Crisis Analysis from Social Media

Using Hadoop MapReduce

Gaurav Shad
CIDSE, ASU
gshad@asu.edu

Abhishek Vellanki
CIDSE, ASU
avellank@asu.edu

Nitesh Gupta
CIDSE, ASU
ngupta40@asu.edu

***Abstract*—Social Media contains wide variety of data that can be analyzed to get some meaningful information which can be used for predictions, advertisements, marketing etc. The amount of data that needs to be analyzed is so big that its processing is inefficient on a single machine using limited resources. This problem can be addressed by merging cloud computing and Hadoop to do distributed computing of data. The goal of this project is to develop an application that analyzes twitter data related to crisis (including natural disasters, terrorist attacks, etc) and represent in an interactive format the areas that are prone to some crisis. The scope of this project includes setting up Hadoop in a private cloud cluster and implementing MapReduce to process this data. The main tasks include designing of Mapper and Reducer that will help generate some quality output.**

Keywords—Hadoop, MapReduce, FLUME, Twitter, Data Visualization, Big Insights, Spark, Google Fusion Table,Google Maps, Google Drive

I. Introduction

Social media connects people around the globe and hence turned itself into a platform where people share every event happening in their lives. Twitter's ease and speed of communication has made it the heart of social networking with millions of tweets every day. Because of its popularity, it has led to the development of applications and research in various domains [1]. Disaster detection is one such domain where researchers have already worked in the past and successfully predicted their occurrences."The 2014 earthquake in Napa was detected by USGS in 29 seconds using Twitter data, likely due to the tech savvy population that dominates the area" [2]. In this project we will analyze previous tweets for chosen time period (like 5 years) relating to crisis and return end user, graphical information based on the search criteria (location or crisis or both).

The problems addressed here are computing time and scalability. By processing data in a multimode cluster using MapReduce, we are enabling parallel processing which decreases the computing time and increases the scalability.

The big data that needs to be analyzed can be unstructured which traditional databases cannot handle, query and analyze. There comes a need for a system that can handle and analyze raw unstructured data efficiently. Since this big data is increasing exponentially, if not analyzed, this data remains meaningless. To extract some information from it, requires this data to be processed.

Technologies used:

Apache Hadoop – It's a framework for processing large data sets.

MapReduce – It's a programming paradigm which performs highly parallelized processing of large data sets over multiple clusters of nodes.

Apache FLUME – This is a service that helps in extracting data with the use of Twitter API and storing it into HDFS.

Data Visualization – ~~IBM Insights or~~ Google Maps API~~ or KDE or Apache Spark~~ will be used on the output set to visualize it.

Google Fusion Tables is used for visualization over the other mentioned tools because:

1. System requirements for the installation is very high for BigInsights;12GB RAM, 100GB Hard Disk etc.
2. BigInsights installs its own Hadoop on top of which it will works, our specific task was related to **BigSheets** component which was available separately in the older version but now is integrated as BigInsights complete package.
3. Apache Spark on the other hand has very limited options, it runs its own command to extract twitter data and the virtualization is done on that data.

Google fusion Table is, an experimental project, a web service provided by Google for data management. Fusion tables can be used for gathering, visualizing and sharing data tables.

Google Maps Java Script APIs are used to plot the location on map. Additionally, Fusion tables allows us to create different visualization for our purpose we have visualized data as Bar Graphs, Heatmaps, Map Markings and Table View.

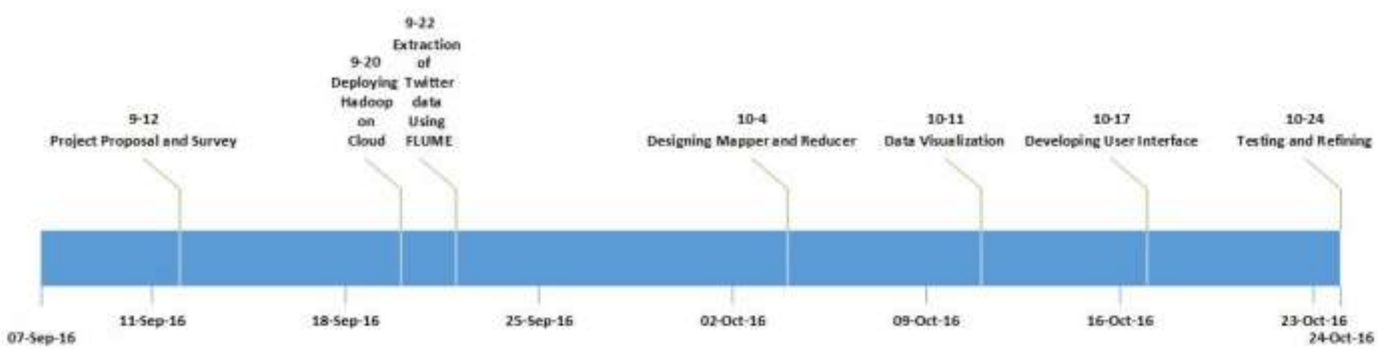
Expected Outcomes:

By working on this project, we expect to develop a data analytics application that will help visualize big data. The user interface will be developed to help end user in finding crisis prone areas in particular region.

The team comprises of three students:

1. Gaurav Shad
2. Abhishek Vellanki
3. Nitesh Gupta

We will work together in all the tasks (Tasks 1 -7) so that all team members learn everything involved in the project. Timelines for the project are as follows:



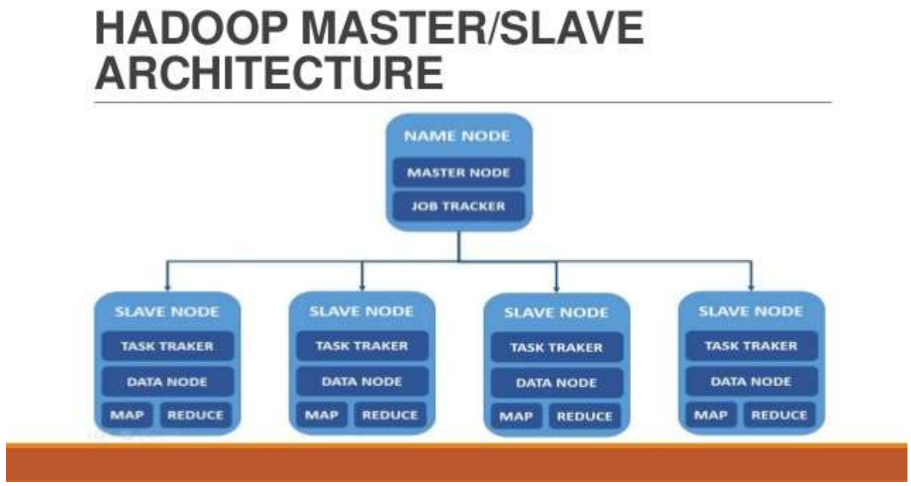
1. Timeline

II. SYSTEM MODELS

A. System Model

The model consists of setting up a Hadoop Multi node cluster in a cloud. All the nodes, including the Master Node will reside in the cloud environment. The files need to be configured so as to interact and perform processing over different data nodes present in the cluster.

HDFS Architecture:



2. HDFS Architecture

Master Node:

- It is the master of the system which maintains and manages data on slave nodes.

Slave Node:

- These nodes are deployed for actual storage and data processing.

Job Tracker:

- It resides in the master node which determines the execution plan and manages all the tasks.

Task Tracker:

- It resides in all the slave nodes which keeps track on the task and keeps job tracker updated.

B. Software

1. Apache Hadoop

   It's an open source framework for storage and processing of large data sets. It consists of four modules:

   - Hadoop Common
   - Hadoop Distributed File System (HDFS)
   - Hadoop YARN
   - Hadoop Map Reduce

2. Open Stack (modified)

   It's an open source cloud computing platform for public and private clouds. It lets user deploy virtual machines and other instances that handle different tasks for managing the cloud environment.

   Open Stack is not required for our project since the major task is performed by the Hadoop Yarn (Resource Manager) i.e. parallel processing. Initially we thought that we might require open stack capabilities to dynamically control the nodes (name and data) which later on we found that Hadoop YARN can take care of.

3. Apache FLUME

   It helps in efficiently collecting and moving large amounts of streaming data directly into HDFS. In our project, it will build a connection by using Twitter API to collect data.

4. Twitter API

   It helps to extract tweets based on a specific query. It can restrict the number of tweets to the given language, location and number of tweets allowed per page.

5. Data Visualization Tool

   We will use ~~Big Insights or Spark~~ to visualize the data that is received from MapReduce.

   We are using Google Fusion Tables to visualize the data.

   Following Visualizations are the options given to users:

   1. Bar Graph
   2. Heat Maps
   3. Location Map
   4. Table View

III. Project Description

The project concentrates on the data set retrieved from Twitter based on time filtering (like last 5 years) and process this data to represent end user how prone a region is to a crisis. Region can be a city, state or country.Performing crisis analysis on tweets can be trickier than normal text or paragraph because tweets are short messages which contain hash tags, slangs etc.

The main task of this implementation is to automatically filter which tweets that contain the crisis information required in this project. The MapReduce will use certain conditions to perform this task. In a research done by USGS National Earthquake Information Center (NEIC), they built similar kind of analytics application by filtering tweets on the following conditions [2]:

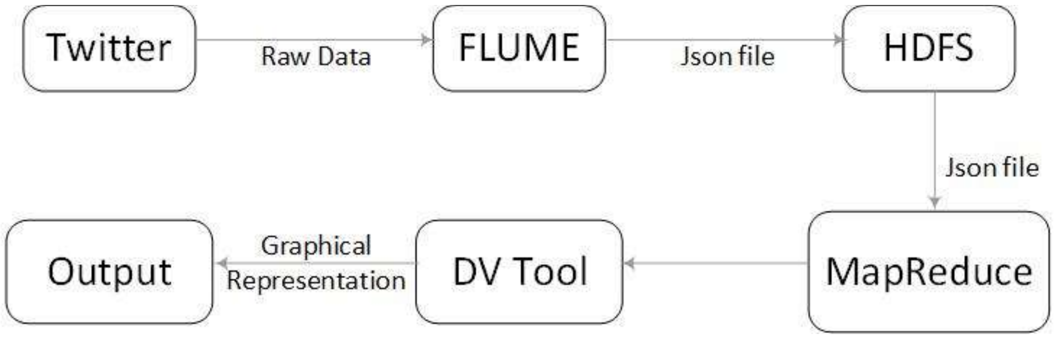
~~1. Remove the tweets containing more than seven words~~

~~2. Remove the tweets that contains any links~~

1. The tweet needs to contain less than 10 words

   (People caught up in a crisis don't write long tweets).
2. The tweet should not contain a link to any website
   (Someone might just be sharing an article).
3. It should not be a retweet
   We only need the original tweets to get accurate location of the crisis).
4. Tweets containing some keywords need to be ignored, for example, prayer, RIP, Magnitude etc.
   (People tweeting for others with the keywords mentioned. Magnitude won't be known by the people in crisis).
5. To get the location there are two ways if the geo feature is enabled, we will directly get the latitude and longitude or else, we will store the full name from the place parameter.

We will use the similar conditions to perform basic filtering and generalize it for all types of crises. After the basic filtering, we will apply more conditions to remove the tweets that contain crisis information of some other location. Finally, the processing will extract geo locations and data visualization tool will help build a graphical representation of it. And the end user will use the user interface of this application.



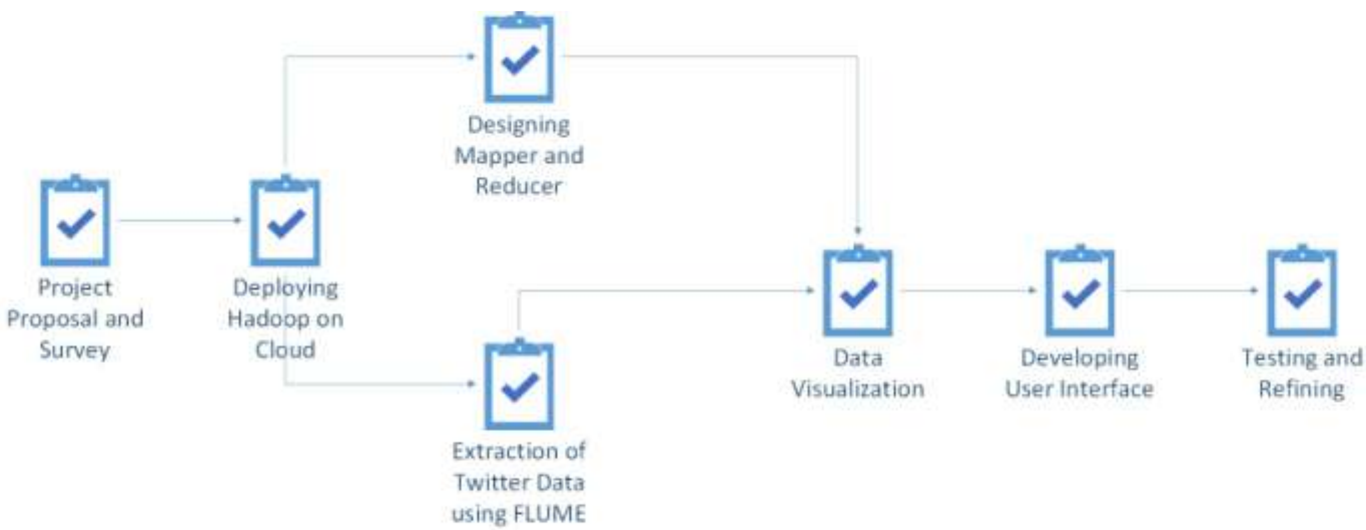
3. Workflow Diagram

A. Project Overview

The project is implemented in 6 tasks. **First task** is to install and configure Hadoop in a private cloud. **Second task** is to retrieve data from Twitter using Apache FLUME and sink this data to HDFS. **Third task** is to design Mapper and Reducer to process the data. **Fourth task** is to visualize data generated in the last task. **Fifth task** is to develop a user interface for better communication. **Sixth task** is to test the application and refine it.

Mid-Term goal for this project is to complete till task 3, designing Mapper and Reducer. We will use these Mapper and Reducer to analyze data and return outputin a readable format.

Final-Term goal for this project is to complete all the tasks that involve generating a final graphical representation of the output and a user interface.



4. Task Dependencies

B. Task 1 : Deploying Hadoop on cloud

This task involves setting up a working environment for this project. It involves understanding Hadoop and its configuration files and making a multi node cluster on a private cloud which is the main requirement to process the data.

C. Task 2: Extracting Twitter data using FLUME

This task involves collecting tweets using FLUME. FLUME provides this service by establishing a connection with the help of twitter API for collecting streaming data. This data will further be used by Mapper and Reducer.

D. Task 3: Designing Mapper And Reducer

This task involves designing of Mapper and Reducer which will filter tweets based on certain conditions similar to the ones used by NEIC (National Earthquake Information centre) to improve the efficiency is identification of crisis. This part will also filter out the tweets that are generated from a location that is different from the crisis location. This will give only those tweets that further needs to be processed for determining the type of crisis and its geo location.

E. Task 4: Data Visualization

This task involves using the data generated in the previous step to be represented on a graph, or a heat map by using data visualization tools like ~~Big Insights or Spark~~. The Reducer will generate the data in such a format that can be directly given as an input to the above-mentioned tools.

Google Fusion Table is used for visualization purposes.

F. Task 5: Developing User Interface

This task involves creation of user interface which will enable the user to just select a region or a crisis and see the end results in graphical format. If the user enters a region, it will display details for all crises that happened in that region. And if the user enters a crisis, it will return regions.

G. Task 6: Testing and Refine

This task involves testing of the developed application for performance and efficiency testing as well as any potential bugs.

H. Project Task Allocation

	Gaurav Shad	**Nitesh Gupta**	**Abhishek Vellanki**
Deploying Hadoop on Cloud	34%	33%	33%
Extraction of Twitter Data using FLUME	33%	33%	34%
Designing Mapper and Reducer	33%	34%	33%
Data Visualization	33%	33%	34%
Developing User Interface	33%	34%	33%
Testing and Refining	34%	33%	33%

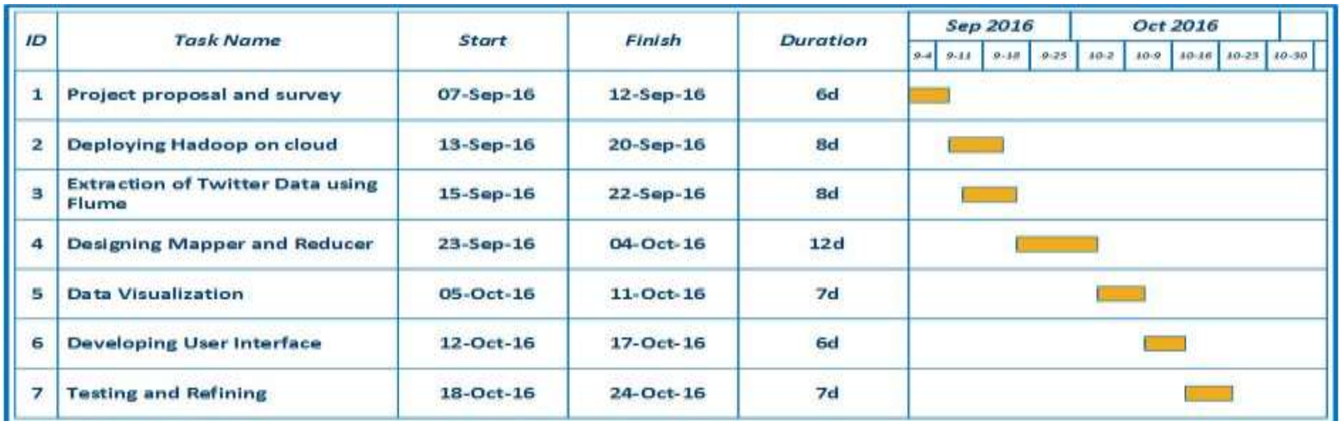
5. Task Allocation

I. Deliverables

Outcomes of the project

- Working Hadoop Map Reduce Environment in a private cloud.
- User Interface designing to receive input from user.
- An application of big data analysis using twitter data.
- Data Visualization on processed data

J. Project Timeline



ID	Task Name	Start	Finish	Duration
1	Project proposal and survey	07-Sep-16	12-Sep-16	6d
2	Deploying Hadoop on cloud	13-Sep-16	20-Sep-16	8d
3	Extraction of Twitter Data using Flume	15-Sep-16	22-Sep-16	8d
4	Designing Mapper and Reducer	23-Sep-16	04-Oct-16	12d
5	Data Visualization	05-Oct-16	11-Oct-16	7d
6	Developing User Interface	12-Oct-16	17-Oct-16	6d
7	Testing and Refining	18-Oct-16	24-Oct-16	7d

6. Gantt Chart

PROJECT PROGRESS

Project Status: Till Midterm

GIT Link: http://gitlab.thothlab.org/gshad/StormRiders-Crisis-Analysis.git

Project has progressed as planned initially during the project proposal. Tasks 1,2 and 3 have been completed successfully.

Task Name	Status
Project Proposal and Survey	Complete
Deploying Hadoop on Cloud	Complete
Extraction of Twitter Data using Flume	Complete
Designing Mapper And Reducer	Complete(Add-On Required)
Data Visualization	Yet to start
Developing User Interface	Yet to start
Testing And Refining	Yet to start

Tasks Completed
Task 1:
Deploying Hadoop on Cloud
In this task we setup Hadoop on private cloud in Thoth Lab and got it working. For this we understood how Hadoop works and then configured the XML files accordingly and setup the HDFS.

Task 2:
Extraction of Twitter data using FLUME
In this task we collected Twitter data using Twitter APIs. Twitter has 2 types of API to collect their data. The Streaming API gives us tweets in real-time. As the tweets are posted in twitter they get pushed into the HDFS. Apache FLUME has been used to integrate with this streaming API. After FLUME was installed, its configuration files were edited accordingly. It would push twitter data into the HDFS based on the parameters mentioned in its configuration files.

Since our application also requires historical data, we used a REST API by twitter to get some old twitter data as proof of concept. We used a Python Script to connect to the REST API which will search data from last 7 days.

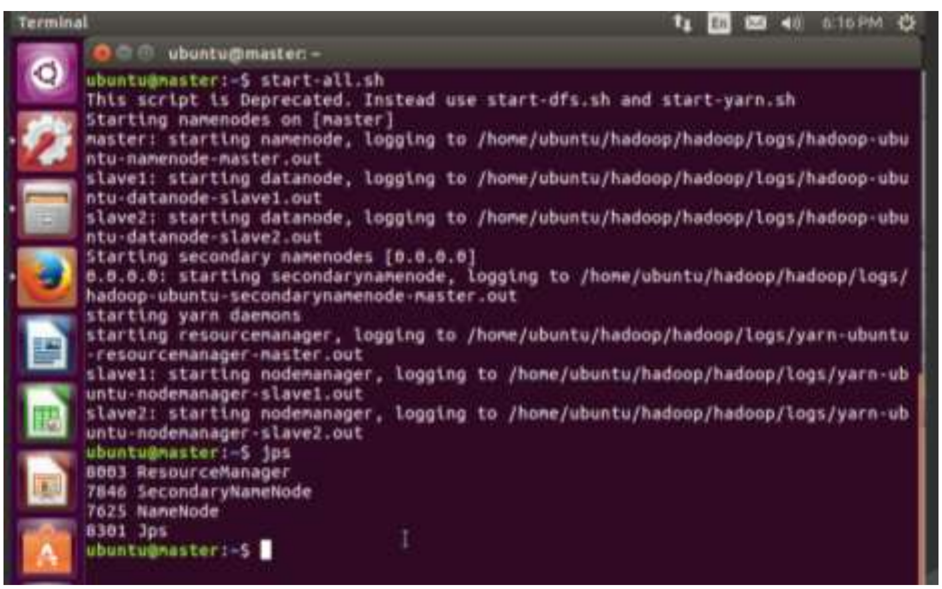
Task 3:
Designing Mapper and Reducer

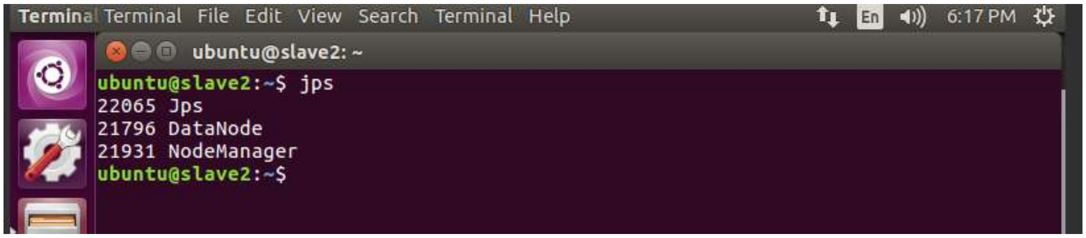
We designed a MapReduce which is the important part of Hadoop framework to process the tweets. The tweets will be filtered out in the Mapper class. The Mapper class will give output as key value pairs which then be combined and used by Reducer class to generate final output.

Tasks detailed walkthrough
Deploying Hadoop on Cloud:

- Rename the nodes as master, slave1 and slave 2
- Edit the etc/hosts file on all nodes to mention ip address of each node with hostname
- setup **ssh**: generate **rsa keys** on master and transfer it to other data nodes to configure key based login in cluster
- Install Hadoop-2.7.3 on master node
  (Hadoop can be downloaded from Apache Software Foundation as tar file)
- Set up Hadoop environment variables in **bashrc file**
- Edit Hadoop configuration files in /etc/Hadoop

  - ➢ Edit **core-site.xml** to enter details of master node used for Hadoop instance
  - ➢ Edit **hdfs-site.xml** to enter details about replication, namenode and datanode
  - ➢ Edit **yarn-site.xml** to enter details about resource manager and node manager to configure yarn into Hadoop
  - ➢ Edit **mapred-site.xml** to enter details about job tracker and mapreduce framework
  - ➢ Edit **Hadoop-env.sh** to enter environment variables
- Transfer Hadoop files from master to all slave nodes using **scp**
- Create masters and slaves file in etc/Hadoop to enter hostnames
- Format namenode on master
- Start all the services and Hadoop is all set to go





Extraction of Twitter data using FLUME

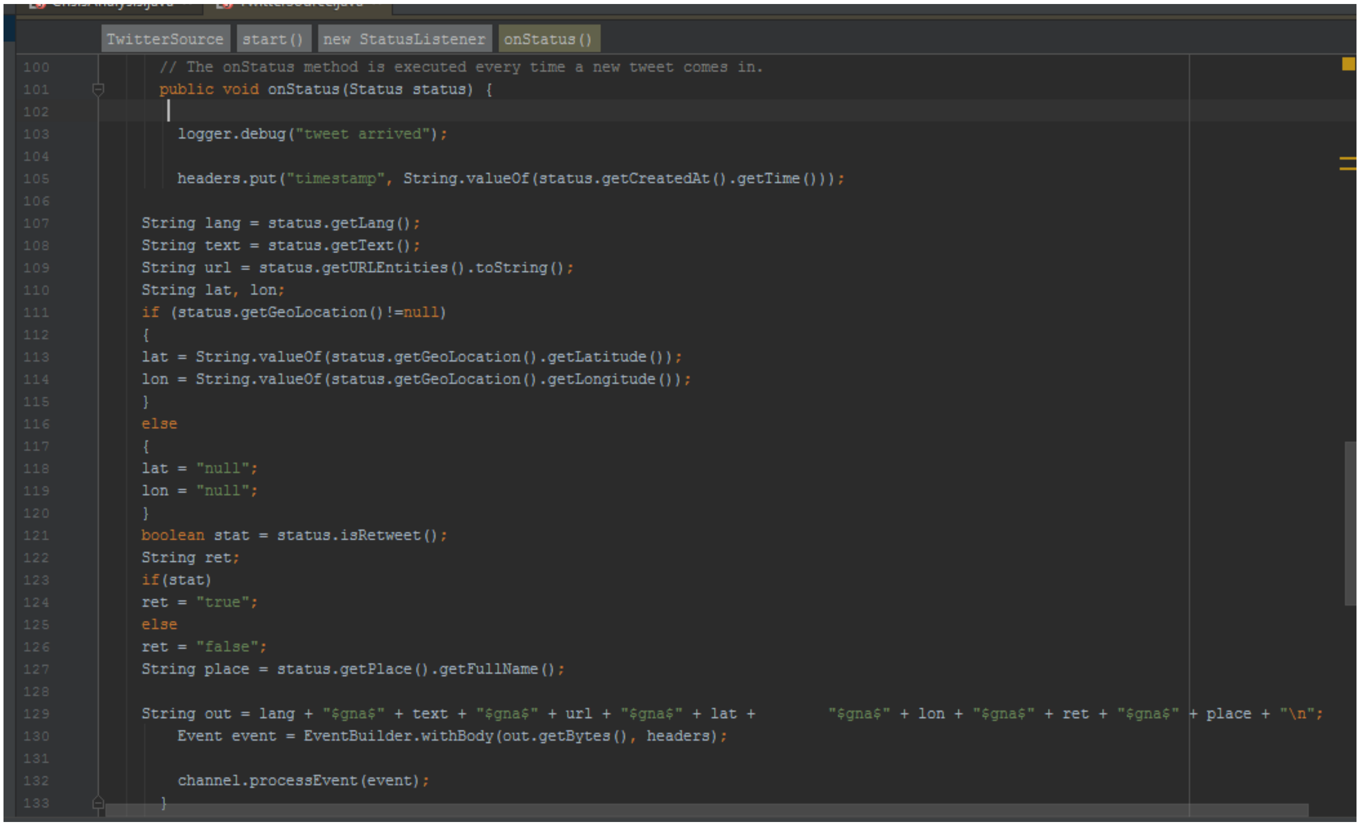
FLUME was setup on our Master node, it pushed the tweets into the hdfs path we mention in its configuration files. FLUME connects to the Steaming API of twitter and pushed tweets in real-time as they are published on Twitter.

Steps to setup FLUME:

- Install Apache FLUME
  FLUME can be downloaded from the Apache website as a tar file
- Create an app on **apps.twitter.com** and get the Authentication tokens. There are 4 tokens Consumer key, Consumer Secret, Access Token, Access Secret Token
- Extract the FLUME tar file
- Edit flume.env.sh.template file and mention the Java path and save as **flume.env.sh**
- Create **twitter.conf** file in the conf folder, FLUME works as per this configuration file
  And mention details about the source, channel and sink
- Place **FULME-sources-1.0SNAPSHOT.jar**, **twitter-4j-core-4.0.5.jar**, twitter4j-stream-4.0.5.jar, twitter4j-media-support-4.0.5.jar in the lib folder inside the flume directory
- Run FLUME
- Tweets generated this way will be stored in the hdfs directory as mentioned in the twitter.conf

We require specific data from the tweets coming through flume i.e. language, text of tweet, url in tweet (if it exist), latitude, longitude, retweet status and place (will be used in case if there is no geo location).

To get this specific data we require to modify the **TwitterSource.java**, compile it and generate TwitterSource.class and TwitterClass$1.class. This file is uploaded in gitlab along with the TwitterSource.java file. After editing this file, we get data in text file with the required fields.



```java
TwitterSource  start()  new StatusListener  onStatus()

    // The onStatus method is executed every time a new tweet comes in.
    public void onStatus(Status status) {

        logger.debug("tweet arrived");

        headers.put("timestamp", String.valueOf(status.getCreatedAt().getTime()));

    String lang = status.getLang();
    String text = status.getText();
    String url = status.getURLEntities().toString();
    String lat, lon;
    if (status.getGeoLocation()!=null)
    {
    lat = String.valueOf(status.getGeoLocation().getLatitude());
    lon = String.valueOf(status.getGeoLocation().getLongitude());
    }
    else
    {
    lat = "null";
    lon = "null";
    }
    boolean stat = status.isRetweet();
    String ret;
    if(stat)
    ret = "true";
    else
    ret = "false";
    String place = status.getPlace().getFullName();

    String out = lang + "$gna$" + text + "$gna$" + url + "$gna$" + lat +        "$gna$" + lon + "$gna$" + ret + "$gna$" + place + "\n";
        Event event = EventBuilder.withBody(out.getBytes(), headers);

        channel.processEvent(event);
    }
```

Extraction of Twitter data using REST API

The REST API GET search/tweets can also be used for collection of tweets as an alternative. The REST API required the above mention tokens as well, the same tokens can be used.

To access this API, a python script was used. The script contained the 4 tokens and the API request.

The API allows us to mention the following parameters

q - the keywords based on which tweets are to be extracted
geocode - returns tweets of users in a certain geographical radius
lang – restricts tweets to given language
locale - language of the query we are sending
result_ type - allows us to decide if the tweets collected need to be recent, mixed or old
count - the number of tweets to be returned per page
until – returns tweets created before the given date
since_id - returns tweets with a result ID greater than the ID specified
max_id – returns tweets with an ID less than or equal to specified ID
include_entitites - the entities will be excluded when set to false
callback - if supplied the response will use the JSONP format with a callback of the given name

The output file with the required tweets and its attributes will be in json format and will be put in the directory mentioned in the Python Script.

```
#!/usr/bin/python

import json
import oauth2 as oauth

consumer_key = "zDb2aHd5qqxNHTTuetQ308203"
consumer_secret = "KOLYwBDZL5a35pM3uC5A2UF2CBvkYyCPOzIyKmwnImMbz245W7"
access_token = "48966004-B7AB7fstVwf62RjMbkd51xH5YOQyQtJ36CgRJ5vE8"
access_token_secret = "yVOaIu2APNryitpnjWIxP1NOU4dKJejuMtQWPM2b6GU6r"

consumer = oauth.Consumer(key=consumer_key, secret=consumer_secret)
access_token = oauth.Token(key=access_token,  secret=access_token_secret)
client = oauth.Client(consumer, access_token)

timeline_endpoint= "https://api.twitter.com/1.1/search/tweets.json?q=earthquak&result_type=mixed&count=20"

response, data = client.request(timeline_endpoint)

tweets = json.loads(data.decode())
for tweet in tweets:
        print (data)
```

Designing Mapper and Reducer

Mapper and Reducer are working on top of the text file generated by flume.

Mapper:

The mapper class reads data line by line from text file generated by flume.
It then extracts all the parameters language, text, url, latitude, longitude, retweet_status and place.

It then filters the tweets as per the following conditions:
- Language should be English
- Retweet_status should be false
- Word count should be less than 10
- No url should be there in text
- No words like prayer, RIP etc should be present

If a tweet satisfies all these conditions, then it will consider it for analysis.
It will check what crises are mentioned in that tweet using a dictionary given and give the location and name of crisis as key value pair to output collector

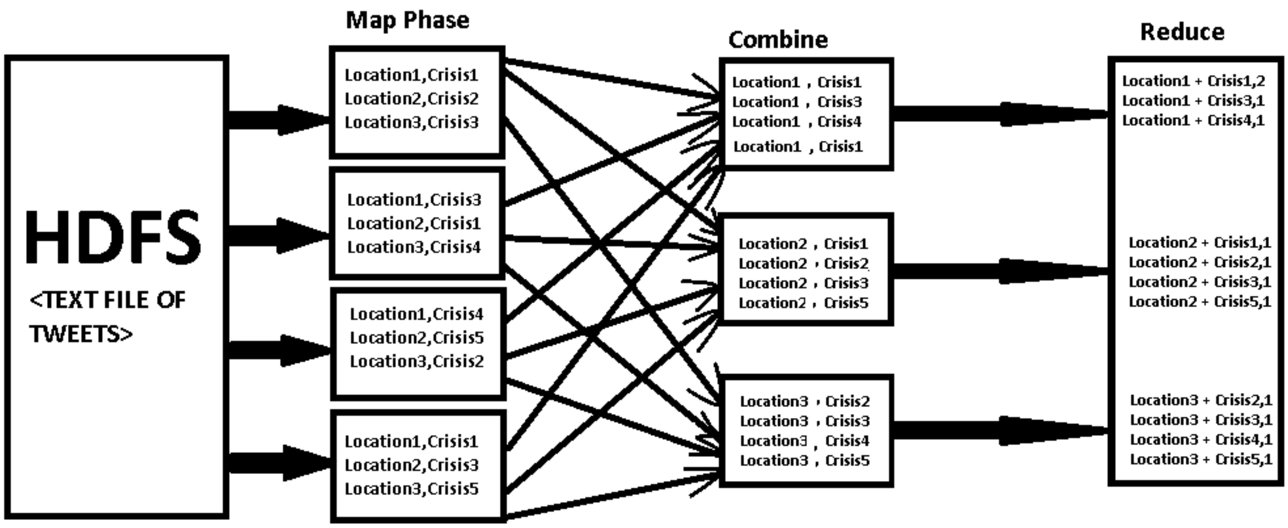
Output format: <location, crisis_name>

Reducer:

The reducer will use the output generated by mapper.
It will take key as a location and iterate over all the values(crisis) happened at that location.
It will use a hash map to temporarily store the count for each crisis happened at that location.
It will finally give location name, crisis name and count to output collector.

Output format: <location + crisis name, count>

The output generated tells us for each location, how many times a particular crisis has happened there.



7.HADOOP MAPREDUCE

Challenges and Roadblocks

1) **Historical Data:**
Twitter does not provide complete historical data with its APIs. At most one-week old data can be collected using the REST API.
As historical data is critical for our project we have to look for a twitter data dump posted by anyone online.
We found such data by archieve.org

2) **Configuring flume to give us the specific data:**
Accessing the json particular fields was a bottle neck since we didn't need to analyze whole data from json. For this purpose, we had to overwrite the TwitterSource.java where we added code to return us only the data that we require. After successful compilation of the java we had two class files which were put into the flume-source-1.0-SNAPSHOT.jar.

To compile the TwitterSource.java it required to import packages like org. apache. flume, twitter4j, org. json. simple and slf4j after all the packages were locally added then we built the java file for class file generation.

Comments Addressed:

Comment 1: Why is Open stack required in your project? Will the results be different in a Map reduce environment without involvement of Open stack?

Reply: Open Stack is not required for our project since the major task is performed by the Hadoop Yarn (Resource Manager) i.e. parallel processing. Initially we thought that we might require open stack capabilities to dynamically control the nodes (name and data) which later on we found that Hadoop YARN can take care of.

Comment2:

a) How are cloud features such as scalability, on-demand etc. incorporated by project?

Reply: Hadoop supports distributed data storage and processing for big data sets across nodes in multiple clusters. Hadoop Common with the core libraries and utilities; the Hadoop Distributed File System (HDFS) for storing data on distributed machines; Hadoop YARN for managing and scheduling cluster computing resources; and Hadoop MapReduce, the programming language for big data processing.

b) What is size of data-set used?

Reply: Size of data set is variable since we are analysing data through flume that gives the live tweets which may vary from few megabytes to gigabytes.
However, the static data set that is json format is around 30 gigabytes.This data set is pretty big to prove the abilities of cloud computing processing.

c) What specific features are being used to identify crisis information?

Reply: To make the results accurate we need to filter out the tweets that won't tell us about the crisis that actually happened at that particular location. Some of the parameters we are using are same as parameters used by the NEIC (National Earthquake Information Center) and others are modified and created as per our needs. The parameters we established are:

- The tweet needs to contain less than 10 words
  (People caught up in a crisis don't write long tweets).
- The tweet should not contain a link to any website
  (Someone might just be sharing an article).
- It should not be a retweet
  (We only need the original tweets to get accurate location of the crisis).
- Tweets containing some keywords need to be ignored, for example, prayer, RIP, Magnitude etc.
  (People tweeting for others with the keywords mentioned. Magnitude won't be known by the people in crisis).
- To get the location there are two ways if the geo feature is enabled, we will directly get the latitude and longitude or else, we will store the full name from the place parameter.

d) Can the quality/trust of information related to crisis be justified using your analysis algorithm?

Reply: People have started to post their entire lives on social media, the food they eat, places they visit etc. It is safe to assume that a lot of people will tweet if their neighborhood gets hit by a crisis.

The results being produced from the application are as accurate as they can be. Some of the parameters of the application to filter tweets are based on the parameters developed by the NEIC in their research on the same topic. We added a couple of filters of our own to make the results more optimal.

However, the precision and reliability of this application cannot be used to get an accurate search on a crisis as the Twitter API do not give access to every tweet on their website. The streaming API gives access to only 1% of the tweets.

FINAL IMPLEMENTATION:

Project Status: Complete

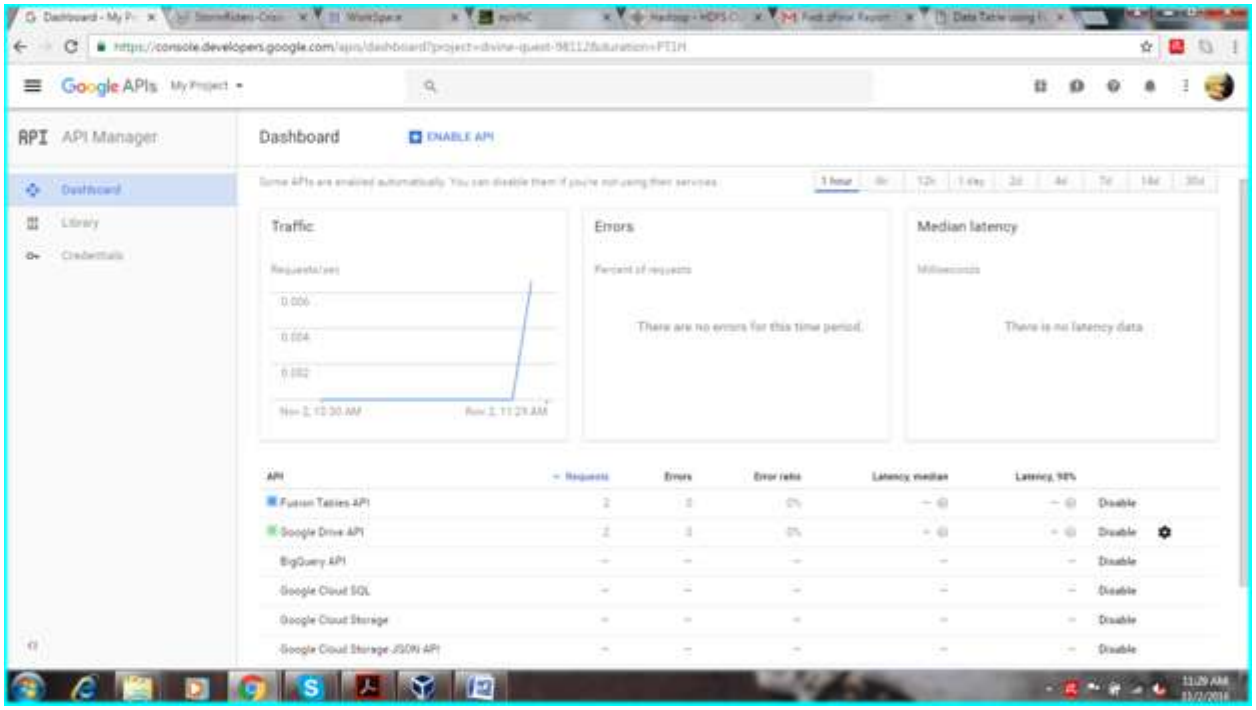
GIT Link: http://gitlab.thothlab.org/gshad/StormRiders-Crisis-Analysis.git
YouTube Link: https://www.youtube.com/watch?v=26zU-7ieyow

Project has progressed as planned initially during the project proposal. Tasks 5,6 and 7 have been completed successfully.

Task Name	Status
Project Proposal and Survey	Complete
Deploying Hadoop on Cloud	Complete
Extraction of Twitter Data using Flume	Complete
Designing Mapper And Reducer	Complete
Data Visualization	Complete
Developing User Interface	Complete
Testing And Refining	Complete

Tasks Completed
Task 5: Data Visualization

1. **Google Fusion Table**
   We used Google Fusion Table APIs to represent the data in the form of maps, tables and charts. We made a Google Developers Service account for this project in which we enabled the following APIs:
   - Google Fusion Tables API
   - Google Drive API
   - Google Maps Java Script API



We also generated an API key for OAuth authentication and p12 key to use in Fusion Tables.
The Fusion Table APIs were accessed using a Java application we developed. Since Fusion Tables only operate data present in the Google Drive, we had to upload data using Google Drive API into a service account for us to operate on it.

```
public static void main(String[] args) throws IOException, GeneralSecurityException {
    // Sample usage
    MyFusionTable myFusionTable = new MyFusionTable();
    myFusionTable.createNewTable("crisis-data", "C:\\Users\\Gaurav\\Desktop\\cc pro\\filteredout.txt");
}

public MyFusionTable() throws GeneralSecurityException, IOException {
    GoogleCredential credential = new GoogleCredential.Builder()
            .setTransport(sHttpTransport)
            .setJsonFactory(JSON_FACTORY)
            .setServiceAccountId("crisis-analysis@divine-quest-98112.iam.gserviceaccount.com") // TODO: Add proper email.
            .setServiceAccountScopes(Arrays.asList(FusiontablesScopes.FUSIONTABLES, DriveScopes.DRIVE))
            .setServiceAccountPrivateKeyFromP12File(new File("C:\\Users\\Gaurav\\Desktop\\cc pro\\My Project-3b882e290dfc.p12"))
            .build();
    mFusionTable = new Fusiontables.Builder(sHttpTransport, JSON_FACTORY, credential)
              .setApplicationName(MY_APP_NAME).build();
    mDrive = new Drive.Builder(sHttpTransport, JSON_FACTORY, credential)
              .setApplicationName(MY_APP_NAME).build();
}
```

2. Creating Fusion Table

The Fusion Table was created using the Fusion Table API, column names and their data types had to be set during this process. This is important, because when the Fusion Table perceive a column as location data type, let it be location names or coordinates, we can plot these locations on a map, or present it in the form of a heat map, using one of the other columns of our choice as the key for that map.

3. Entering data into Fusion Table

Upon setting the appropriate delimiters application, the Fusion Table accepts text files as csv files and load the data into the mentioned table, which is also created using the Fusion Tables API.

```
public String createNewTable(String tableName, String csvFilePath) throws IOException {
    // Must have isExportable set to true when using service account.
    Table table = new Table().setName(tableName).setIsExportable(true);
    List<Column> columns = new ArrayList<>();
    // TODO: Update with real column names and types.
    columns.add(new Column().setName("location").setType("LOCATION"));
    columns.add(new Column().setName("Crisis").setType("STRING"));
    columns.add(new Column().setName("Count").setType("NUMBER"));
    table.setColumns(columns);
    String tableId = mFusionTable.table().insert(table).execute().getTableId();

    mDrive.permissions().insert(tableId, getPermission1()).execute();
    mDrive.permissions().insert(tableId, getPermission2()).execute();

    mFusionTable.table().replaceRows(tableId).setDelimiter("~");
    mFusionTable.table().importRows(tableId,
            new FileContent("application/octet-stream", new File(csvFilePath))).execute();
    tid = tableId;

    return tableId;
}
```

4. Representation

Once the table is created and filled, the application will create html files so that, through the REST API of Fusion tables and Google Maps JavaScript API, we will be able to display the output in various forms like table, bar graph, location map and Heat Map. Each form of out output requires the creation of its own unique html file. When user asks for the same representation of output again the respective old file will be replaced.

```
<script>
  var map, heatmap;
    var heatmapData = [];
  function initMap() {
    map = new google.maps.Map(document.getElementById('map'), {
      zoom: 2,
      center: {lat: 0.0, lng: 0.0},

    });
    heatmap = new google.maps.visualization.HeatmapLayer({
      data: heatmapData,

      map: map
    });

  }

  function feed(results) {

    for (var i = 0; i < results.rows.length; i++) {

      var temp = results.rows[i] + "";
    var coords = temp.split(" ");
    var latLng = new google.maps.LatLng(coords[0], coords[1]);
      heatmapData.push(latLng);

  }

    }
```

Task 6: Developing User Interface

Java Swing is used to create an application to provide Graphical User Interface.

1. **Dataset**
   The user will have the option to start collecting tweets or use a repository of tweets that have already been collected. Should the user choose to start the collection of tweets, the application requires as input, either or both, the location of the crisis and name of crisis.

2. **Integrating GUI with tweet collection and Map Reduce using Process Builder:**
   We needed the application to initiate the tweet collection process. So, we had to save the commands required to initiate the hdfs, flume and the Map Reduce algorithms into a shell script file. Then we used the Process Builder class to execute the commands in the .sh file.

3. **Stopping flume service**
   When the user wishes to stop collecting the tweets. They will have an option to do so, and when they press 'Stop Collecting Tweets', stopflume.sh will be executed and it will kill the flume agent using its process ID.

4. **Procuring the output file**
   The output file is a text file stored in the hdfs. We needed another .sh file with search command in it to copy the text file with the data in it from the hdfs into the local storage. We need the file in the local storage because this makes it easier to use it to create the Fusion Table.

5. **Filtering the output**
   The GUI application will be responsible for filtering this text file to contain only the relevant tweets based on the users input of Crisis name and location and a new file will be created with the relevant data in it.

6. **Connection with Data Visualization application:**
   The class responsible for Java Application responsible for Visualizing the data will be called. The user will be presented with the choice of 4 types of output, table, bar graph, location map and heat map. Based on their choice, a html file will be created and it will be opened using the respective .sh file.
   The user will just have to retype the queries and press 'Search' again if they want to conduct a new search.

Task 7: Testing & Refining

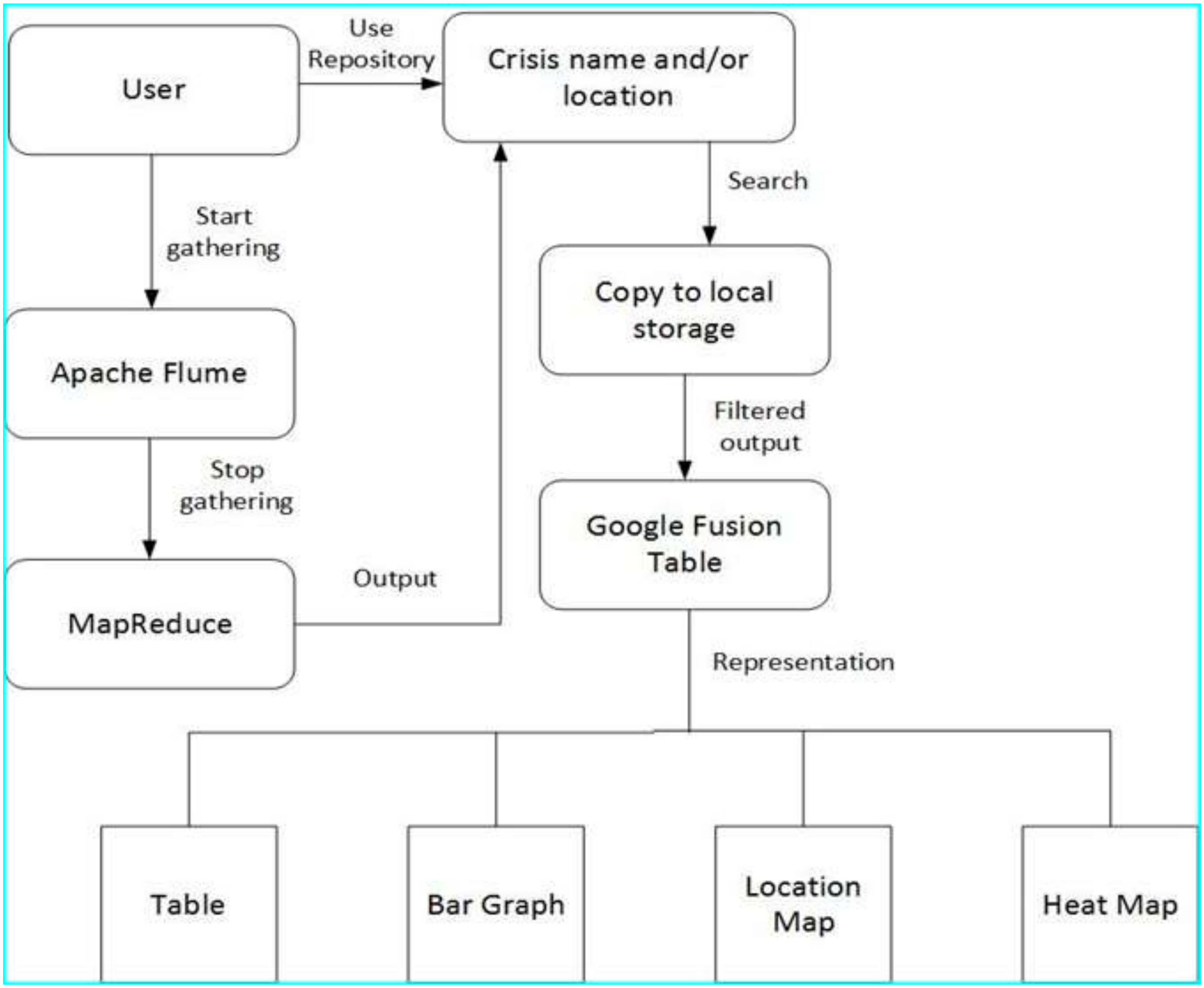
1. **Testing:**
   We tweaked the MapReduce to improve the performance of the system.

2. **Refining:**
   Although we were filtering the irrelevant tweets using our MapReduce algorithm, a significant number of them were still getting through. So, we had to add more keywords to the MapReduce which would only be present in the irrelevant streams. After coming up with enough words, we could filter out most of the irrelevant tweets. This significantly reduced the number of false positives we were getting in the result.

Application Workflow
The application is responsible for managing interaction between Hadoop, Cloud and twitter.

1. First it asks the user if they want to use the repository or start collecting a fresh batch of tweets.
2. Should the user choose to collect the fresh batch of tweets, the GUI starts Apache FLUME and requests it to start collecting tweets.

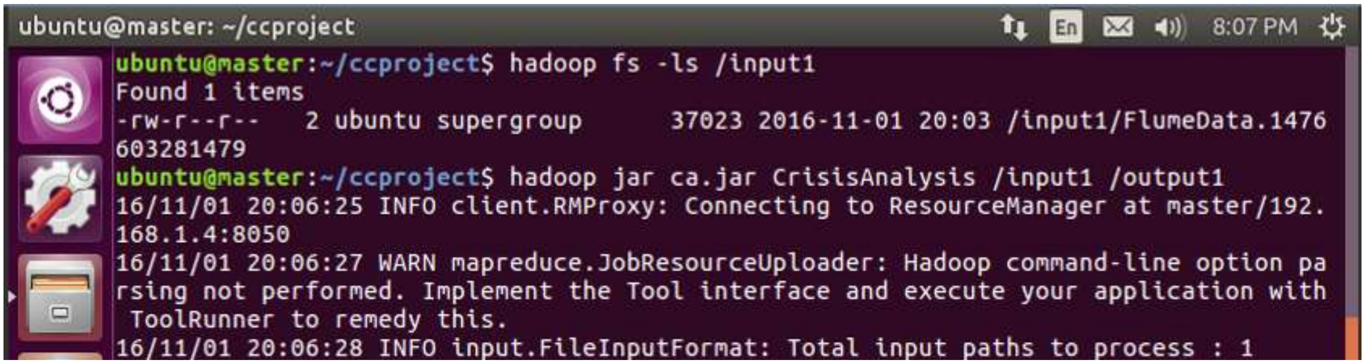
3. After the user chooses to stop collecting tweets, the file with tweets will be operated on by Map Reduce so the irrelevant tweets will be removed in the Mapper class. The output of the Mapper class will then be combined with the Reducer class to generate the filtered output.
4. The application will then download this file into the local storage.
5. The user is now asked for either or both the location and crisis name they want to conduct the search on. The application finds the tweets matching these parameters and stores them in a new file.
6. This file is then uploaded to the Google Drive as a Fusion Table.
7. To cross check we can login to Google Drive, table is created. We can click on Map of Location which will plot point on the map.
8. The user is now asked the type of representation they would prefer the output in. Based on their input the application creates a HTML file will be then opened to show the user the final output.



8.APPLICATION WORKFLOW

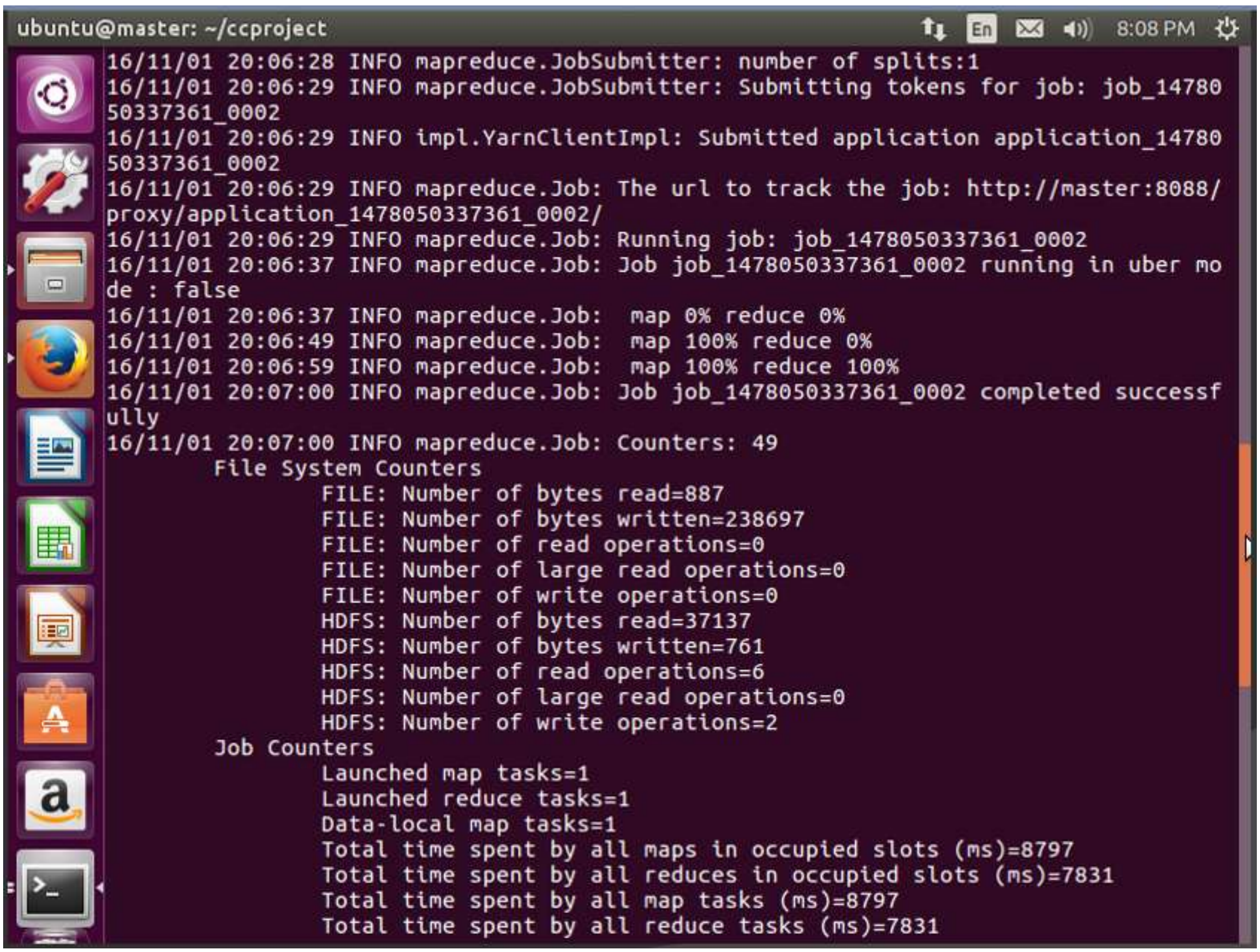
Screen-Shots:

1. **Running Hadoop Map Reduce:**



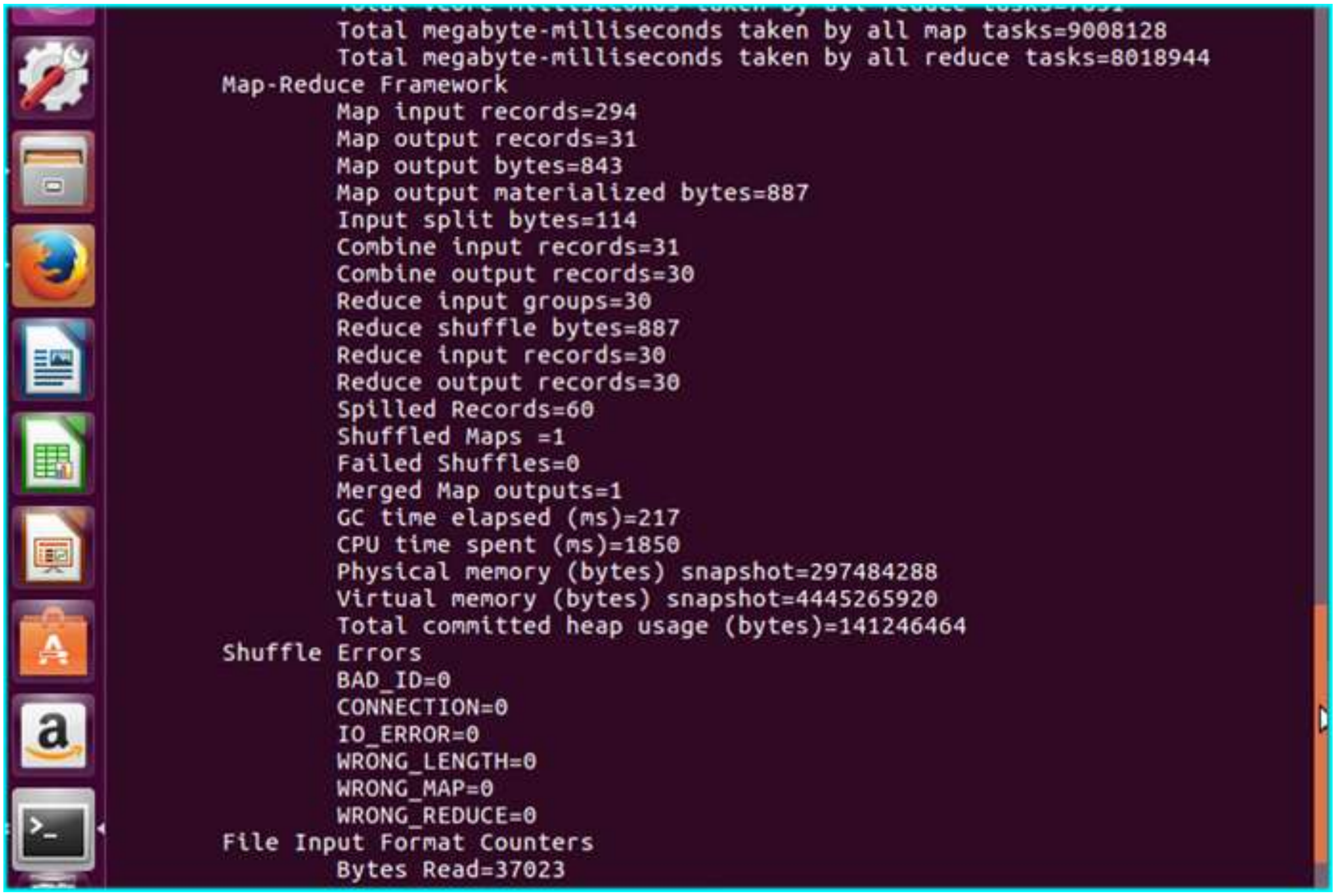
```
ubuntu@master: ~/ccproject                                    8:07 PM
ubuntu@master:~/ccproject$ hadoop fs -ls /input1
Found 1 items
-rw-r--r--   2 ubuntu supergroup      37023 2016-11-01 20:03 /input1/FlumeData.1476
603281479
ubuntu@master:~/ccproject$ hadoop jar ca.jar CrisisAnalysis /input1 /output1
16/11/01 20:06:25 INFO client.RMProxy: Connecting to ResourceManager at master/192.
168.1.4:8050
16/11/01 20:06:27 WARN mapreduce.JobResourceUploader: Hadoop command-line option pa
rsing not performed. Implement the Tool interface and execute your application with
 ToolRunner to remedy this.
16/11/01 20:06:28 INFO input.FileInputFormat: Total input paths to process : 1
```

2. Mapping and reducing on a file:



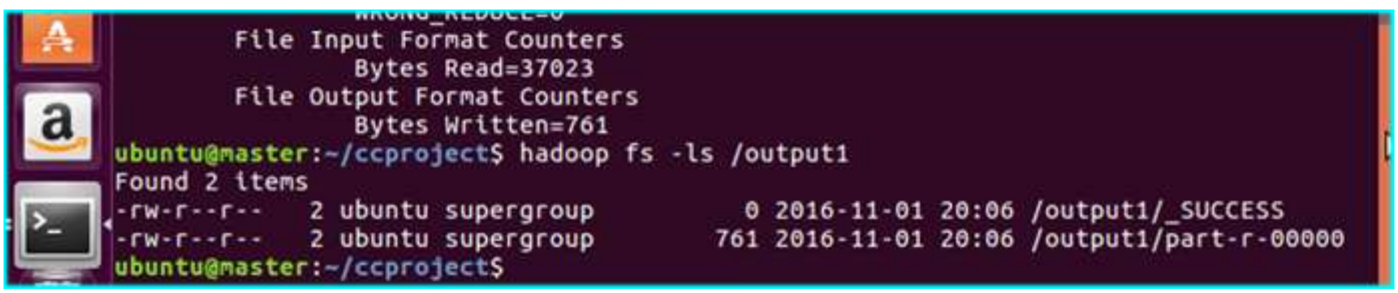
```
ubuntu@master: ~/ccproject                                    8:08 PM
16/11/01 20:06:28 INFO mapreduce.JobSubmitter: number of splits:1
16/11/01 20:06:29 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_14780
50337361_0002
16/11/01 20:06:29 INFO impl.YarnClientImpl: Submitted application application_14780
50337361_0002
16/11/01 20:06:29 INFO mapreduce.Job: The url to track the job: http://master:8088/
proxy/application_1478050337361_0002/
16/11/01 20:06:29 INFO mapreduce.Job: Running job: job_1478050337361_0002
16/11/01 20:06:37 INFO mapreduce.Job: Job job_1478050337361_0002 running in uber mo
de : false
16/11/01 20:06:37 INFO mapreduce.Job:  map 0% reduce 0%
16/11/01 20:06:49 INFO mapreduce.Job:  map 100% reduce 0%
16/11/01 20:06:59 INFO mapreduce.Job:  map 100% reduce 100%
16/11/01 20:07:00 INFO mapreduce.Job: Job job_1478050337361_0002 completed successf
ully
16/11/01 20:07:00 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=887
                FILE: Number of bytes written=238697
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=37137
                HDFS: Number of bytes written=761
                HDFS: Number of read operations=6
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Launched map tasks=1
                Launched reduce tasks=1
                Data-local map tasks=1
                Total time spent by all maps in occupied slots (ms)=8797
                Total time spent by all reduces in occupied slots (ms)=7831
                Total time spent by all map tasks (ms)=8797
                Total time spent by all reduce tasks (ms)=7831
```

3. Map Reduce complete bytes read:



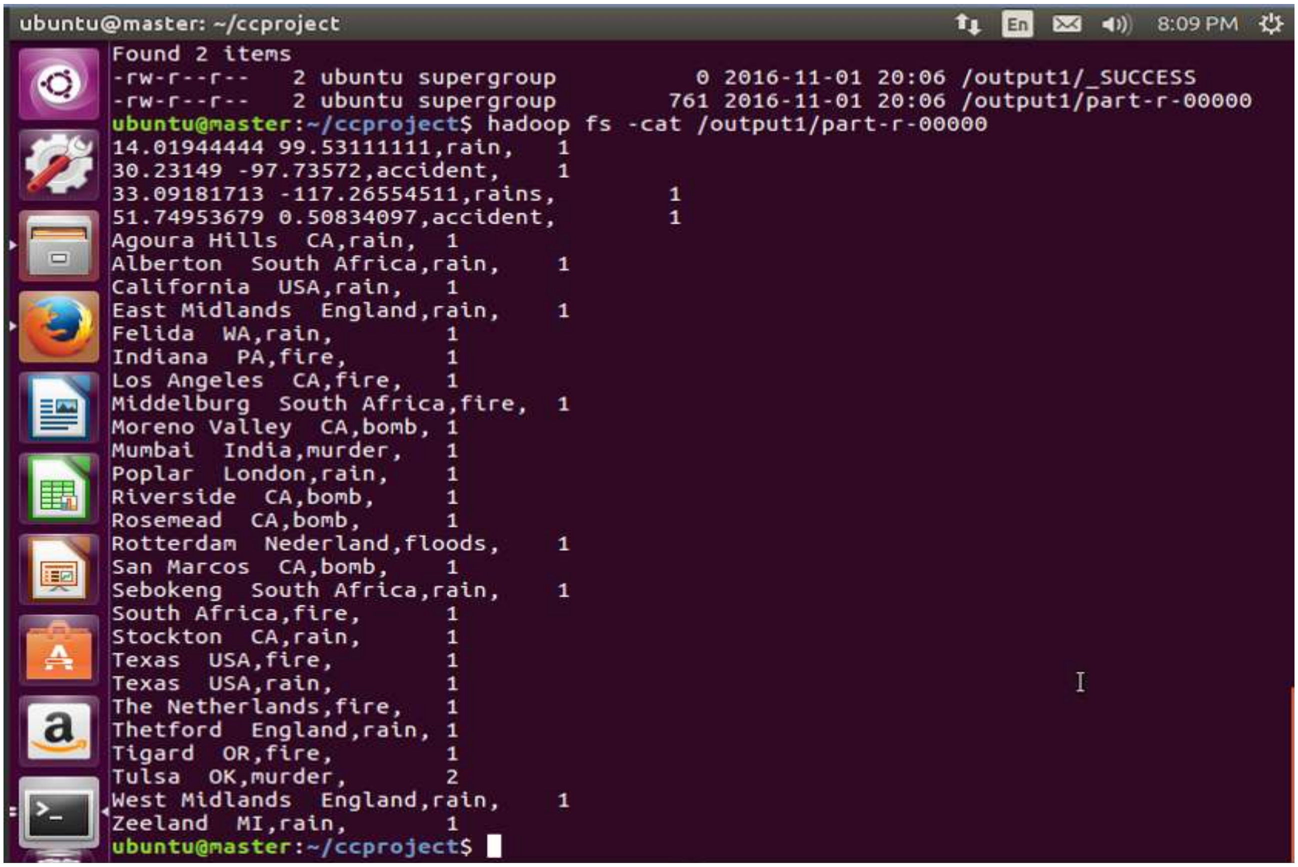
```
		Total megabyte-milliseconds taken by all map tasks=9008128
		Total megabyte-milliseconds taken by all reduce tasks=8018944
	Map-Reduce Framework
		Map input records=294
		Map output records=31
		Map output bytes=843
		Map output materialized bytes=887
		Input split bytes=114
		Combine input records=31
		Combine output records=30
		Reduce input groups=30
		Reduce shuffle bytes=887
		Reduce input records=30
		Reduce output records=30
		Spilled Records=60
		Shuffled Maps =1
		Failed Shuffles=0
		Merged Map outputs=1
		GC time elapsed (ms)=217
		CPU time spent (ms)=1850
		Physical memory (bytes) snapshot=297484288
		Virtual memory (bytes) snapshot=4445265920
		Total committed heap usage (bytes)=141246464
	Shuffle Errors
		BAD_ID=0
		CONNECTION=0
		IO_ERROR=0
		WRONG_LENGTH=0
		WRONG_MAP=0
		WRONG_REDUCE=0
	File Input Format Counters
		Bytes Read=37023
```

4. Map And reduce complete Bytes Written:

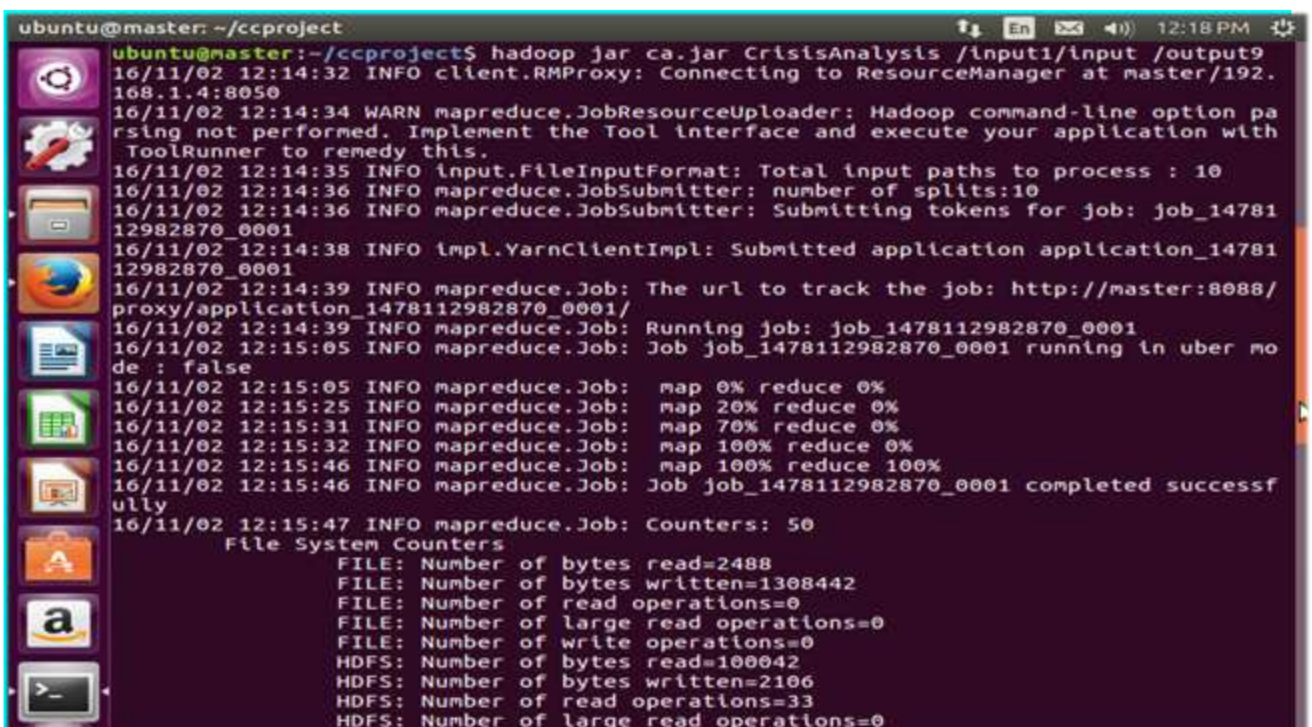


```
	File Input Format Counters
		Bytes Read=37023
	File Output Format Counters
		Bytes Written=761
ubuntu@master:~/ccproject$ hadoop fs -ls /output1
Found 2 items
-rw-r--r--   2 ubuntu supergroup          0 2016-11-01 20:06 /output1/_SUCCESS
-rw-r--r--   2 ubuntu supergroup        761 2016-11-01 20:06 /output1/part-r-00000
ubuntu@master:~/ccproject$
```

5. Output File:

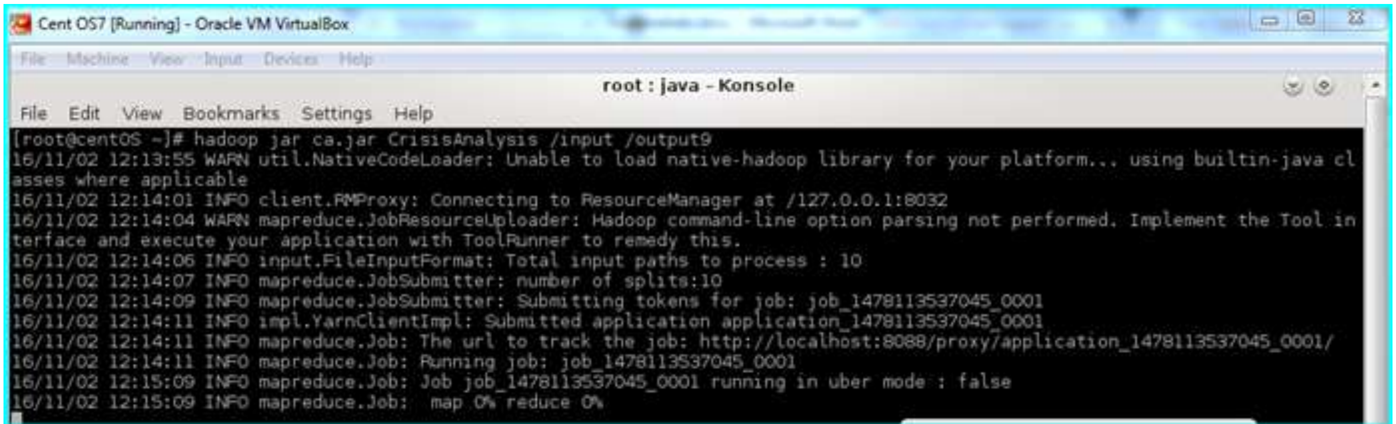


Single Node Vs Multi-Node Map Reduce



9. Sample Input of 10 files to a three-node cluster

Job completed in 1m15s



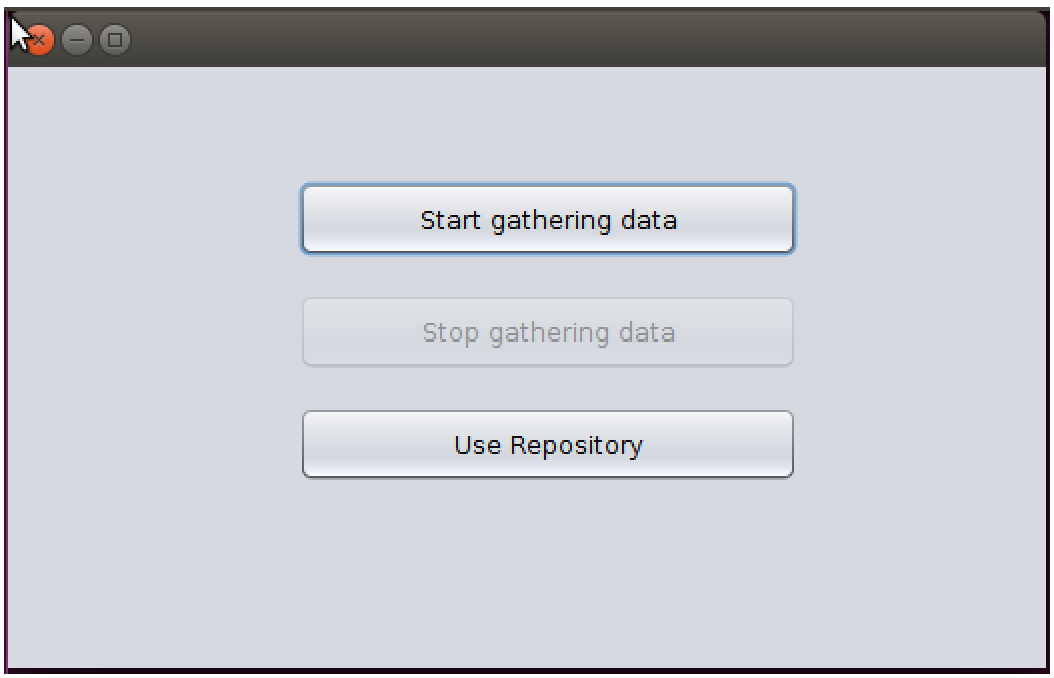
10. Sample Input of 10 files to a single node

Job just started executing in that time frame

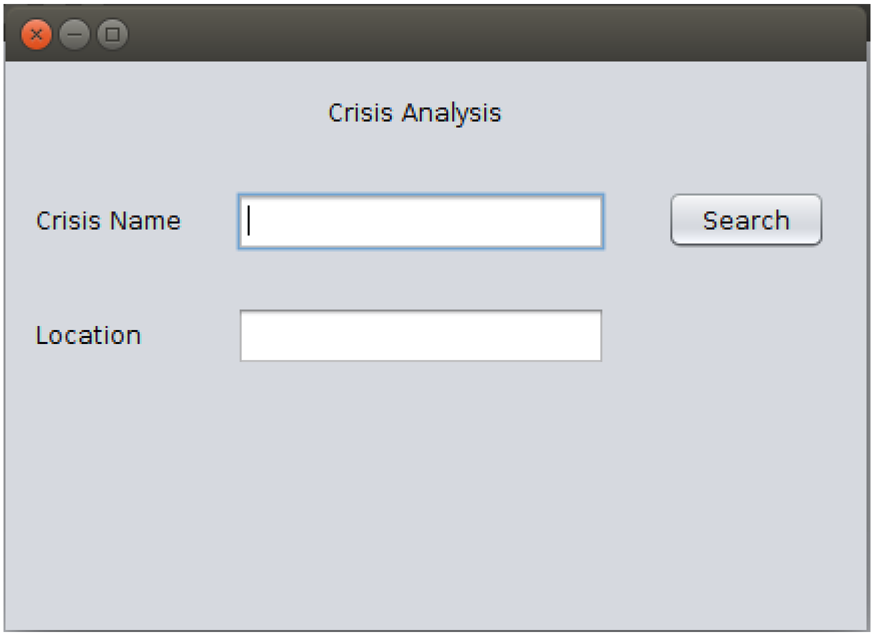
This illustrates the cloud feature of resource pooling to provide more computing power.

UI Design:

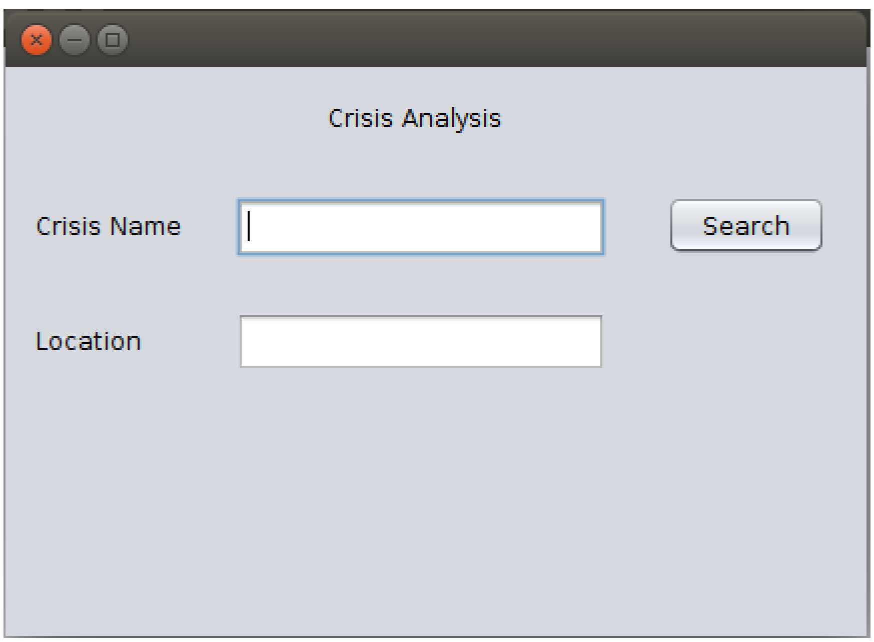
1. **Home Page(Main Screen)**



2. User-Input Page:



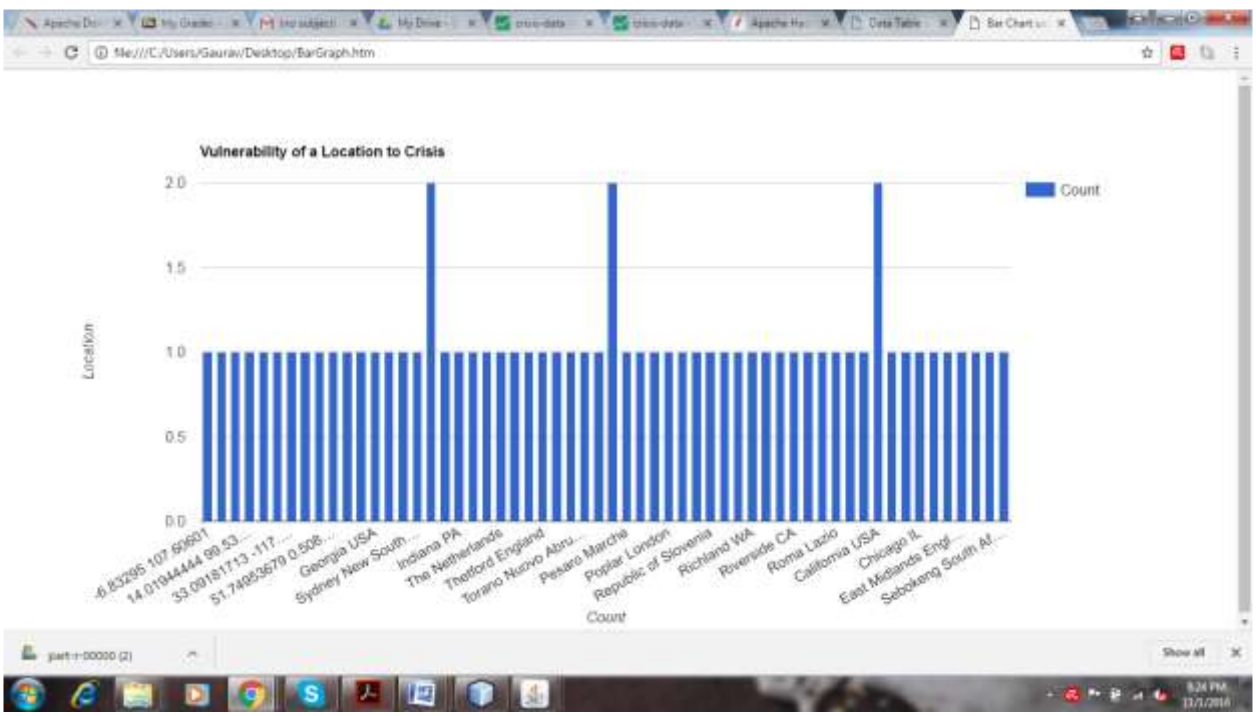
3. User Visualization Options(User Input Screen):



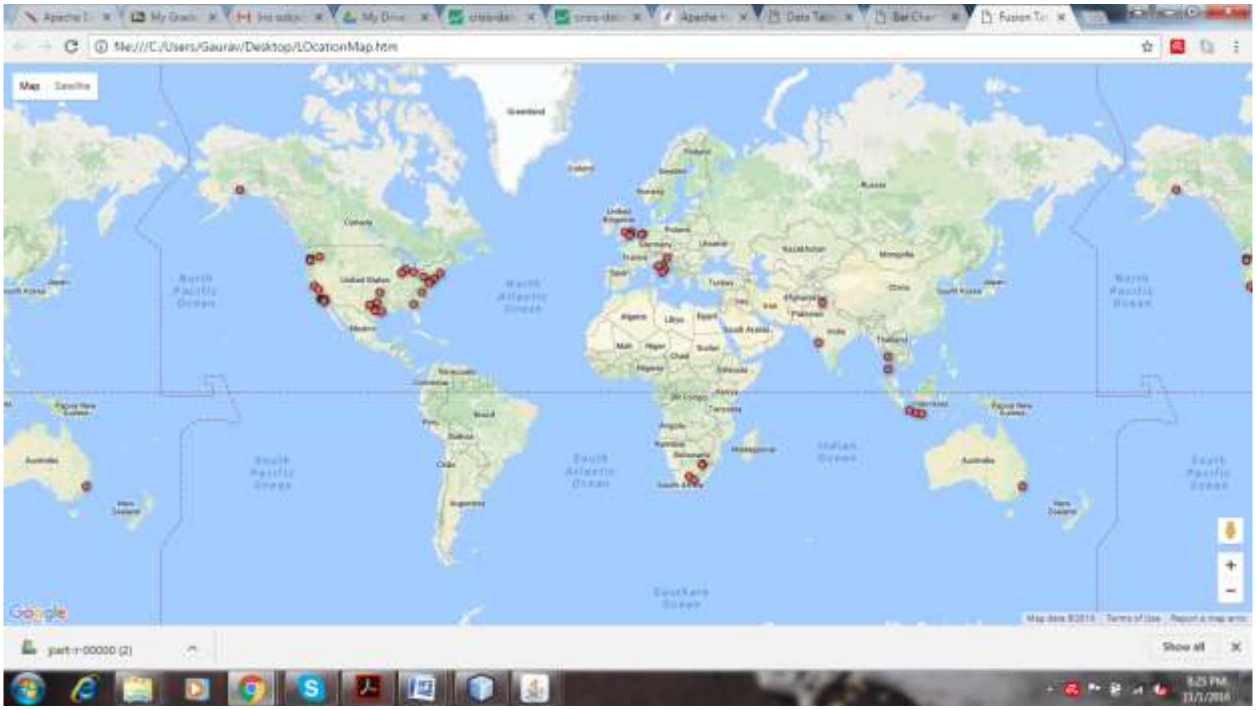
4. Table View (Output Screen):

	Location	Crisis	Count
1	-6.83295 107.60601	rain	1
2	-7.78127 110.39446	fire	1
3	-7.9637757 112.6195475	rain	1
4	14.01944444 99.53111111	rain	1
5	30.23149 -97.73672	accident	1
6	30.26713 -97.74316	accident	1
7	33.09181713 -117.26554511	rains	1
8	38.901 -77.04047	fire	1
9	40.24829 -75.34584	accident	1
10	51.74953679 0.50834097	accident	1
11	Florence Tuscany	earthquake	1
12	South Africa	fire	1
13	Georgia USA	murder	1
14	Stockton CA	rain	1
15	Greensboro NC	gun	1
16	Sydney New South Wales	rain	1
17	Houston TX	bomb	2
18	Texas USA	fire	1
19	Indiana PA	fire	1
20	Texas USA	rain	1
21	Irving TX	bomb	1
22	The Netherlands	fire	1
23	Los Angeles CA	fire	1
24	Middelburg South Africa	fire	1
25	Thetford England	rain	1
26	Tigard OR	fire	1
27	Moreno Valley CA	bomb	1
28	Torano Nuovo Abruzzo	earthquake	1
29	Mumbai India	murder	1

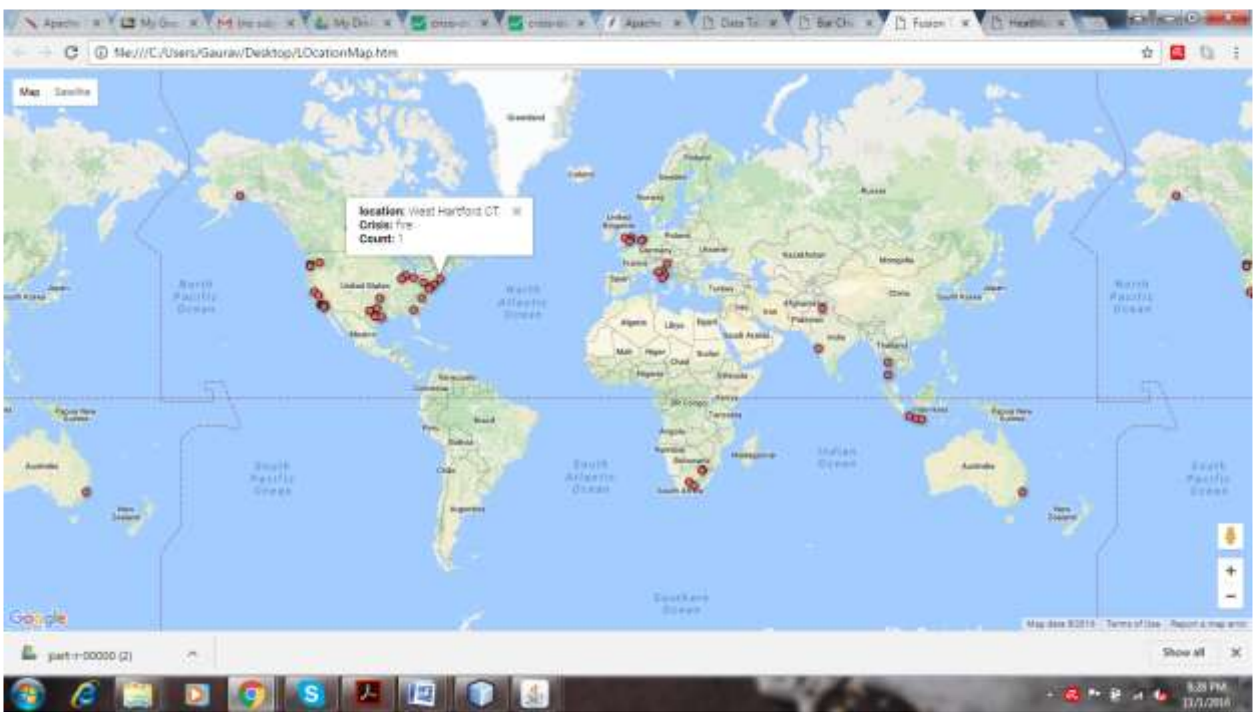
5. Bar Graph View (Output Screen):



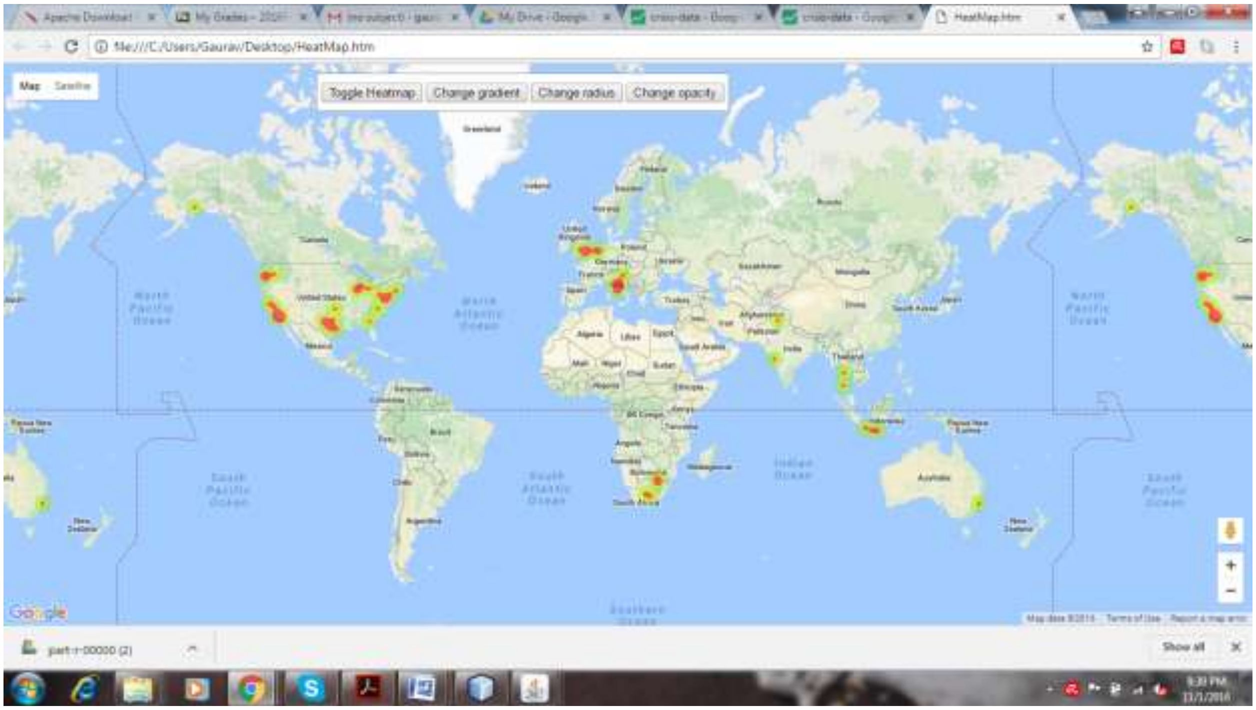
6. Location Map (Output Screen):



Clicking over a marker gives crisis name and its tweet count



7. Heat Map (Output Screen):



Challenges and Roadblocks

Selection of Visualization Tool

Selecting data visualization tool was one of the major challenges since the initial plan was to use IBM BigInsights, upon researching about it we found that its component BigSheets cannot be used individually complete BigInsights needs to be installed. Installation requirements of Biginsights is way too high it requires 12GB RAM, 100 GB of Hard Disk which is greater than the resources that we have. Moreover, it installs and runs its own Hadoop. On further research, we found the best match for our use is Google Fusion Tables.

Our project Crisis Analysis can be best related to following two cloud features:

1. **On Demand Self Service**
2. **Rapid Elasticity**
3. **Broad Network Access**

On Demand Self Service

If a user wants to gather more data, it can be done using a start button given in UI Homepage this action will start flume which will keep on gathering the data till the user stops it. The user does not have to worry about where this data is being stored the Hadoop Distributed File System will create new blocks (Size: **128Mb** in our case) to hold the streaming data which is spread over **two other systems Slave 1 and Slave 2.** So, till the time the other system does not reach their full capacity new storage will keep on allocating for use.
On submitting a job to Hadoop, the resource manager i.e. YARN will allocate the required resources on two other systems Slave1 and Slave 2 to process the data. It will process data in parallel on these two nodes, hence, combining the resources from both.

This is analogous to the cloud feature “On Demand Self Service” as user is getting all the computing resources as in when required which is being managed by Hadoop automatically.

Rapid Elasticity

Computing capabilities are dynamically allocated as per the requirement of the submitted job. If the submitted job needs processing on large data set more resources will be allocated this is taken care of by YARN e.g. more Yarn Childs will run on both the slaves once a job is successfully completed Yarn will release the allocated resources and it will be available for the next job.

Broad Network Access

Since our application is running on a Virtual Machine hosted on thothlab(IaaS) it can be accessed from any location.

IV. Risk Management of the Project

Risk	**Mitigation Strategy**
Hadoop installation and configuration is a tricky. Issue can come up with the configuration of xml files.	To solve this issue, installation must be done with proper time and care.
Lack of large data sets from Twitter because of rate limit of Twitter API	Twitter API must be run in parallel and flume must be used to get more data
Distinguishing between normal tweets and useful tweets (crisis information)	Conditions needs to be improved so as filter out the data precisely

V. Conclusion

To summarize, this project results in data visualization from the output of MapReduce algorithm. It returns us a graphical representation of the crisis that already happened in a region, queried in by the user. The tweets are streamed using Hadoop, Flume and Twitter API. Mapper and Reducer will be used to filter out the tweets based on certain conditions and process the required data. A user interface will help user interact with this big data analysis application to gain knowledge on this topic.

Futurework:
In the Mapper and Reducer, K-means or some other clustering method can be implemented to improve the quality of data and efficiency of the application. Additionally, data from other social media like Facebook can also be extracted and processed to return more meaningful data.

Acknowledgment

We would like to express our gratitude towards our professor, Dr. Dijiang Huang and Teaching Assistant Mr. AnkurChowdhary for assisting and giving us valuable inputs so that we could deliver it in time.

References

[1] Shamanth Kumar, Fred Morstatter, Huan Liu "Twitter Data Analytics" http://tweettracker.fulton.asu.edu/tda/TwitterDataAnalytics.pdf

[2] “How the USGS uses Twitter data to track earthquakes”
https://blog.twitter.com/2015/usgs-twitter-data-earthquake-detection

[3] “How-to: Analyze Twitter Data with Apache Hadoop”
http://blog.cloudera.com/blog/2012/09/analyzing-twitter-data-with-hadoop/

[4] Manoj Kumar Danthala, Dr. Siddhartha Ghosh "Bigdata Analysis: Streaming Twitter Data with Apache Hadoop and Visualizing using BigInsights"
https://www.ijert.org/view-pdf/13162/bigdata-analysis-streaming-twitter-data-with-apache-hadoop-and-visualizing-using-biginsights

[5] “Hadoop” http://hadoop.apache.org/

[6] “FLUME” http://flume.apache.org/

[7] “Open Stack” http://www.openstack.org/

[8] “Twitter Developers API” https://dev.twitter.com/docs/api/1.1/get/search/tweets