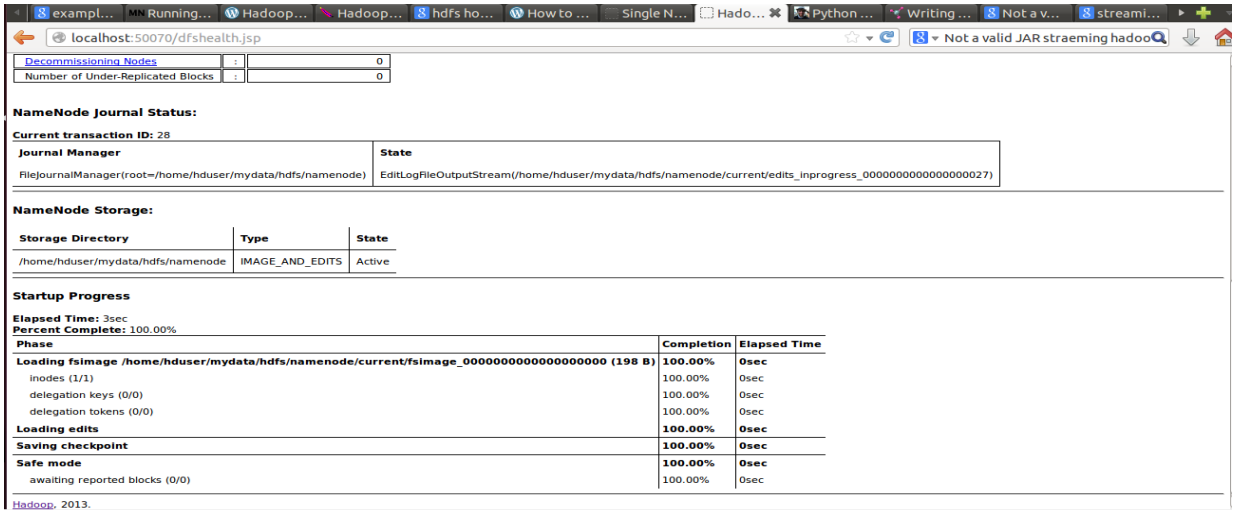
Now we could see following services running:

```
$ jps
```

```
hduser@gaurav-VirtualBox: /usr/local/hadoop
hduser@gaurav-VirtualBox:/usr/local/hadoop$ jps
3114 ResourceManager
2445 NameNode
10766 Jps
3326 NodeManager
2664 DataNode
2909 SecondaryNameNode
hduser@gaurav-VirtualBox:/usr/local/hadoop$
```

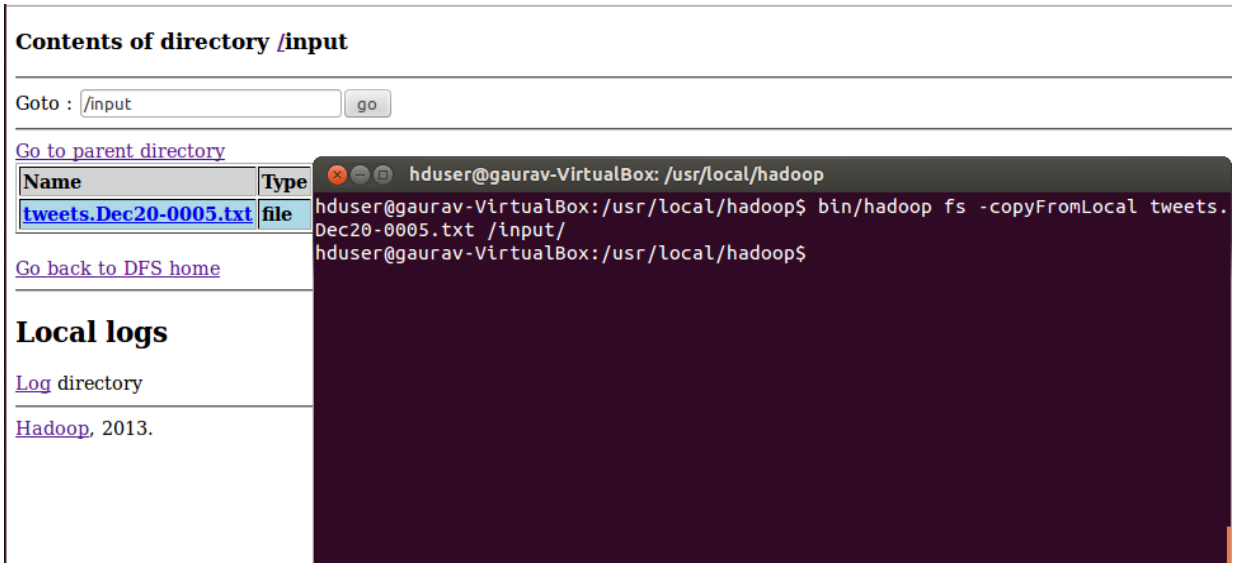
J. **View Name node details on default link localhost:50070 as follows:**





K. Copy input twitter text file from local directory to hdfs:

```
$bin/hadoop fs –copyFromLocal tweets.Dec19-0921.txt /users/
```

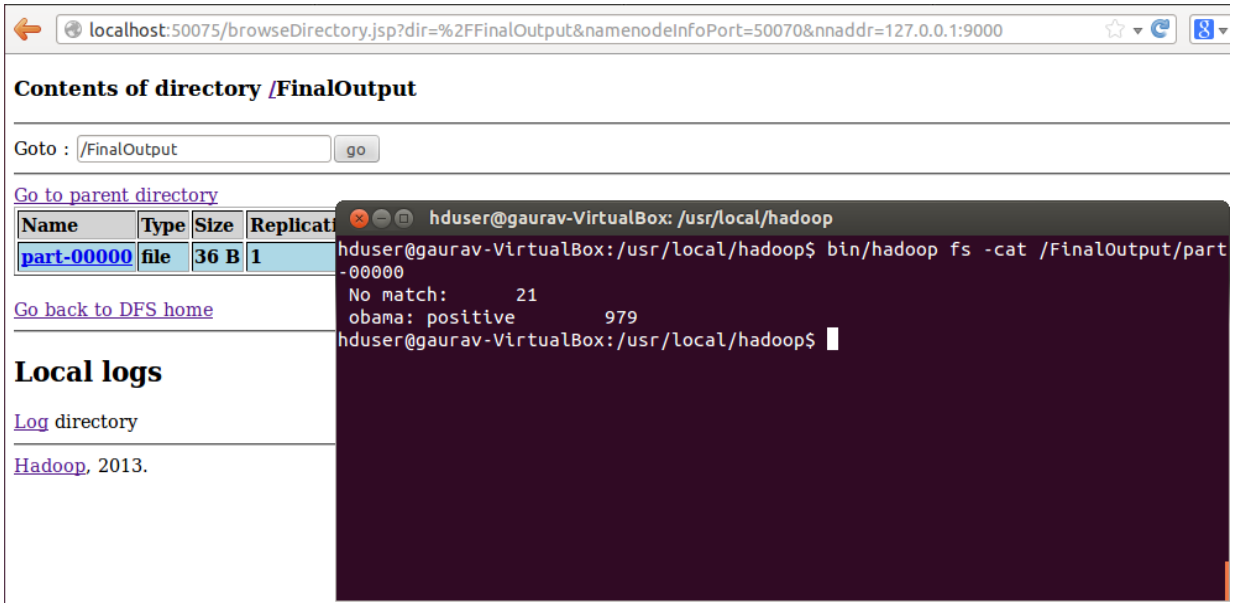


L. Run Map Reduce task as follows:

```
$ bin/hadoop jar hadoop-streaming-1.0.3.jar -input "/input/tweets.Dec20-0005.txt"
-output "/FinalOutput/" \-file "$PWD/sentimentmapper.py"  -mapper "python
$PWD/sentimentmapper.py" –reducer aggregate
```

```
hduser@gaurav-VirtualBox:/usr/local/hadoop$  bin/hadoop jar hadoop-streaming-1.0.3.jar -input "/input/tweets.Dec20-0005.txt" -output "/FinalOutp
ut/" -file "$PWD/sentimentmapper.py"  -mapper "python $PWD/sentimentmapper.py" -reducer aggregate
13/12/20 19:06:03 INFO Configuration.deprecation: mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
packageJobJar: [/usr/local/hadoop/sentimentmapper.py] [] /tmp/streamjob7123771897233277581.jar tmpDir=null
13/12/20 19:06:03 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
13/12/20 19:06:04 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
13/12/20 19:06:06 INFO mapred.FileInputFormat: Total input paths to process : 1
13/12/20 19:06:07 INFO mapreduce.JobSubmitter: number of splits:2
13/12/20 19:06:07 INFO Configuration.deprecation: user.name is deprecated. Instead, use mapreduce.job.user.name
13/12/20 19:06:07 INFO Configuration.deprecation: mapred.jar is deprecated. Instead, use mapreduce.job.jar
13/12/20 19:06:07 INFO Configuration.deprecation: mapred.output.value.class is deprecated. Instead, use mapreduce.job.output.value.class
13/12/20 19:06:07 INFO Configuration.deprecation: mapred.mapoutput.value.class is deprecated. Instead, use mapreduce.map.output.value.class
13/12/20 19:06:07 INFO Configuration.deprecation: mapred.job.name is deprecated. Instead, use mapreduce.job.name
13/12/20 19:06:07 INFO Configuration.deprecation: mapred.input.dir is deprecated. Instead, use mapreduce.input.fileinputformat.inputdir
13/12/20 19:06:07 INFO Configuration.deprecation: mapred.output.dir is deprecated. Instead, use mapreduce.output.fileoutputformat.outputdir
13/12/20 19:06:07 INFO Configuration.deprecation: mapred.map.tasks is deprecated. Instead, use mapreduce.job.maps
13/12/20 19:06:07 INFO Configuration.deprecation: mapred.output.key.class is deprecated. Instead, use mapreduce.job.output.key.class
13/12/20 19:06:07 INFO Configuration.deprecation: mapred.mapoutput.key.class is deprecated. Instead, use mapreduce.map.output.key.class
13/12/20 19:06:07 INFO Configuration.deprecation: mapred.working.dir is deprecated. Instead, use mapreduce.job.working.dir
13/12/20 19:06:07 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1387586885918_0003
13/12/20 19:06:08 INFO impl.YarnClientImpl: Submitted application application_1387586885918_0003 to ResourceManager at /0.0.0.0:8032
13/12/20 19:06:08 INFO mapreduce.Job: The url to track the job: http://gaurav-VirtualBox:8088/proxy/application_1387586885918_0003/
13/12/20 19:06:08 INFO streaming.StreamJob: getLocalDirs(): [/tmp/hadoop-hduser/mapred/local]
13/12/20 19:06:08 INFO streaming.StreamJob: Running job: job_1387586885918_0003
13/12/20 19:06:08 INFO streaming.StreamJob: Job running in-process (local Hadoop)
13/12/20 19:06:10 INFO streaming.StreamJob:  map 0%  reduce 0%
13/12/20 19:07:19 INFO streaming.StreamJob:  map 100%  reduce 0%
13/12/20 19:07:29 INFO streaming.StreamJob:  map 0%  reduce 0%
13/12/20 19:08:30 INFO streaming.StreamJob:  map 100%  reduce 0%
13/12/20 19:08:34 INFO streaming.StreamJob:  map 0%  reduce 0%
13/12/20 19:09:19 INFO streaming.StreamJob:  map 100%  reduce 0%
13/12/20 19:09:22 INFO streaming.StreamJob:  map 0%  reduce 0%
13/12/20 19:10:21 INFO streaming.StreamJob:  map 100%  reduce 0%
13/12/20 19:10:27 INFO streaming.StreamJob:  map 100%  reduce 100%
```

M. Once completed, check output as follows:



4. Amazon EMR

Amazon Elastic Map Reduce is a data analysis tool that simplifies the task of set up and management of compute clusters and helps to implement the data processing jobs quickly. It works in conjunction with the Amazon EC2 to set up Hadoop clusters and Amazon S3 to store data files, log files, scripts and output results. The EMR service and Hadoop enable big data processing with ease, low cost, scalability. It removes the cumbersome task of setting up hardware and network required by the Hadoop cluster, monitoring cluster and configuring the cluster. EMR takes care of provisioning Hadoop clusters, running and terminating job flows, moving data between EC2 and S3, optimizing Hadoop.

A typical EMR process can be outlined as follows:

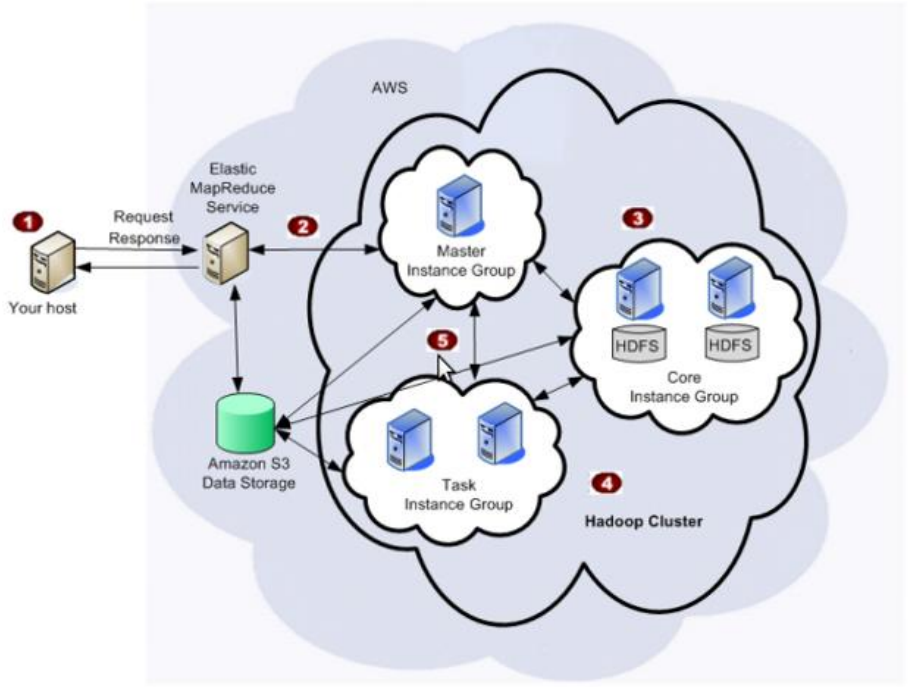


Figure: EMR JobFlow

Elastic MapReduce Process:

1) Upload to Amazon S3 the data you want to process, as well as the mapper and reducer executables that process the data, and then send a request to Elastic MapReduce to start a job flow.

2) Elastic MapReduce starts a Hadoop cluster, which loads any specified bootstrap actions and then runs Hadoop on each node.

3) Hadoop executes a job flow by downloading data from Amazon S3 to core and task nodes.

Alternatively, the data is loaded dynamically at run time by mapper tasks.

4) Hadoop processes the data and then uploads the results from the cluster to Amazon S3.

5) The job flow is completed and you retrieve the processed data from Amazon S3.

4.1 Core Nodes and Task Nodes:

Amazon EMR handles the details of creating a cloud infrastructure for running Hadoop. It defines the concept of instance groups which is a collection of EC2 instances that are analogous to the master and slave nodes in Hadoop. There are three instance groups: 1) Master instance group 2) Core Instance group 3) Slave Instance group.

Master Node: The Master node is responsible for distribution of MapReduce Tasks and the raw data to the core and task nodes, tracking the status of tasks and monitoring the health of instance groups. The core and the task nodes process the data, transfer the data back to Amazon S3, and provide status metadata to the master node. In a single node cluster the master node runs the NameNode, JobTracker, TaskTracker and DataNode daemons.

Core Node: A core node is an EC2 instance that runs Hadoop map and reduce tasks and stores data using the Hadoop Distributed File System (HDFS).

Task Node: Task Node refers to a node that contributes only compute resources for TaskTracker, and does not contribute any disk space to the cluster's storage pools.

A single node cluster of EMR an instance is simultaneously a master and core node. For clusters running on more than one node, one instance is the master node and the remaining are core or task nodes.

4.2 AWS CloudFormation

Amazon CloudFormation provides an easy way to manage and create a collection of AWS resources, provisioning and updating them in an orderly fashion. AWS CloudFormation provides templates to describe the required AWS resources and the associated dependencies required to run the application. A template is a JSON (JavaScript Object Notation) formatted text file that describes the AWS infrastructure needed to execute an application or service along with any inter-connection between them. The collection of resources that result from instantiating a template is known as a stack. A stack is created by supplying a template and any required parameters to the AWS CloudFormation service. Based on the template, the service determines what AWS resources need to be created and in what order. The order is determined by the dependencies specified between the resources declared in the template. We used AWS CloudFormation service because of following reasons:

a) Support for Wide Range of AWS Resources: AWS CloudFormation supports a wide variety of services that allows building a highly available and scalable infrastructure for application. The service was used to create an EC2 instance that was configured to collect twitter data.

b) Ease of use: CloudFormation provides an easy way to deploy and describe a collection of AWS resources. The user need not care about how the dependencies work or orders in which AWS services are provisioned.

c) Pricing: AWS CloudFormation is available at no additional charge. The user is billed only the normal rates for the AWS resources that are created by AWS CloudFormation and used by your application. For the project the only cost of using AWS CloudFormation was that of cost for provisioning a single EC2 instance.

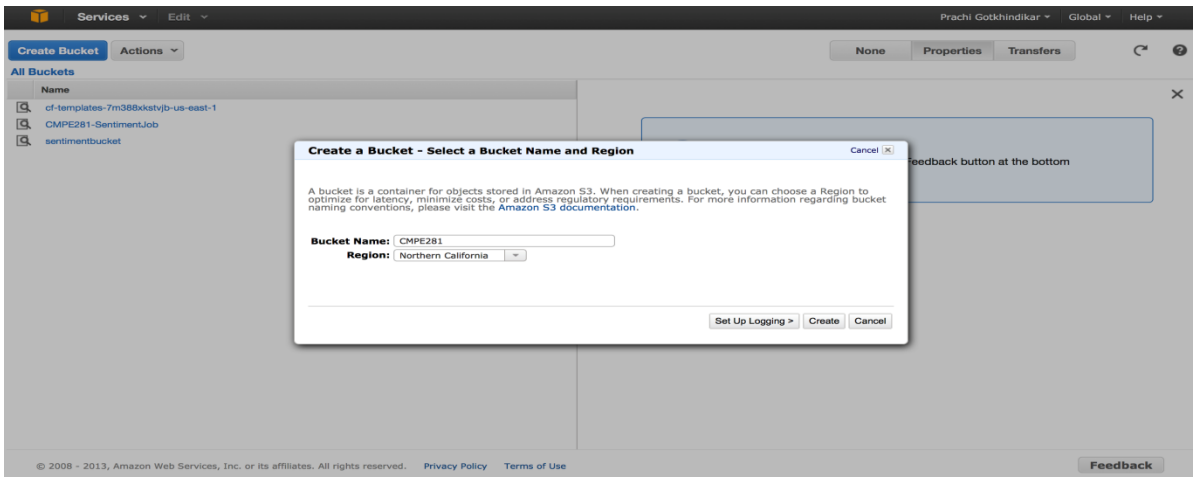
4.3 Implementing Sentiment Analysis on EMR:

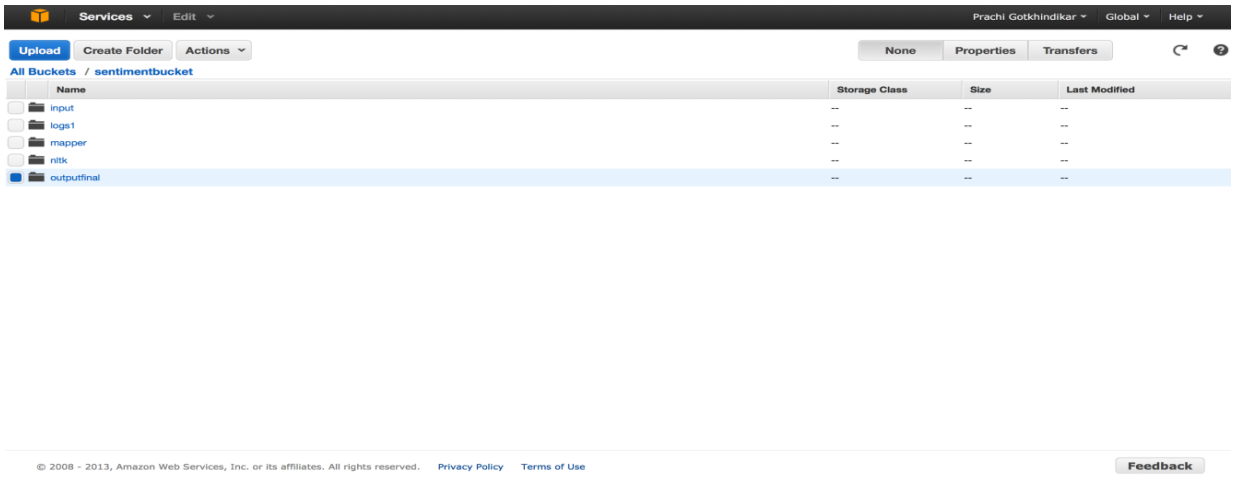
Amazon Simple Storage Service:

Amazon S3 is used for storing and retrieving any amount of data anytime. It is highly scalable, reliable secure and fast. Data in S3 is stored in buckets. Generally in AWS EMR, Amazon S3 buckets are used for storing mapper, reducer, input, and output and logs files if opted to store.

1. Steps for creating Amazon S3 bucket to store twitter data:

Before creating an EMR cluster, S3 bucket must be created to give the path for storing the logs file. While creating the bucket it asks for the name and the region in which we wish to store the bucket.

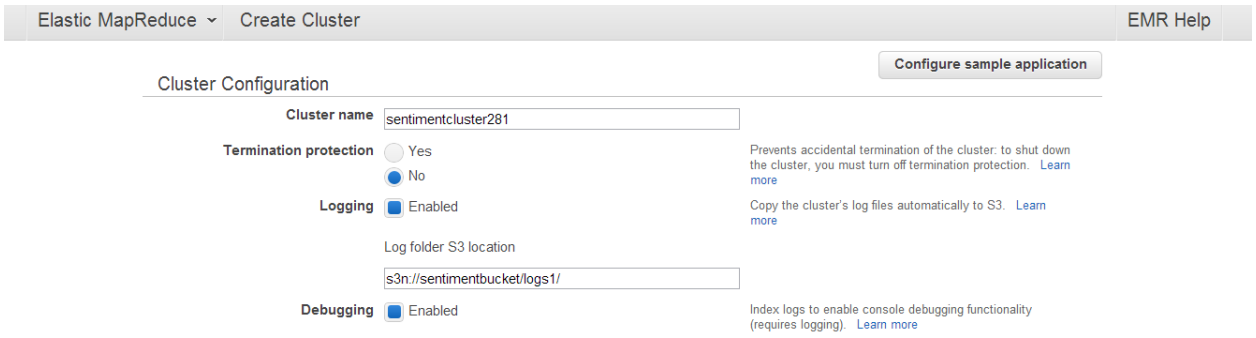




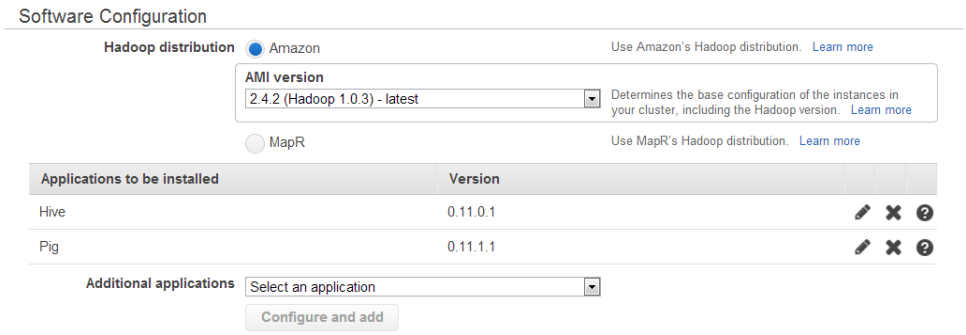
2. Steps to create an EMR cluster:

1) EMR cluster can be created using Amazon EMR console https://console.aws.amazon.com/elasticmapreduce/.

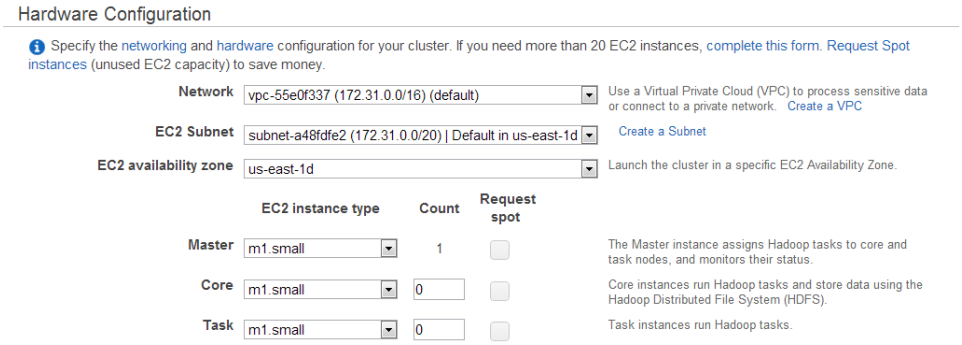
2) The cluster configuration section allows to provide the name of the cluster to be created , setting termination protection and enabling and disabling logging. Logging is useful for analyzing errors or inefficiencies in Amazon EMR steps applications. A path for S3 bucket needs to be provided to store the logs.



3) The Software configuration allows to select the Hadoop version as well as any other Hadoop applications such as Hive, Pig or HBase that need to be installed.



4) The hardware configuration allows to set the number of EC2 instances to be launched as master or slave nodes, the availability zone, network settings. The configuration for the master or core nodes can be selected from the 13 EC2 instance types provided by EMR. Since in our project we are setting up a single node cluster the number of master nodes is 1 and number of core nodes is 0. The instance type of master node is m1.small.

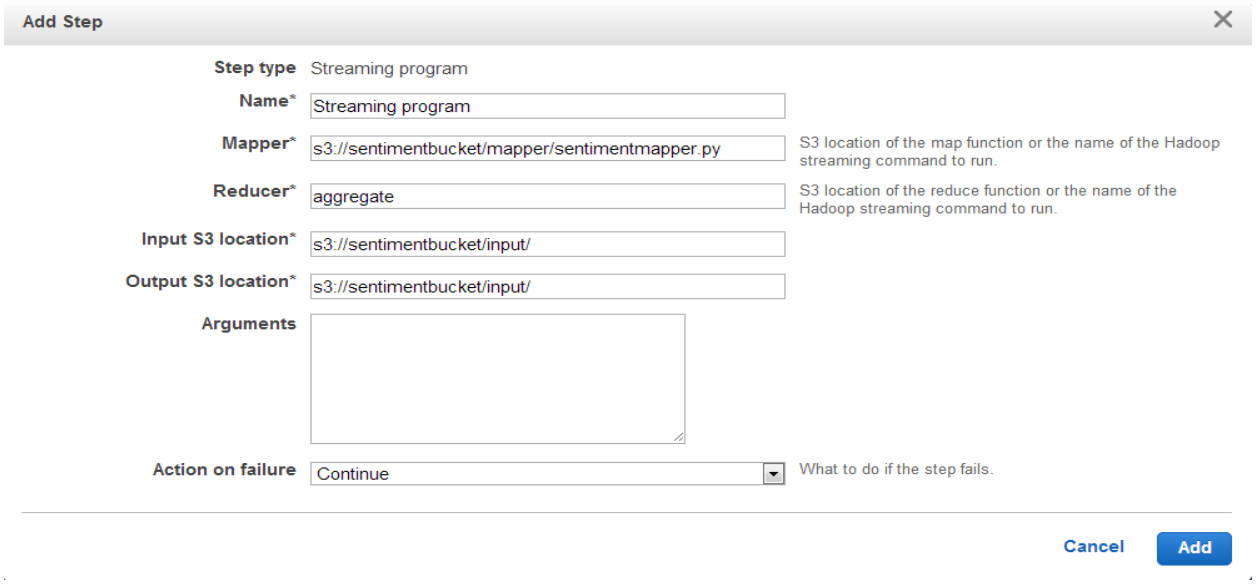


5) The security and access section allows selecting EC2 key pair and providing IAM settings.

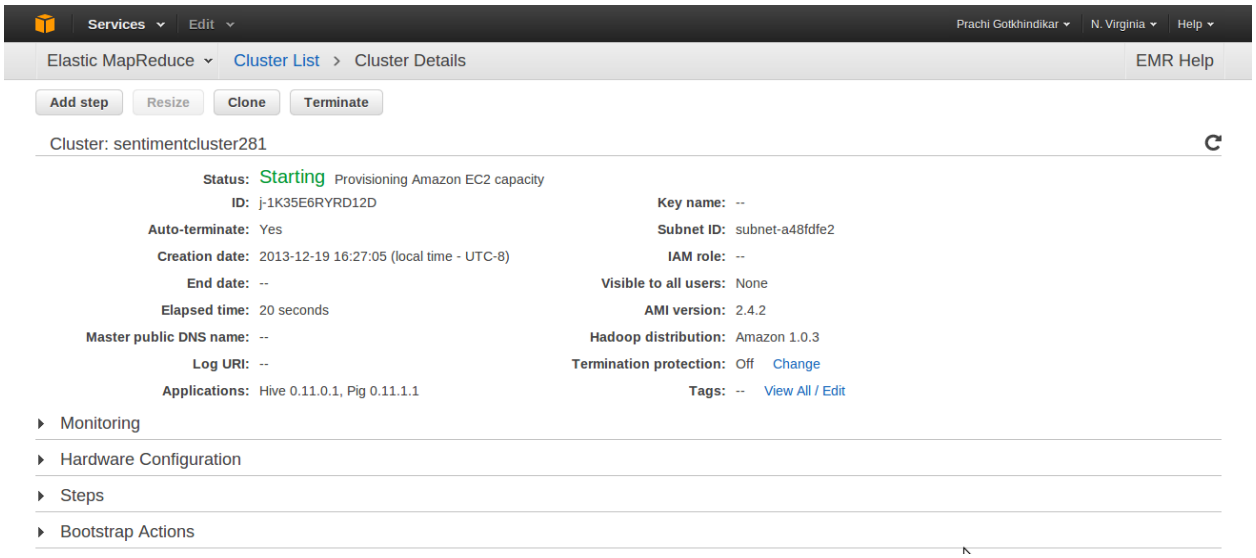
6) EMR also allows providing any bootstrap actions. A bootstrap action is a mechanism that lets you run a script on Elastic MapReduce cluster nodes before Hadoop starts. Bootstrap action scripts are stored in Amazon S3 and passed to Amazon Elastic MapReduce when creating a new job flow. Bootstrap action scripts are downloaded from Amazon S3 and executed on each node before the job flow is executed. In

our project we provide a custom action that configures and installs the Natural Language Toolkit on the cluster.

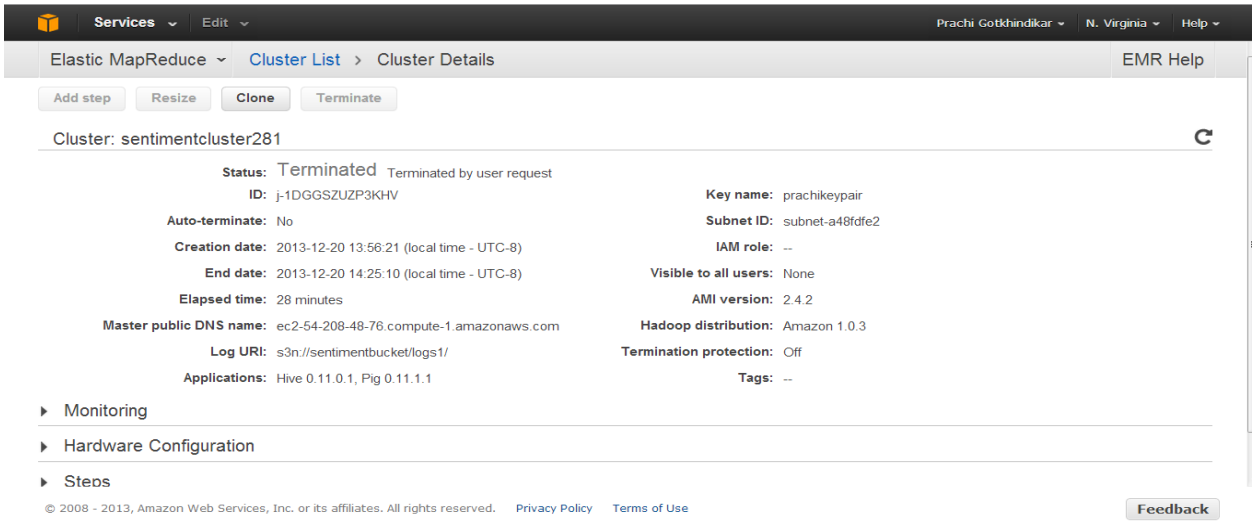
7) Typically, data processing involves performing a series of relatively simple operations on large amounts of data. In Amazon Elastic MapReduce, each operation is called a step and a sequence of steps is a job flow. The steps section allows providing the steps and configuring the job. The details about the mapper location, reducer, input data location and output location need to be provided.

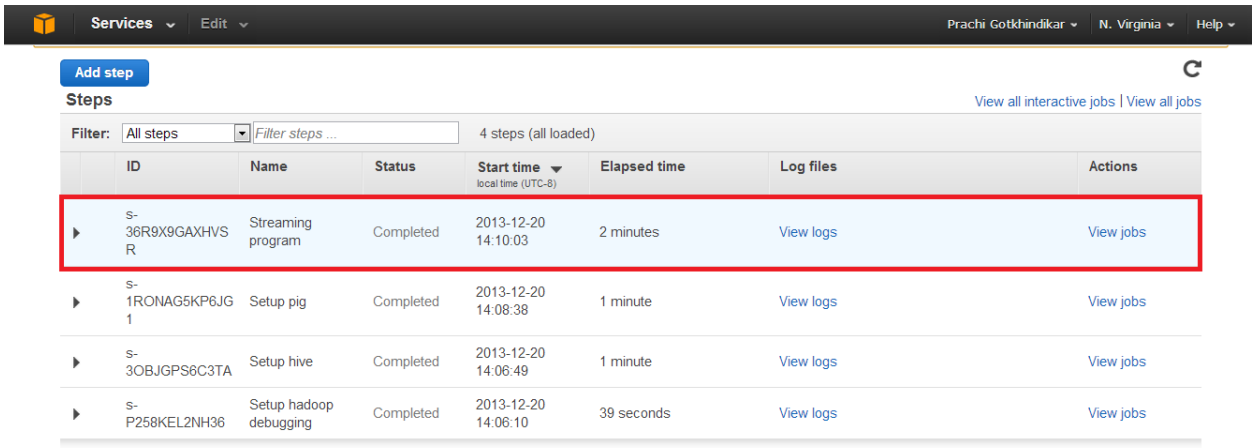


8) A summary of the new cluster created is provided.



9) Cluster terminated by user on success of job





3. Monitoring:

For monitoring the Map Reduce metrics, we use the Amazon monitoring web services. Generally there are many ways to monitor the cluster.

1. From the command line interface using the console we can detect the health of the services.
2. View the logs files that are stored in the S3 bucket. While creating the cluster we give a path to store the log files.
3. Using SSH we can connect to the master nodes and check the logs of the cluster.
4. Select the option of Ganglia monitoring application as a bootstrap action at the time of launching the cluster. This is a open source application which is used to monitor clusters. When this option

is enabled we can generate reports and check the performance of the cluster and the individual nodes of the cluster as well.

5. Cloud Watch is one of the Amazon web services; EMR reports the metrics of the cluster directly to Cloud Watch for free of cost. These metrics can be accessed directly in the EMR console or through Cloud Watch service.

Among all the ways available to check the metrics, the easier and convenient way is checking from the EMR console directly instead of going into the S3 and checking the log files. Here in the console the metrics are showed visually.

Monitoring using Cloud Watch on EMR console

All the metrics can be seen in the EMR console which can be tracked easily and can has an accessibility to enable alarm if any metric goes above or below the typical range and alert users with notifications. These metrics are updated for every 5 minutes.

In the Amazon EMR console, we have a Cluster details page in which there is a Monitoring tab. Under this tab we have Cluster Status, Map Reduce, IO, Node Status, and HBase. All these metrics are shown in graphical view with a time range option where we can select time range of last hour, last 2 hours, last 6 hours, last 12 hours, last 2 weeks etc. Some of the metrics tracked more frequently are regarding the progress of the cluster, check if any node runs out of storage, and check if the clusters went into idle state.

Metrics used for monitoring:

CATEGORY	METRIC	DESCRIPTION
Cluster Status	Is Idle	This indicates if the cluster is not in use but alive and also getting charged. If it shows 1 that means the no task is being run on the cluster but it is alive and 0 means the cluster is in use.
	Jobs Failed	Total number of jobs that failed.
	Jobs Running	Total number of jobs that are running.
	Map Tasks Running	The total number of mapper

Map/Reduce		tasks running for each job.
	Map Tasks Remaining	The total number of mapper tasks remaining for each job.
	Reduce Tasks Running	The total number of reducer tasks running for each job.
	Reduce Tasks Remaining	The total number of reducer tasks remaining for each job.
Node Status	Live Data Nodes	Percent of data nodes among all the data nodes available receiving job/work.
	Core Nodes Running	Number of core nodes running Unit: Count
	Live task Trackers	Percent of the total Task trackers that are functional
	Task Nodes Running	Number of task nodes running.
I/O	S3 Bytes Written	The number of bytes written to S3
	S3 Bytes Read	The number of bytes read from S3
	HDFS Utilization	Percent of HDFS storage being used currently from the total availability
	Total Load	Total number of data transfers done

5. Lessons learned from observations:

In analyzing and comparing Amazon EMR and Apache Hadoop Map Reduce running on local machine, we found that there are salient differences between them. On basis of those differences we learned that both Map Reduce paradigm are unique in nature and has its own ways of benefitting users.

A) Following are few observations that bolstered are claim of, "using Amazon EMR service is better than creating Hadoop structure in our own infrastructure."

Cost: Maximum cost required for us was $0.015 as we had used their standard Small (Default) on demand instance and $0.105 per GB for storage as the data was less than 1 TB whereas $0.0055 per request. Hence total cost required was less than $1.Infact you pay for what you use.

Whereas, cost required to implement Apache Hadoop on our local machine was comparatively too high than Amazon EMR considering factors like local resources. A detailed discussion about the cost savings due to the cloud is done in 'Economics of using Amazon EMR' section.

Complexity: Amazon EMR is very easy and user-friendly to build and run a Map Reduce job. We just had to concentrate on our Map reduce business logic (in our case sentiment analysis) and deploy over cloud.

In Apache Hadoop running on local machine had many difficulties of creating the distributed environment where as there are also other over heads like distributing software, managing and installation of Hadoop clusters, executing jobs, debugging, collection logs etc are too hard.

Accessibility: To perform a job on Amazon EMR we can access their machine from anywhere and from any computer. Whereas to run a map reduce job on Apache Hadoop framework on local machine, we have to run it on a specific machine on which it has been deployed.

Configuration: Amazon EMR is simple and easy to deploy it requires no additional knowledge to run the program.

In case Apache Hadoop user requires core knowledge of distributed system and installation of Hadoop distributed file system (HDFS) and should have basic knowledge of commands to run the program.

Hardware Optimization: We used only specific configuration of hardware that was required to run the program in Amazon EMR. In apache Hadoop in local machine, it was difficult to provide required configuration of hardware that a program requires.

Data provision: It is easy to provide data to perform map reduce on Amazon EMR, whereas it very inconvenient to get data in and out of HDFS.

Security in Amazon EMR:

Despite of the cost-effectiveness of the cloud many organizations hesitate to move to cloud due to the security concerns. Eliminating the cloud security concerns can lead to better ad option of cloud and increase its use. Increase in cloud usage leads to better utilization of resources, ease of deployment and maintenance.

a) Amazon takes care of all the security considerations. The master and slave nodes are launched in separate security groups. Only the master security group has a port open for communication with the service. The slaves operate in a separate security group, which only allows interaction with the master instance. Security groups can be reconfigured using standard EC2 tools and dashboards. By default security groups are not allowed access from external EC2 instances.
b) Amazon EMR also allows its customers to store the input data as well as the results into S3. This data is secured against unauthorized access. Unless it is specified by the customer who is uploading the data specifies otherwise, only that customer can access the data. Amazon EMR customers can also choose to send data to Amazon S3 using the HTTPS protocol for secure transmission. In addition, Amazon EMR always uses HTTPS to send data between Amazon S3 and Amazon EC2. For added security, customers may encrypt the input data before they upload it to Amazon they then need to add a decryption step to the beginning of their cluster when Amazon EMR fetches the data from Amazon S3.

B) Observations where we found that Apache Hadoop on local machine was preferred over Amazon EMR

Hadoop version: We have freedom t use whichever Hadoop frame work version that are currently released .Whereas in Amazon EMR case, we are forced to use only those Hadoop framework that are provided by Amazon.

Network latency: Apache Hadoop running on local machine has less network issues compared to Amazon EMR as Apache Hadoop are accessed personally where as for accessing Amazon EMR machine we have to depend upon the network connection between user and Amazon EMR instance.

Any facts and observation that reinforce the conclusion

- **Scalability**: It was very easy to increase or decrease the data whenever required with very high request rates at very low latency. Even after adding task and operation to the cloud, cloud resource and management configured the resource for task without much effort by adding more hardware without any drop in performance.
- **Availability**: Map reduces Service was always accessible with very ease.
- **Fault Tolerance**: By backing up the data it was easy to recover from a failure without loosing any data or updates from recently committed transactions.

6. The Economics of using Amazon EMR

One of the most cited benefits of moving to cloud is the cost savings that Cloud Computing brings to the organization. These cost savings are generated through distinct mechanisms which are declared as follows:

1. Total Cost of Ownership (TCO)

Leveraging the cloud for big data analysis significantly lowers down the total cost of ownership of technology. With cloud most of the costs are upfront and readily calculated. This is because of following reasons:

a) Amazon provides very transparent pricing based on several factors such as storage, RAM, bandwidth etc. Due to this it becomes easy for customer to gain certainty over pricing and readily calculate costs based on several different usage estimates. Amazon EMR pricing is per instance hour. This means even if you use the instance for part of an hour, Amazon charges for the complete hour. The pricing starts with 0.015 per hour for small instances. The price is a combination of EC2 base cost in addition to EMR service provided. The extra EMR price is usually 60% of the EC2 instance base price. Some typical combined prices are:

- Small: $0.015/hr
- Medium: $0.03/hr
- Large: $0.06/hr

b) Amazon also provides spot pricing wherein the price of the instance type depends upon the demand for that instance. Spot prices do offer significant cost savings for running huge jobs. The issue with the spot pricing is that if ever the spot price for an instance exceeds the price which the user bided, all the servers instantly disappear.

c) Amazon EMR provides a facility of integrating S3 for storing the input data as well as the results. Besides S3, EMR can also leverage other data stores such as DynamoDB, Amazon RedShift etc. The price for each storage type is then added to the total price for running the job.

Comparing this with on premise technology, there are several hidden costs which are not taken account during in-house cost calculation. They are as follows:

- Often the direct costs of running the server includes power, floor space, storage and IT operations.
- The indirect costs of running a server include network and storage infrastructure and other IT operations to manage general infrastructure.
- To manage in house infrastructure a dedicated Ops team is required for managing the infrastructure. Finding skilled professional is an additional overhead cost of owning a server.

2. Shift to Operational Expenses from Capital Expenses:

Amazon EMR offers Hadoop -as- a service. This model provides businesses greater agility by giving instant access to Hadoop clusters with pay -per -use consumption model. One of the important features of cloud computing is that it is a recurring expenditure model. It provides the flexibility of terminating the costs at will whenever the service is not required. The elastic nature of Amazon EMR makes it possible to obtain Hadoop clusters on demand. It is not necessary to provision excess capacity for future requirement or peak periods. Comparing this with traditional IT which is more capital intensive. It requires hardware to be brought, software to be licensed. Investing in buying hardware implies a full commitment towards the servers or software being purchased. That means regardless of whether it is being utilized, ongoing costs need to be borne.

3. Opportunity Cost:

Opportunity cost is an economics term that is concerned with the cost related to choices that must be forgone in order to pursue a certain action. Ex. the opportunity cost of going to college is money that one would have earned if one works instead.

At its core the investment in cloud is the opportunity cost of investing the capital in business, rather than IT systems. Amazon EMR allows focusing on Hadoop-based analysis rather than worrying about the underlying infrastructure. This allows organizations to concentrate on improving their efficiency rather than wasting time in worrying about non-core activities. The automatic scalable nature of the Amazon EMR service makes it possible for the user to eliminate the cost involved in upfront provisioning of hardware for peak periods, and additional operational staff.

7. Comparison on basis of observations.

	Apache Hadoop on local machine	**Amazon EMR**
Programming model Used	Hadoop Map Reduce	Hadoop Map Reduce
Data Handled	HDFS(Hadoop Distributed File System)	Amazon S3
Dynamic Scalability	No	Yes
Configuration	Manual	Auto(by Amazon)
Time to run task	Comparatively more than Amazon EMR(in our case approximately 4 mins)	Less than Apache Hadoop on local machine(in our case approximately 2 mins)
Flexibility	No	Yes
Data transferred internally	HTTP	TCP

8. Team member's Contribution

Sr. No.	Name	Contribution
1	Abhiram Uppaluri	1. Research on Hadoop HDFS. 2. Map Reduce Algorithm.
2	Prachi Gotkhindikar	1. Research on Amazon's EMR service 2. Configuration of EMR service and ran map reduce job 3. Economics of using Amazon EMR
3	Srividya Reddy Thota	1. Research tweepy library for collecting twitter data using python. 2. Configuration of EMR service. 3. Monitoring of EMR using Amazon's CloudWatch service.
4	Sumant Murke	1. Mapper and Reducer code. 2. Collected monitoring metrics for both local HDFS and EMR. 3. Comparison on basic of observations for both approaches.
5	Vaibhav Bhor	1. Java, Python, and Hadoop installation and configuration on local Linux machine 2. Mapper and Reducer code. 3. Running Map Reduce jobs on the local HDFS and collecting results.

9. Code listing

Code for Collecting Twitter data (collector.py):

```
# Tweet collector via tweepy and user-supplied search terms,
# to be used with CloudFormation template
# based on http://badhessian.org/2012/10/collecting-real-time-twitter-data-with-the-streaming-api/
# with modifications by http://github.com/marciw

from twaiter import TWaiter
import tweepy, sys, twitterparams

# authentication params (supplied via cfn)
consumer_key = twitterparams.OAuthConsKey
consumer_secret = twitterparams.OAuthConsSecret
access_token = twitterparams.OAuthToken
access_token_secret = twitterparams.OAuthTokenSecret

# OAuth via tweepy
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)
api = tweepy.API(auth)


def main(term):
    if len(term) == 2:
        search = [term[1]]
    else:
        print "Please provide only one search term. Multi-word terms should be enclosed in quotation marks.
\nExample:  python collector.py \"kindle fire\" "
        sys.exit()

    collection = 'tweets'
    waiter = TWaiter(api, collection)
    stream = tweepy.Stream(auth, waiter)

    print "Collecting tweets. Please wait."

    try:
        stream.filter(track=search)
    except Exception, e:
        print "An error occurred. No tweets collected.", e
        stream.disconnect()


if __name__ == '__main__':
    main(sys.argv)
```

twaiter.py

```
from tweepy import StreamListener
```

```
import json, time, sys
class TWaiter(StreamListener):
     def __init__(self, api = None, label = 'default_collection'):
       self.api = api or API()
       self.counter = 0
       self.label = label
       self.output  = open(label + '.' + time.strftime('%b%d-%H%M') + '.txt', 'w')
       self.deleted  = open('deleted_tweets.txt', 'a')

   def on_data(self, data):
       # The presence of 'in_reply_to_status' indicates a "normal" tweet.
       # The presence of 'delete' indicates a tweet that was deleted after posting.
       if  'in_reply_to_status' in data:
          self.on_status(data)
       elif 'delete' in data:
          delete = json.loads(data)['delete']['status']
          if self.on_delete(delete['id'], delete['user_id']) is False:
             return False


   def on_status(self, status):
       # Get only the text of the tweet and its ID.
       text = str(json.dumps(json.loads(status)['text']))
       id = str(json.dumps(json.loads(status)['id_str']))
       self.output.write("id:" + " " + id[1:-1] + ", " + "text:" + " " + text[1:-1] + "\n")

       self.counter += 1

       # For tutorial purposes, only 500 tweets are collected.
       # Increase this number to get bigger data!
       if self.counter >= 50000:
          self.output.close()
          print "Finished collecting tweets."
          sys.exit()
       return

   def on_delete(self, status_id, user_id):
       self.deleted.write(str(status_id) + "\n")
       return

   def on_error(self, status_code):
       sys.stderr.write('Error: ' + str(status_code) + "\n")
       return False
```

Mapper File :

```
#!/usr/bin/python

# Simple python script that loads a classifier from file, then uses that to generate the sentiment of each line of input.
Designed for    use with EMR as a mapper, but can be used on the command line as well.

import cPickle as pickle

import nltk.classify.util

from nltk.classify import NaiveBayesClassifier
```

```
from nltk.tokenize import word_tokenize
import sys
sys.stderr.write("started mapper\n");
def word_feats(words):
    return dict([(word, True) for word in words])
def subj(subjLine):
    subjgen = subjLine.lower()
    # Replace term1 with your subject term
    subj1 = "obama"
    if subjgen.find(subj1) != -1:
        subject = subj1
        return subject
    else:
        subject = "No match"
        return subject
def main(argv):
    classifier = pickle.load(open("classifier.p", "rb"))
    for line in sys.stdin:
        tolk_posset = word_tokenize(line.rstrip())
        d = word_feats(tolk_posset)
        subjectFull = subj(line)
        if subjectFull == "No match":
            print "LongValueSum:" + " " + subjectFull + ": " + "\t" + "1"
        else:
          print "LongValueSum:" + " " + subjectFull + ": " + classifier.classify(d) + "\t" + "1"
if __name__ == "__main__":
    main(sys.argv)
```

10. References

- Baum,D (2012, December) . An Engine For Big Data. Java Magazine,25-29
- HDFS & MapReduce,Cloudera.com.Retrieved November 7,2013,from http://www.cloudera.com/content/cloudera/en/products/cdh/hdfs-and-mapreduce.html
- Amazon Elastic MapReduce (n.d.). Amazon Elastic MapReduce (Amazon EMR). *Elastic Map Reduce*. Retrieved December 18, 2013 from http://aws.amazon.com/elasticmapreduce.
- Apache Hadoop (2013, October 15). Hadoop. *Apache*. Retrieved November 7, 2013 from http://hadoop.apache.org.
- Apache Hadoop(2013, August 11). Apache Hadoop. *Wikipedia.* Retrieved November 7, 2013, from http://en.wikipedia.org/wiki/Apache_Hadoop.
- Apache Hadoop(n.d.). MapReduce Tutorial. *Apache.* Retrieved November 7, 2013, from http://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html#Purpose
- Realtime Sentiment Analysis(n.d.). Realtime Sentiment Analysis Application Using Hadoop and HBase. *Slideshare*. Retrieved November 7, 2013, from www.slideshare.net/Hadoop_Summit/realtime-sentiment-analysis-app-using-hadoop-and- h-base
- Amazon Elastic MapReduce (2009, March 31). Amazon Elastic MapReduce Developer Guide. Retrieved December 19, 2013 from http://www.technicalwritingprofessional.com/images/emr-dg.pdf
- Amazon Web Services (n.d.) Auto Scaling. *Auto Scaling.* Retrieved December 19, 2013 from http://aws.amazon.com/autoscaling/
- Code Fusion (2013, October 21). Setup newest Hadoop 2.2.0 on Ubuntu. *Codes Fusion.* Retrieved December 17, 2013 http://codesfusion.blogspot.com/2013/10/setup-hadoop-2x-220-on-ubuntu.html
- Amazon Elastic MapReduce (2009, March 31). Create the Amazon S3 Bucket. *Amazon Elastic MapReduce.* Retrieved December 20, 2013 http://docs.aws.amazon.com/ElasticMapReduce/latest/DeveloperGuide/emr-get-started-count-words-step-2.html
- Amazon Elastic MapReduce (2009, March 31). Monitor the Cluster. *Amazon Elastic MapReduce.* Retrieved December 20, 2013 http://docs.aws.amazon.com/ElasticMapReduce/latest/DeveloperGuide/emr-get-started-count-words-step-6.html

- Amazon Elastic MapReduce (2009, March 31). Monitor metrics with CloudWatch. *Amazon Elastic MapReduce*. Retrieved December 20, 2013 http://docs.aws.amazon.com/ElasticMapReduce/latest/DeveloperGuide/UsingEMR_ViewingMetrics.html
- Amazon Elastic MapReduce (2009, March 31). What is Amazon CloudWatch. *Amazon Elastic MapReduce*. Retrieved December 20, 2013 http://docs.aws.amazon.com/AmazonCloudWatch/latest/DeveloperGuide/WhatIsCloudWatch.html