REPORT ON BIG DATA ANALYSIS

Submitted By: Hastimal Jangid

Neha Choudhary

Tarun Shedhani

Motivation:

Technology is growing faster by every passing day. We have invented so many devices till date to make our life easier; mobile phone is one of the best example. It allows us to stay in contact with our loved ones sitting miles away however, this definition of mobile phone has gone outdated. These days mobile phones are not just a medium of conversation but it has become "smart"; it offers various other amazing features and therefore, we call it "smart phone". After its grand success Apple, the renowned mobile company brought "Apple Watch in market". It is not just a wrist watch but it has better features than what your mobile has. It allows you to accept or reject phone calls, read your messages, play game, listen to music and above all it keeps track of your physical activity.

Right after the announcement of its release, we analyzed the overflow of news related to apple Watch on social media; we were amazed. Craziness of people about the Apple Watch inspired us to accomplish big data analytics on tweets that are related to "Apple Watch".

Introduction:

Big Data needs no introduction these days. We all know it well that big data is about performing some computation on extremely large data set to analyze some trend, behavior or pattern. It reduces the cost, time and optimizes the result that helps in risk reduction.

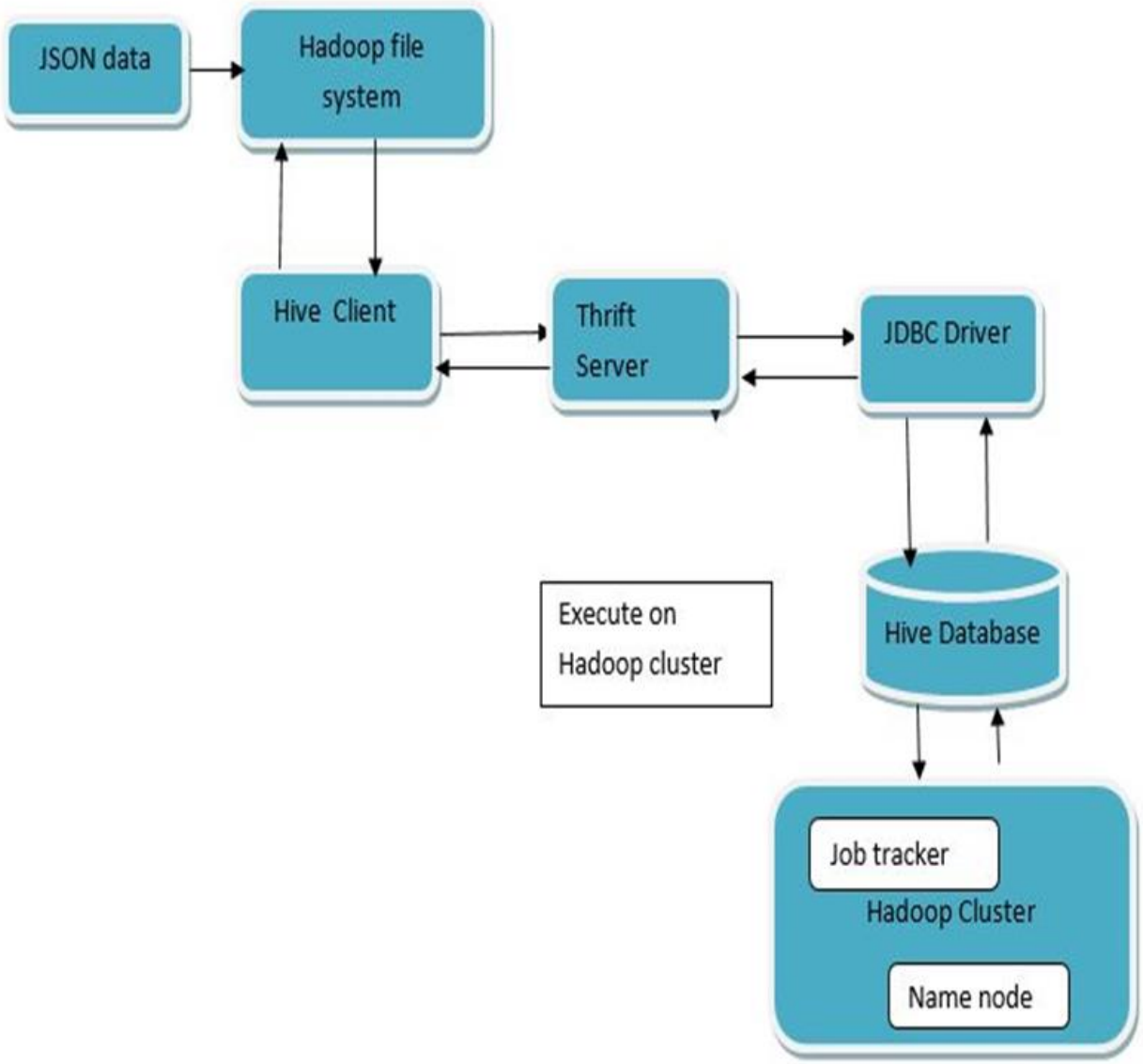
In order to analyze big data, map reduce programs are required and practically, it become so complex that one cannot proceed with it. Apache Hive, a data warehouse infrastructure for Hadoop makes querying easier. Supporting large dataset and providing data summarization are few advantages of Hive. Hive supports HiveQL, which is query like SQL. HiveQL is compiled as map reduce program and further executed using Hadoop.

Tool Implementation:

In this project, we ran hive queries in two ways.

1. Using Hive shell, which is a command line interface. It takes HiveQL as input and after the execution it stores results in hive database.
2. Using Hive JDBC/ODBC drivers. It fetches, executes and handles the data. Later on compiler parses and optimizes the execution using Hadoop map reduce system.

Design Architecture:



Components:

1. **Json Data:** Hive takes structured data. Tweets that are collected from the Twitter website, in form of json file, further processed using map reduce system on Hive.

2. **Hadoop file system:** Later Json data is moved to Hadoop file system from the local file system to make the file accessible to hive.

3. **Hive Client:** Hive client takes user's queries as input and process it further.

4. **Thrift server:** It works as an interface between clients the JDBC driver and passes the request.

5. **JDBC driver:** It provides database access to the client.

6. **Hive Database:** Hive database stores entire dataset and query result.

7. **Hadoop Cluster:** Hive resides on the top of Hadoop. Every client request is processed using map reduce system on Hadoop cluster.

Modules:

There are four modules in our project. The basic idea of our project is to analyze the data. Data that we are using here are tweets, collected on keyword "Apple Watch". We are using twitter streaming API to collect tweets. We created a twitter application to collect the streaming tweets that are directly coming from the twitter website.

1. **Tweet collection from twitter website:** In the first module, we analyzed and collected tweets on "Apple Watch". Tweets are collected by running java code in eclipse; file that contains tweets is originally in JSON file format.

2. **Storing and processing the tweets in Hive:** We are using "Hive" as our database. Hive supports structured file and hence, we are converting our JSON file into CSV. Further we are organizing our data in six different tables to pose queries on it.

3. **Processing various queries on stored data:** We have written several queries using various hive functions. All query results are loaded into six different tables named "user_info", "user_count_info", "tweet_info", "tweet_text_info" "re_tweet_info", "re_tweet_text_info". These tables are later on used for visualizing our overall analysis on "Apple Watch".

4. **Output visualization by different graphical representations:** We selected only few columns from the output table to focus on our idea. In the next step we processed the output to show it in the form of graphs and charts. We completed the visualization of our project using bar graph, bubble chart, donut chart and pie chart.

Analytical Queries:

1. **Collecting tweets on "Apple Watch" from top 10 countries from which we are getting highest number of tweets.**

```
SELECT tweet_place_ctry_code, count(*) count from tweet_info
WHERE  tweet_place_ctry_code != "null"
group by tweet_place_ctry_code
having tweet_place_ctry_code is not null
ORDER BY count(*)
```

2. **Collecting tweets on "Apple Watch" in top 10 languages.**

```
SELECT t_info.tweet_lang, count(*) count from tweet_info t_info
GROUP BY t_info.tweet_lang
HAVING t_info.tweet_lang is not null
ORDER BY count DESC;
```

3. **Looking for user contribution in a given data set.**

```
SELECT u_info.user_screen_name ,count(t_text_info.tweet_text) count
FROM user_info u_info
JOIN tweet_text_info t_text_info ON (u_info.tweet_id = t_text_info.tweet_id)
GROUP BY u_info.user_screen_name
HAVING u_info.user_screen_name is NOT NULL
ORDER BY count DESC LIMIT 100;
```

4. **Finding popular users (based on the number of follower's count) tweeting on "Apple Watch".**

```
SELECT u_info.user_name, u_count_info.user_followers_count
from user_info u_info JOIN user_count_info u_count_info
ON (u_info.user_id=u_count_info.user_id)
ORDER BY u_count_info.user_followers_count DESC;
```

5. **Finding active user (based on no. of tweets) who tweeted on "Apple Watch".**

```
SELECT u_info.user_screen_name,u_count_info.user_statuses_count
```

from user_info u_info JOIN user_count_info u_count_info

ON (u_info.user_id=u_count_info.user_id)
WHERE u_info.user_screen_name IS NOT NULL
ORDER BY u_count_info.user_statuses_count DESC;

6. **Collecting tweets based on time interval. We have divided the day into four quarters.**

- **0 – 6**
- **6-12noon**
- **12 -18**
- **18-24 AM**

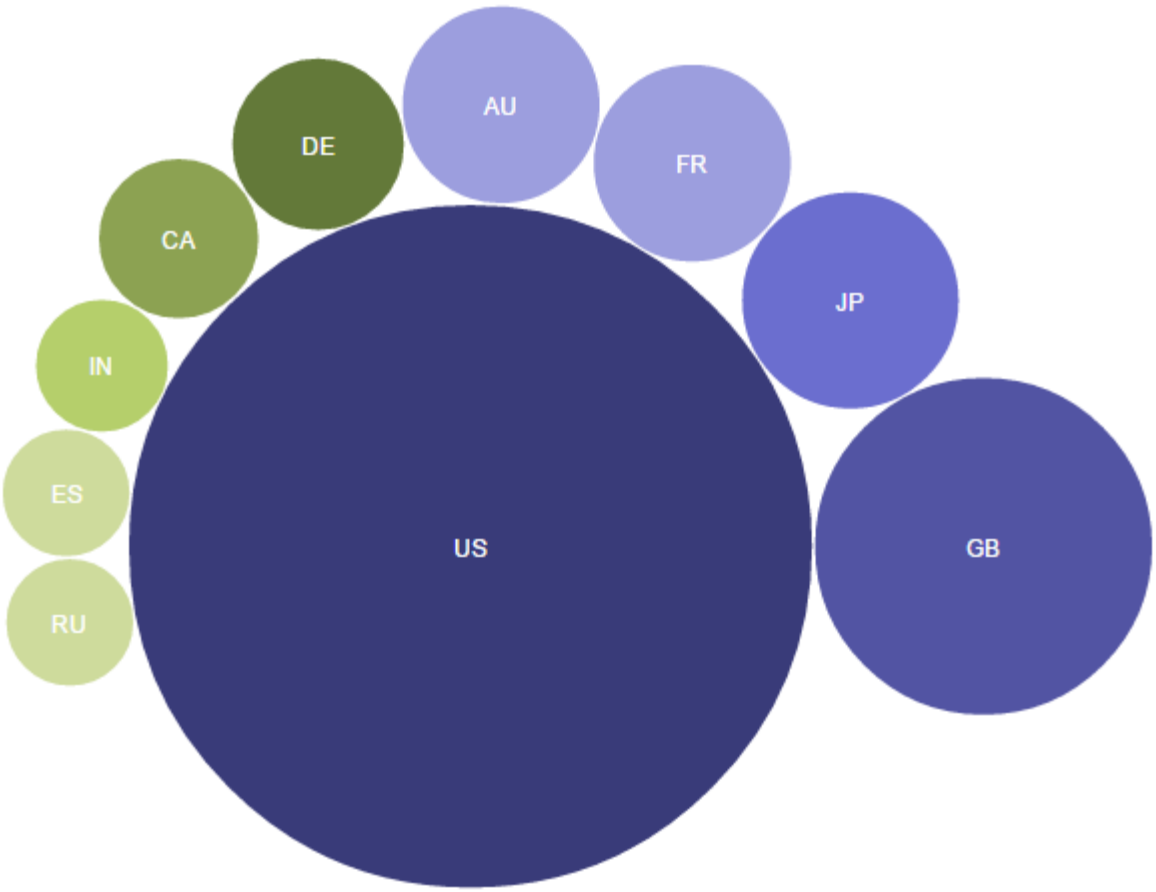
SELECT * from tweet_info where tweet_created_at is not NULL
For this query we are using maps in Java to make the visual output.

```
Connection con = ConnectionFactory.getConnection();
Map<String,Long> m1 = new HashMap<String,Long>();
String FirstQ = "0 AM - 5 AM", SecondQ = "6 AM - 11 AM", ThirdQ = "12 PM - 17 PM ", FourthQ = " 18 PM - 23 PM";
long value = 0;
```

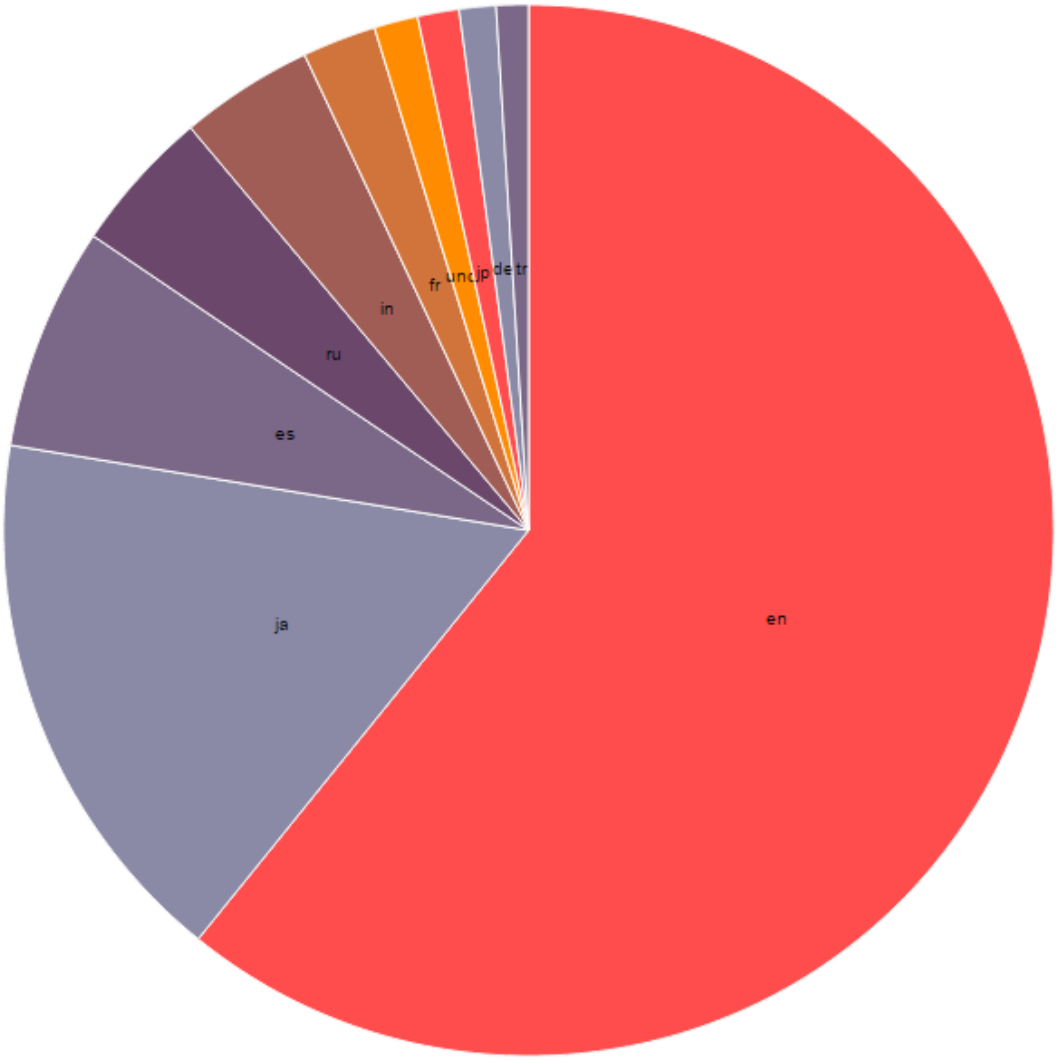
```
if (m1.isEmpty()){

    if (date.getHours() >= 0 && date.getHours() < 6){

        m1.put(FirstQ, (long) 1);
    }
    else if (date.getHours() >= 6 && date.getHours() < 12){

        m1.put(SecondQ, (long) 1);
    }
    else if (date.getHours() >= 12 && date.getHours() < 18){

        m1.put(ThirdQ, (long) 1);
    }
    else {
        m1.put(FourthQ, (long) 1);
    }
```

Screen Shots (Visualization):

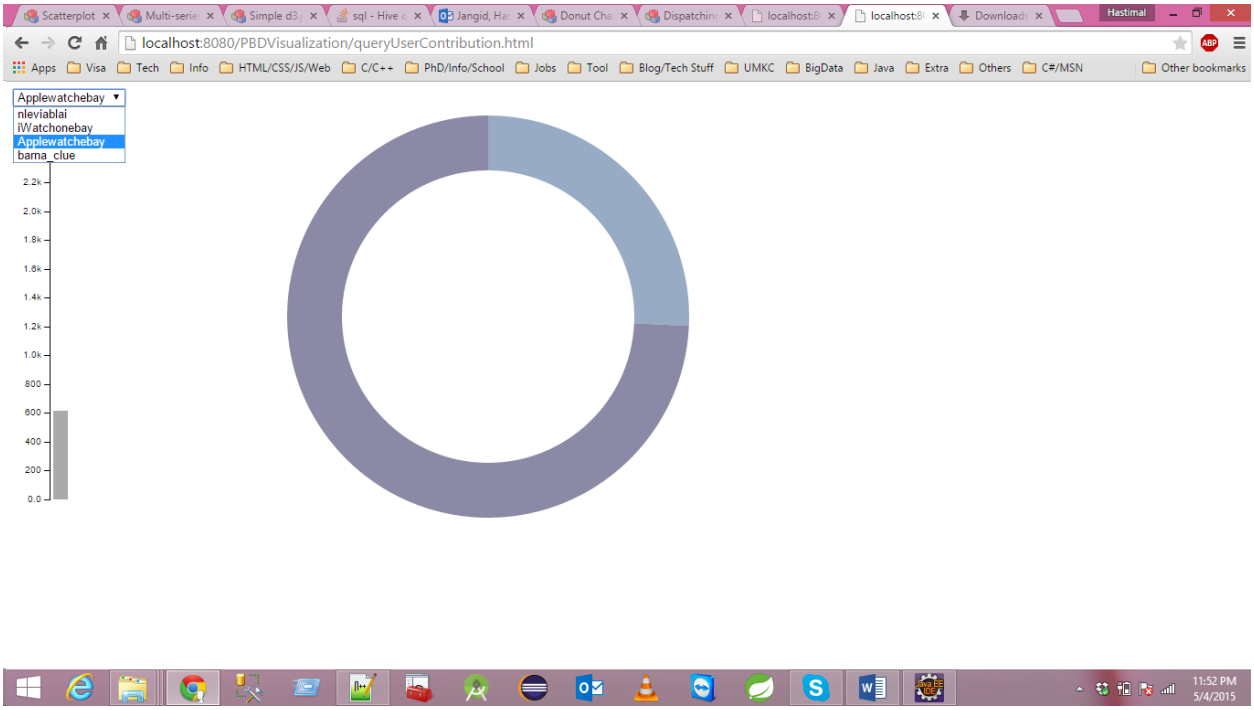
Query 1:



Query 2:

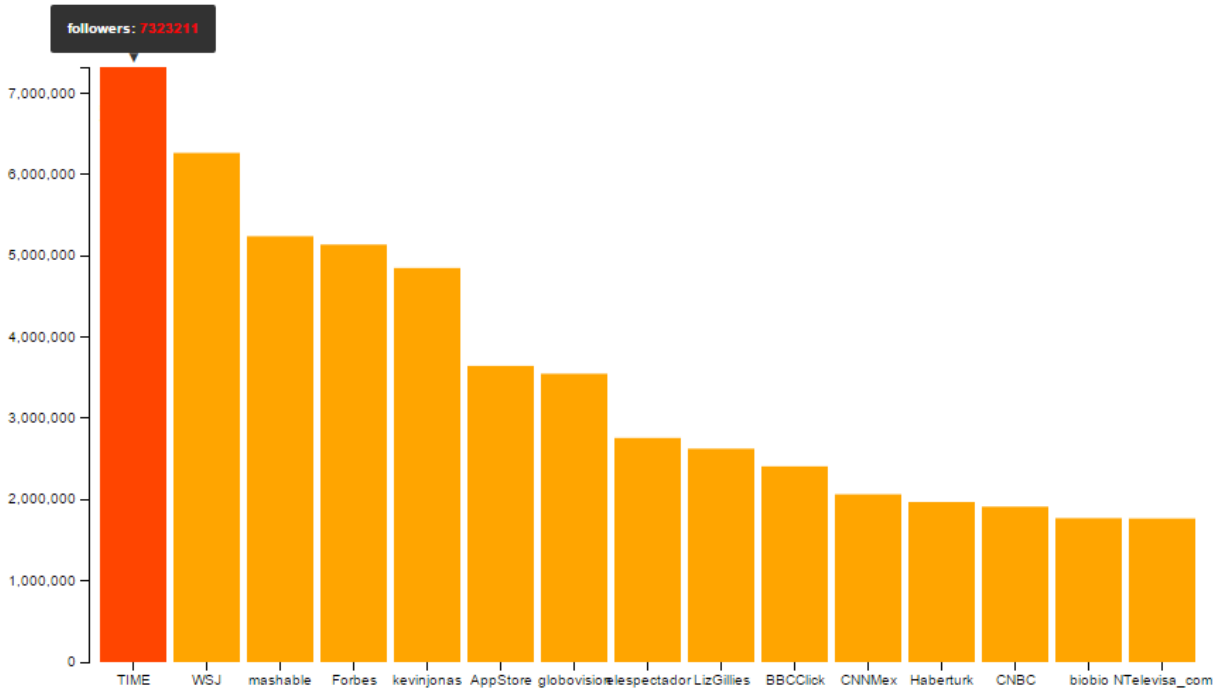


Query 3:


Scatterplot
Multi-serie
Simple d3
sql - Hive
Jangid, Ha
Donut Cha
Dispatchin
localhost:8
localhost:8
Downloads
Hastimal
localhost:8080/PBDVisualization/queryUserContribution.html
Apps
Visa
Tech
Info
HTML/CSS/JS/Web
C/C++
PhD/Info/School
Jobs
Tool
Blog/Tech Stuff
UMKC
BigData
Java
Extra
Others
C#/MSN
Other bookmarks
Applewatchebay
nleviablai
iWatchonebay
Applewatchebay
barna_clue
2.2k
2.0k
1.8k
1.6k
1.4k
1.2k
1.0k
800
600
400
200
0.0
11:52 PM
5/4/2015

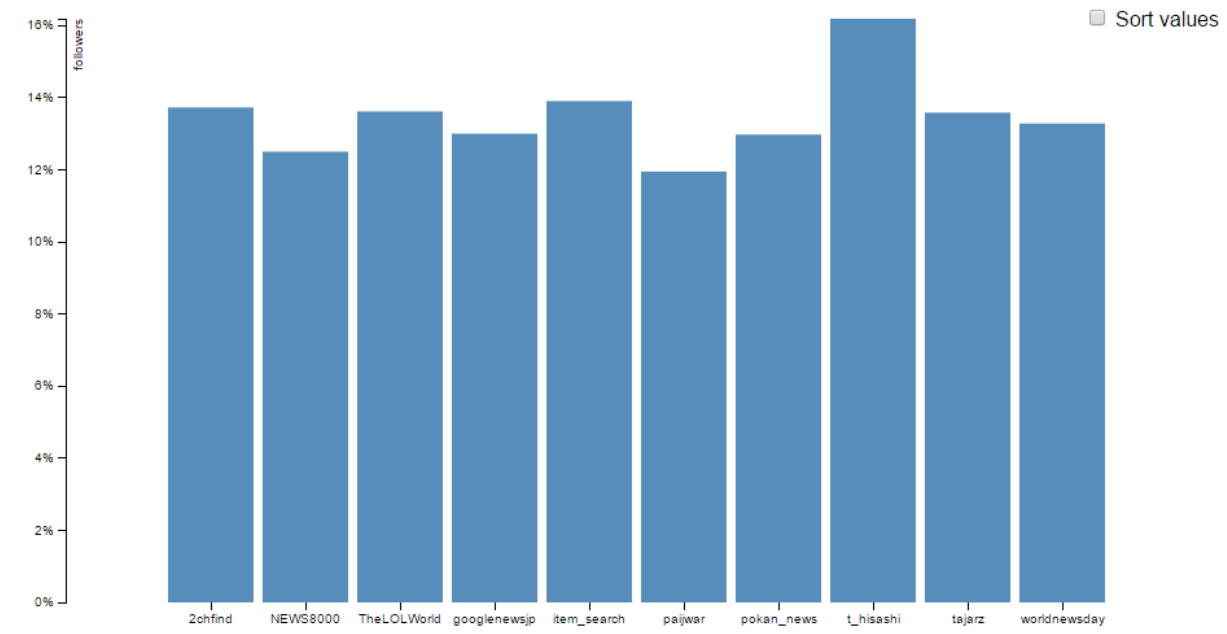
Query 4:

Here, we have matched the followers count shown by Times magazine.

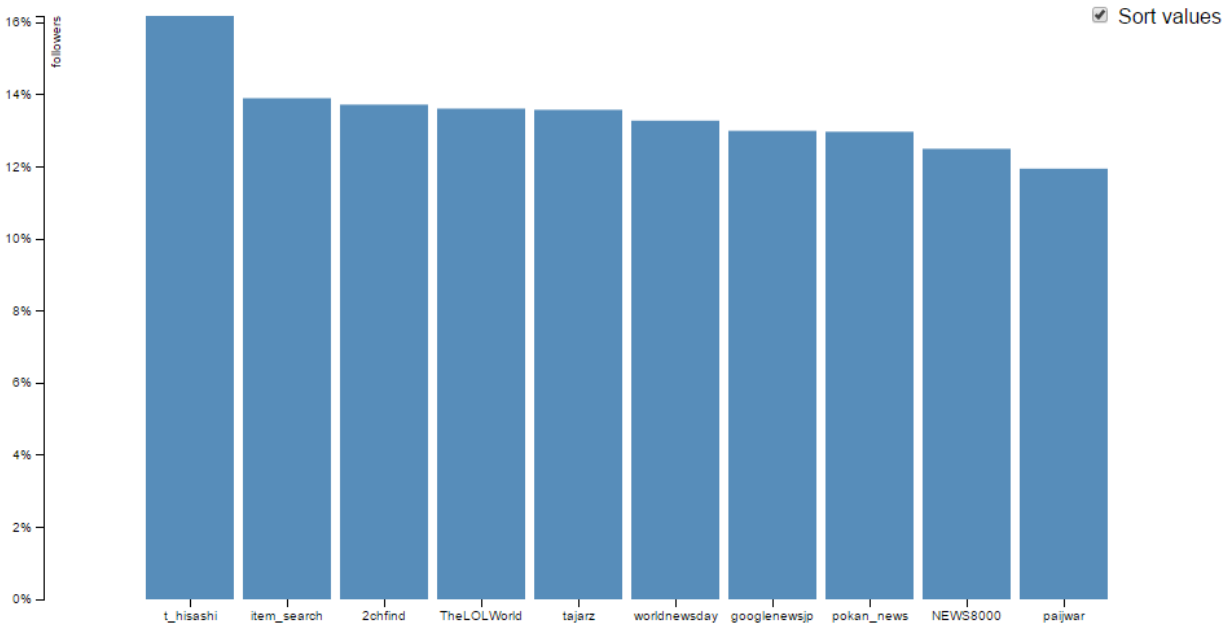


Query 5:

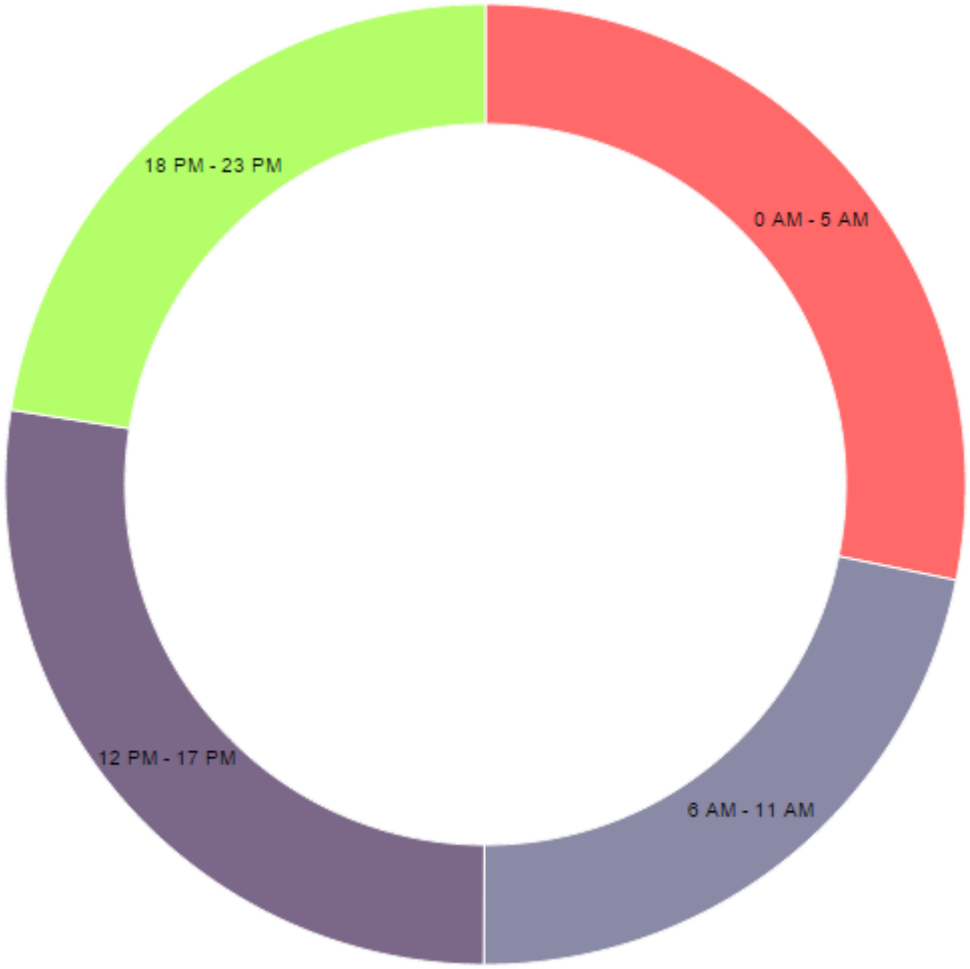
Unsorted Graph



Sorted Graph



Query 6:



Database:

Table : Tweet_info

Table : tweet_info	
	DataType
tweet_id	string,
tweet_created_at	string,
tweet_user_id	string
tweet_geo_lat	float,
tweet_geo_long	float,
tweet_place_id	string,
tweet_place_type	string,
tweet_place_name	string,
tweet_place_ctry_code	string,
tweet_place_country	string,
tweet_place_full_name	string,
tweet_possibly_sensitive	boolean,
tweet_truncated	boolean,
tweet_lang	string

Table: Tweet_text_info

Table : tweet_text_info	
	DataType
tweet_id	string,
tweet_text	string,

Table: re_tweet_text_info

Table : re_tweet_text_info	
	DataType
tweet_id	string
re_tweet_id	string
re_tweet_text	string

Table : retweet_info

Table : retweet_info	
	DataType
tweet_id	string
re_tweet_created_at	string
re_tweet_id	string
re_tweet_soruce	string
re_tweet_isTruncated	string
re_tweet_inReplyto_ScreenName	string
re_tweet_inReplyto_Status_Id	string
re_tweet_inReplyto_User_Id	string

Table2: User_info

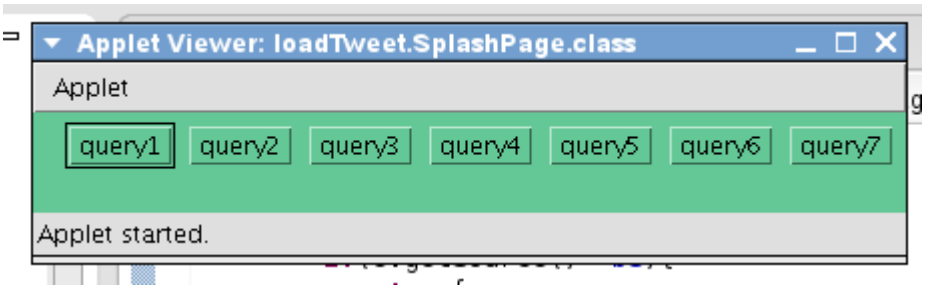
Table : user Info	
	DataType
tweet_id	string,
user_id	string,
user_name	string,
user_screen_name	string,
user_location	string,
user_protected	boolean,
user_created_at	string,
user_utc_offset	int,
user_time_zone	string,
user_geo_enabled	boolean,
user_lang	string,
user_contributors_enablec	boolean,
user_is_translator	boolean,
user_verified	boolean

Table: user_count_info

Table : user_count_info	
	DataType
user_id	string,
user_followers_count	bigInt
user_friends_count	string,
user_listed_count	string,
user_favourites_count	string,
user_statuses_count	string

Implementation:

We are using a Java applet to run all the queries. Screenshot follows,



Applet Code:

```
package loadTweet;

import java.applet.Applet;
import java.awt.Button;
import java.awt.Color;
import java.awt.FlowLayout;
import java.awt.TextField;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.*;

//import com.ibm.jsse2.t;

public class SplashPage extends Applet implements ActionListener
{
        private static final long serialVersionUID = 1L;
        Button b1, b2, b3, b4, b5, b6,b7;
        TextField t1;
```

```
public void init()
{

	Color k=new Color(100, 200, 150);
	setBackground(k);
	FlowLayout gl=new FlowLayout(PROPERTIES);
	setLayout(gl);
	b1 = new Button("query1");
	b2 = new Button("query2");
	b3 = new Button("query3");
	b4 = new Button("query4");
	b5 = new Button("query5");
	b6 = new Button("query6");
	b7 = new Button("query7");

	add(b1);
	add(b2);
	add(b3);
	add(b4);
	add(b5);
	add(b6);
	add(b7);
	b1.addActionListener(this);
	b2.addActionListener(this);
	b3.addActionListener(this);
	b4.addActionListener(this);
	b5.addActionListener(this);
	b6.addActionListener(this);
	b7.addActionListener(this);

}

@Override
public void actionPerformed(ActionEvent e) {
	// TODO Auto-generated method stub

	String [] args = null;
	if(e.getSource()==b1){
		try {
			Query1.main(args);
		}
		catch (Exception ex){}

	}
	if(e.getSource()==b2){
		try {
			Query2.main(args);
		}
		catch (Exception ex){}

	}
	if(e.getSource()==b3){
		try {
			Query3.main(args);
		}
		catch (Exception ex){}
```

```
                }
                if(e.getSource()==b4){
                        try {
                                Query4.main(args);
                        }
                        catch (Exception ex){}
                }
                if(e.getSource()==b5){
                        try {
                                Query5.main(args);
                        }
                        catch (Exception ex){}
                }
                if(e.getSource()==b6){
                        try {
                                Query6.main(args);
                        }
                        catch (Exception ex){}
                }
                if(e.getSource()==b7){
                        try {
                                Query7.main(args);
                        }
                        catch (Exception ex){}
                }
        }
}
```

System Requirement:

- Environment: IBM Info Sphere Big Insights v3.0
- NoSQL Tool: Hive
- Programming Language: Java, python
- Data Source : Twitter
- Data Size : 2. 84 GB
- Data Format : JSON, CSV
- Visualization: D3

Git Hub URL: https://github.com/hastimal/Bigdata