

# RESTful Service API Big Data

Distributed Systems and Networking – Free Project

Spring 2012-2013

Professor: Ana Aguiar

Submitted By: Po-lin Yeh, Pun nawachara Campanang, Vishal Shahane, Yu-chi Chen

## Index

1.	Introduction.....	1
1.1.	Motivation.....	1
1.2.	Objective .....	1
2.	Development and Testing Environment.....	2
2.1.	Tools used .....	2
2.2.	System architecture.....	4
3.	Implementation Detail .....	5
3.1.	CDH Implementation .....	5
3.2.	Cluster Establishment .....	7
3.3.	Flume .....	11
3.4.	Hive.....	13
3.5.	RESTful API.....	15
4.	Testing and Result.....	16
4.1.	Data Collection .....	16
4.2.	Multi-nodes Data Computation .....	16
5.	Achievement and Conclusion.....	18
5.1.	Implementation of Big Data Service .....	18
5.2.	Performance Analysis and Discussions .....	21
6.	Future Goal.....	24
7.	Reference .....	26

# 1. Introduction

## 1.1. Motivation

Big data is buzzwords nowadays and their applications are limitless. That's the main reason we selected this area. Big data solves the challenges related to managing tremendous amount of Internet data, which could not be addressed by conventional database systems. This motivated us to select the area for project and we decided to implement the project primarily based on Apache Hadoop an open source project. It satisfies all the needs for such big data processing and management, it offers one of the possible mechanisms to distribute, manage and process in the distributed fashion. This project provides different components (for example, Hbase, HDFS, mapreduce, etc.), which can be integrated seamlessly for distributed processing. We have used CDH – Cloudera's implementation of Apache Hadoop project to leverage its ability install and configure wide range of applications in clustered environment.

For the application of Big Data we have selected Big Data mainly because of its importance in marketing research nowadays. It is being used for wide scenarios such as, anticipation of share of voice for events, social media political analysis, etc. The area we have targeted is using twitter for marketing. We have come across this application during our MBA studies.

Although we have provided few use cases for how Big Data can be used with twitter data by marketing managers, our implementation could be used to extend for additional use cases and applied for various data sources.

## 1.2. Objective

We choose one of the most prosperous social networks (details in table [8]), Twitter not only covers the huge amount of Internet populations but also, and most importantly, it provides the "Streaming API" for developers to facilitate the procedure of data collection.

Twitter Company Statistics [8] (by May, 7, 2013)	Data
Total number of active registered Twitter users	554,750,000
Number of new Twitter users signing up everyday	135,000
Number of unique Twitter site visitors every month	190 million
Average number of tweets per day	58 million
Number of Twitter search engine queries every day	2.1 billion
Percent of Twitter users who use their phone to tweet	43 %
Percent of tweets that come from third party applicants	60%
Number of people that are employed by Twitter	2,500
Number of active Twitter users every month	115 million
Percent of Twitters who don't tweet but watch other people tweet	40%
Number of days it takes for 1 billion tweets	5 days
Number of tweets that happen every second	9,100

Therefore, by incorporating the data from Twitter, the project aims at providing the interface to the users who wish to retrieve and further apply the data to marketing field or service/product introductions.

Due to limitation of implementation/testing environment we decided to limit the importing of the twitter data based on keywords. Also to give near real time experience we have implemented scripts, details will be explained in subsequent sections.


Our main objective was to build generic setup, which could be adopted for various sources, and we have achieved that.

## 2. Development and Testing Environment

### 2.1. Tools used

In the project, we use several tools to implement Big Data setup based on Hadoop. All the tools are summarized below.

#### CDH [4]

 CDH stands for Cloudera's Distribution Including Apache Hadoop. CDH is an open source and widely available tools for configuring and managing Apache Hadoop. The architecture of CDH and some related engines are illustrated in the Figure 1. CDH will help users to build up and configure the clusters of HDFS which will be used as the storage of Apache Hadoop. Moreover, CDH provides web UI to visualize and simplify the administration procedure.

Therefore, in our project, we adopt CDH as the configuration and management tool for distributed file system; thus CDH can facilitate the way we interact with the data we collect from Twitter and store in the cluster.

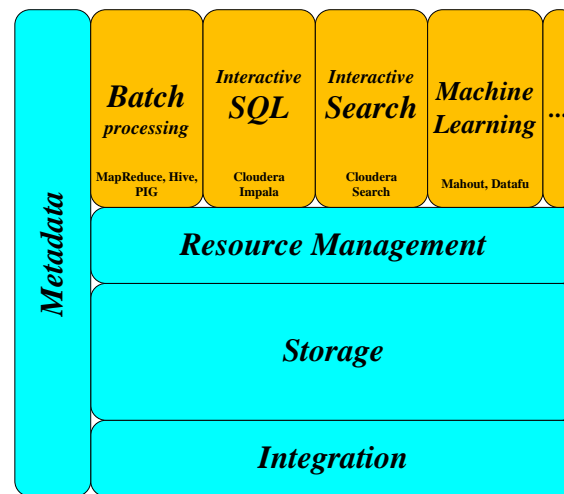


Figure 1 – CDH Architecture

### Hadoop [2]



The Apache Hadoop architecture is dedicated to handle massive data, which could be terabytes or petabytes, far beyond the conventional database system can deal with. The Hadoop also offers as open-source software for reliable and distributed data processing. Its high scalability also provides user with huge opportunities for gathering and processing data in the distributed fashion with hundreds of thousands of nodes. The current Hadoop system is composed of Hadoop kernel, MapReduce, HDFS and some projects such as Mahout, Hive and HBase. In a cluster, it will include a master node and multiple slaves (other nodes rather than mater node); the master node is composed of a JobTracker, a TaskTracker, a NameNode and a DataNode while a slave is composed of a DataNode and a TaskTracker. In our project, we group six nodes in a single cluster. One of them is the master (linus100) and the rest of them (linus50, linus60, linus70, linus80 and linus90) are slaves.

### Flume [1]



Apache Flume is the tool, which collects and parses user-defined data from the sources and delivers to the Hadoop Distributed File System (HDFS) of Apache Hadoop. Therefore, we incorporate Flume into our

application to parse Twitter data, which relates to big data industry, and put into HDFS. Due to Flume is one of the core components in our application, we will discuss Flume in detail in the later section.

### **Hive [3]**



Apache Hive is one of the projects related to the Apache Hadoop. It provides the interface to display data, query data and analyze data in HDFS by using a SQL-like language named HiveQL. In addition, Hive execute MapReduce algorithm every time when users want to retrieve data via HiveQL specified datasets. Therefore, we adopted Hive into our system to interact with our Twitter data in the HDFS; also, we use HiveQL to provide the service for the users as well as play the role of interface between the users and the system. Due to Hive is also another core component in our application, we will discuss Hive in detail in a dedicated section later.

### **SerDe**

SerDe stands for the Serializer and Deserializer and it is also the interface that allows Hive to retrieve (read and write) data from HDFS. Due to that Hive doesn't know (and no need to know) the format of the data which was stored in the HDFS by various mechanisms user chose, it's essential to use SerDe interface that facilitates the users to interact with the data in HDFS via Hive. Since our project and the application highly rely on Hive, the usage of SerDe is essential for us to achieve the objective of our application. Some details of SerDe interface will be provided in the later section.

## **2.2. System architecture**

To build up the infrastructure for analyzing the Twitter data and then providing service, we have selected Apache Flume, Apache HDFS, and Apache Hive to design an end-to-end data pipeline. The whole architecture is shown in Figure 2.

Flume was configured to communicate with the Twitter Streaming API for parsing the tweets as per the keywords defined. After that it writes data into the predefined HDFS. Here, we only use Master (192.168.109.100) to capture the data from Twitter source. In large setup multiple nodes and masters could be used, but configuration will be similar. As captured data is stored into HDFS, it will be accessible from all the nodes in the cluster.

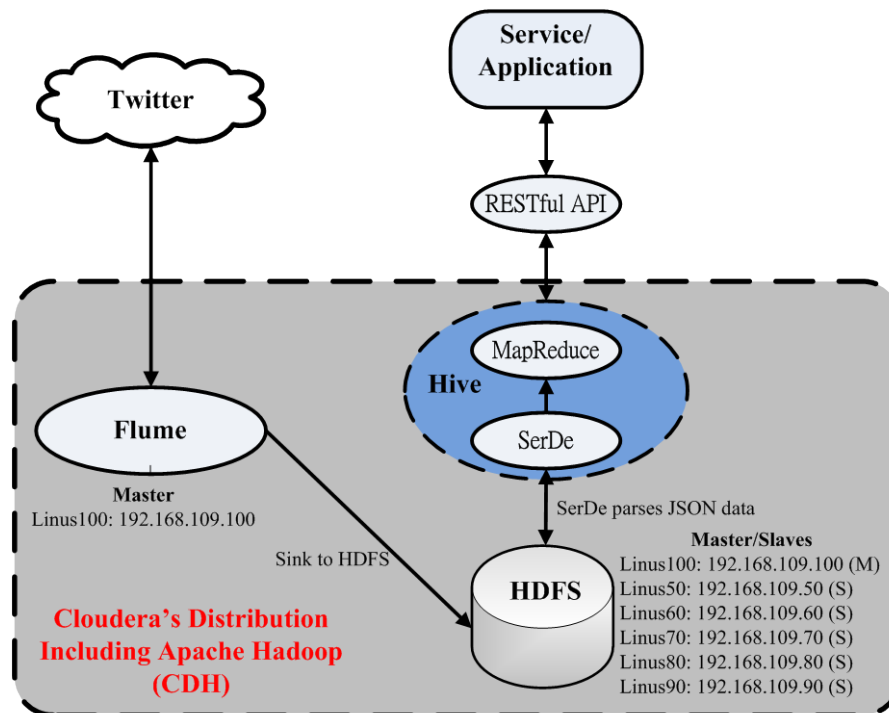


Figure 2 - System Architecture

When user accesses the service/application by keying in the query, the related command (HQL – Hive Query Language) is executed on Hive by using shell script in the background for getting the expected data. Resulting data is returned to service/application in JSON format, which can be parsed and displayed in expected format later.

Hive executes MapReduce jobs in the background to get required data. Number of nodes selected by Hadoop to execute the job depends on amount of data. It also uses SerDe to parse the twitter data stored in HDFS.

## 3. Implementation Detail

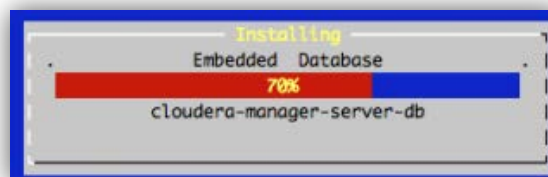
### 3.1. CDH Implementation

As mentioned before, CDH is the tool we use to manage and configure the Apache Hadoop and the big data distributed file system. Before constructing the clusters and the file system, we first installed the Cloudera Manager and bootstrap the CDH cluster, which we will describe in the next section. In this project, we use the Cloudera Manager with version "Cloudera Enterprise 4.6.0". Once we execute the install package of the Cloudera Manager Installer, we will see the following

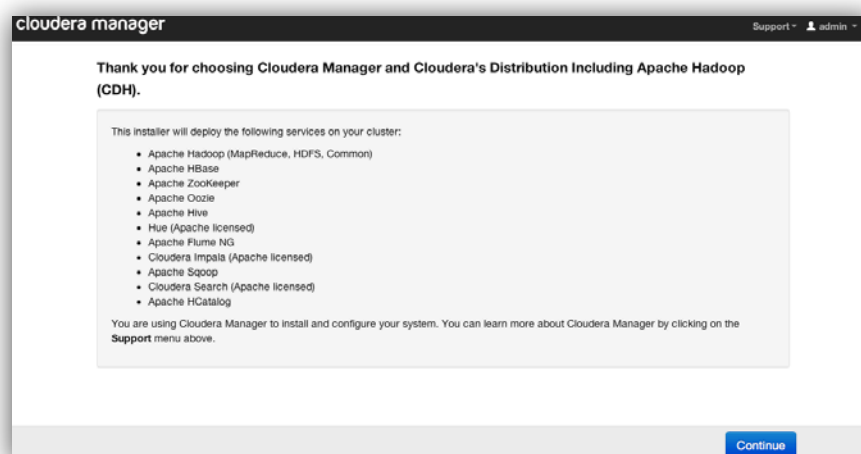
installation instruction:



Then, we installed the Cloudera Manager and downloaded packages for Apache Hadoop environment:



After successfully installing the Cloudera Manager, and accessing the manager service from web, we see the following instruction that indicates the packages the Cloudera Manager Server is going to download, as shown in the following figure, on all the nodes/machines in the clusters:





Before installing these tools, the Cloudera Manager needs to know the machines in the cluster. Therefore, after pressing the continue button, the installer will ask the machines we are going to use in the cluster. Thus, we need to assign the machines for the cluster and setup the cluster.

### 3.2.Cluster Establishment

In our implementation, we use six machines in the cluster. If all the machines are in the normal operation status, the corresponding IP addresses, expanded query and the host names will be listed as shown in following figure:

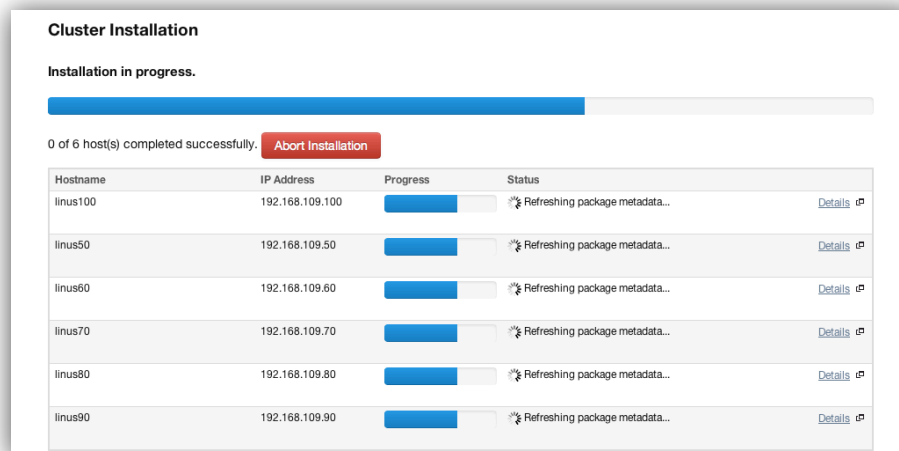
The screenshot shows the Cloudera Manager web interface. At the top, it says "Specify hosts for your CDH cluster installation." Below this, there is a hint: "Cloudera recommends including Cloudera Manager server's host because it is often used for the Cloudera Management Service, and because this will enable health monitoring for that host. Hint: Search for hostnames and/or IP addresses using patterns." A search bar shows "6 hosts scanned, 6 running SSH." Below the search bar is a table with columns: Expanded Query, Hostname (FQDN), IP Address, Currently Managed, and Result. The table lists six hosts: linus100, linus50, linus60, linus70, linus80, and linus90. All hosts are currently managed and show a "Host ready" status. At the bottom, there are "Back" and "Continue" buttons.

Expanded Query	Hostname (FQDN)	IP Address	Currently Managed	Result
<input checked="" type="checkbox"/> linus100	linus100	192.168.109.100	No	✓ Host ready: 0 ms response time.
<input checked="" type="checkbox"/> linus50	linus50	192.168.109.50	No	✓ Host ready: 1 ms response time.
<input checked="" type="checkbox"/> linus60	linus60	192.168.109.60	No	✓ Host ready: 1 ms response time.
<input checked="" type="checkbox"/> linus70	linus70	192.168.109.70	No	✓ Host ready: 0 ms response time.
<input checked="" type="checkbox"/> linus80	linus80	192.168.109.80	No	✓ Host ready: 0 ms response time.
<input checked="" type="checkbox"/> linus90	linus90	192.168.109.90	No	✓ Host ready: 0 ms response time.

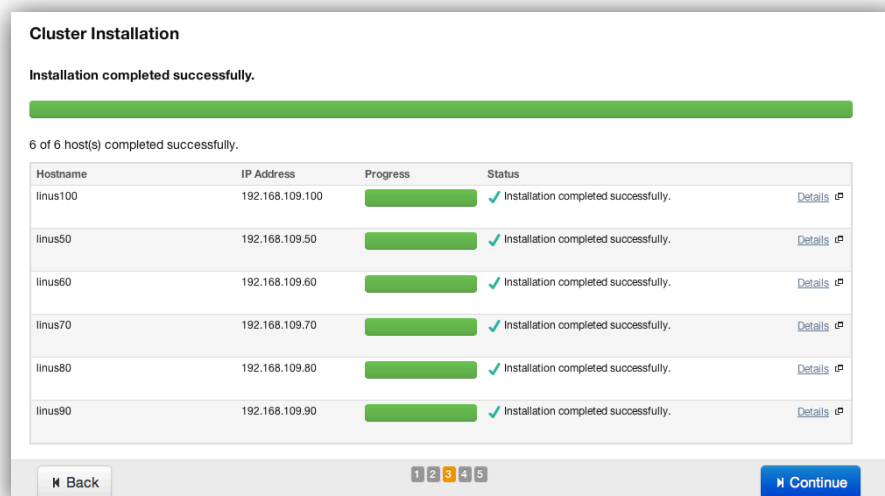
Before installing the cluster, the system needs the SSH login credential information. Here, we simply use the authentication of root as shown in following figure:

The screenshot shows the Cloudera Manager web interface for providing SSH login credentials. It asks for "Root access to your hosts is required to install the Cloudera packages. This installer will connect to your hosts via SSH and log in either directly as root or as another user with password-less sudo/pbrun privileges to become root." The "Login to all hosts as:" section has "root" selected. The "Authentication Method:" section has "All hosts accept same password" selected. There are fields for "Enter Password:" and "Confirm Password:". The "SSH Port:" is set to 22. The "Number of simultaneous installations:" is set to 10. A note at the bottom says: "(Running a large number of installations at once can consume large amounts of network bandwidth and other system resources)".

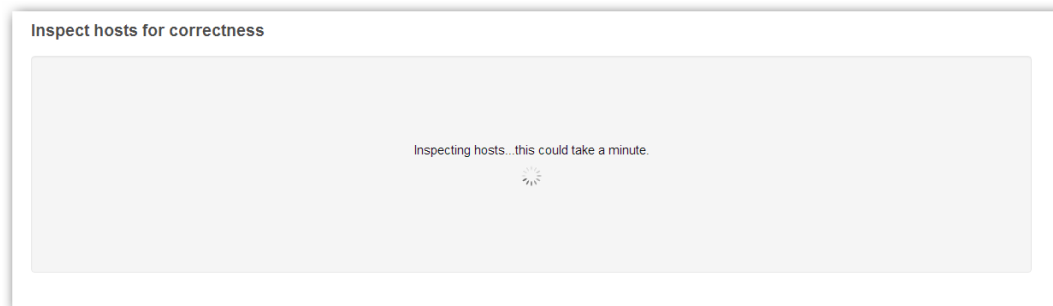
Then, the Installer will start to install and configure the six machines in the cluster:



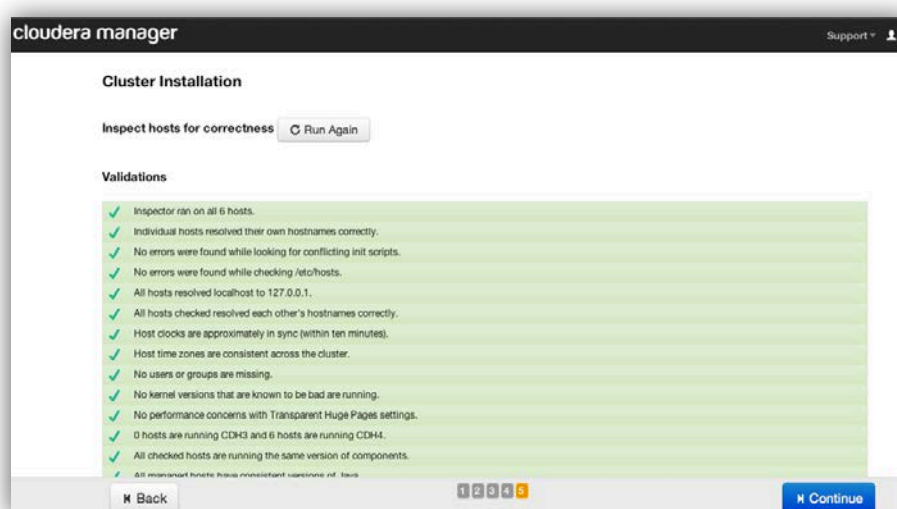
When the installation completes, we will see the UI as shown in the following figure:



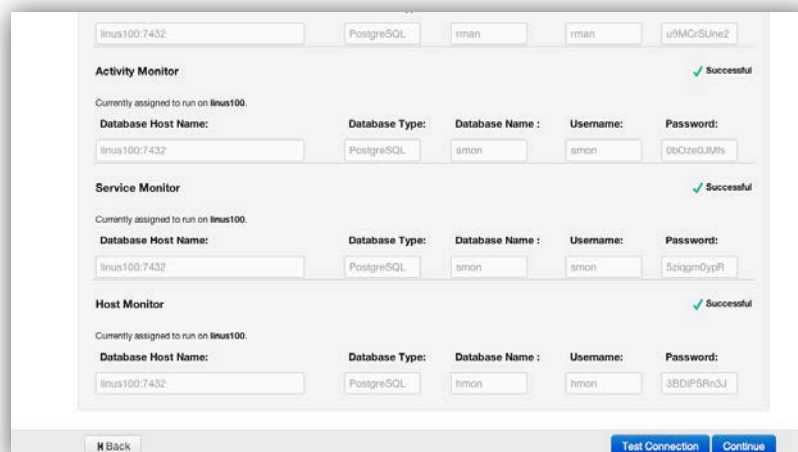
Then the installer will check installations on all the machines. The process will check the item such as network configuration, system time, SSH, HDFS configuration....:



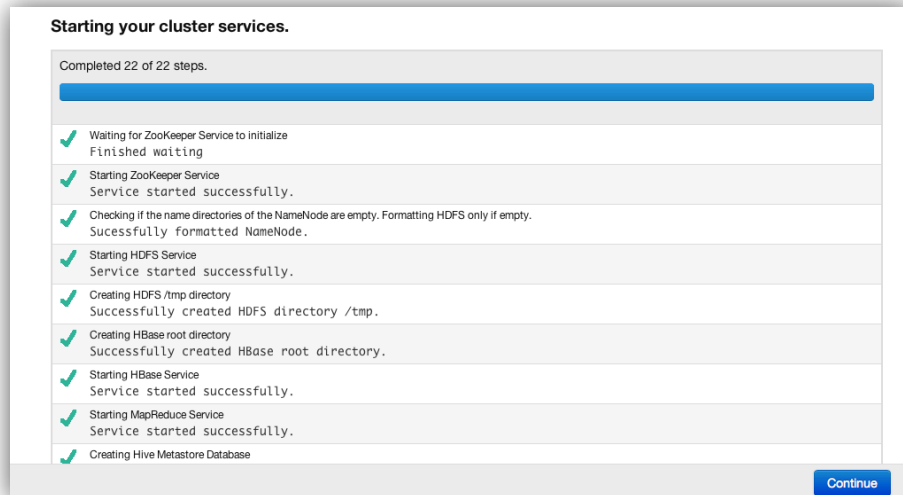
The results of the checking process of our system are illustrated in the following figure:



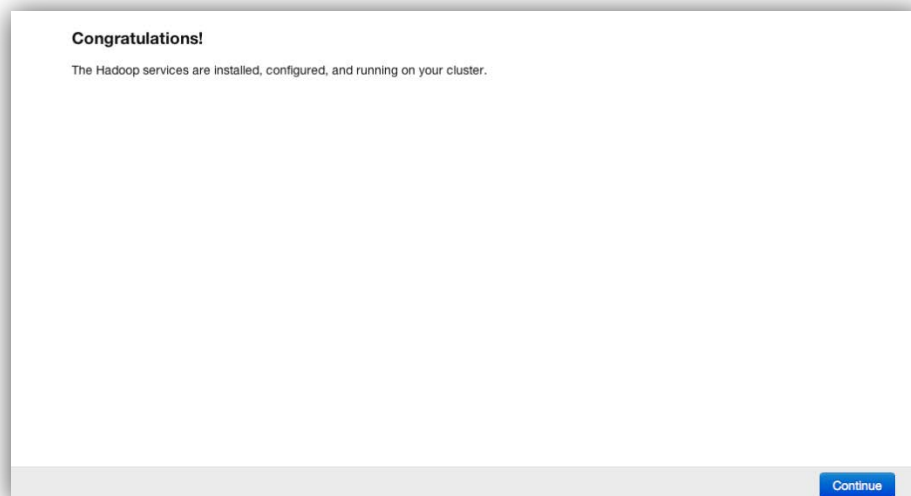
After all the processes are validated, the installer will show the current configurations and system setup. Currently, we use “linus100” as the master node in the cluster which will be responsible for the following services as shown in the figure:



Finally, after checking and configuring all the necessary information for the machines and the cluster following the above steps, the installer will start the cluster services step by step as demonstrated in the following figure:























If the process is successful, the following figure will show up:










The figure told us that the CDH is successfully installed and our cluster service of HDFS is working on the assigned machines now. The current configuration of services and summary of roles assigned to machines in cluster are shown in following two figures:

## All Services

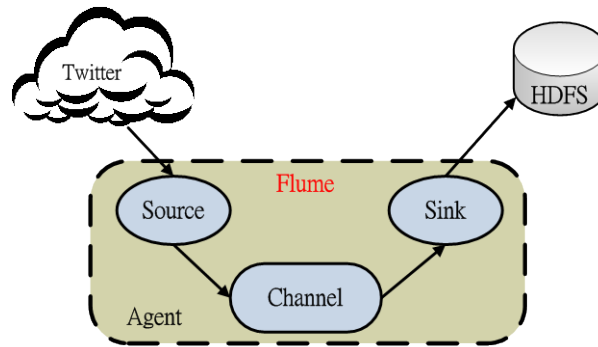
### Cluster 1 - CDH4

Name	Status	Role Counts
 flume1	 Concerning Health	<a href="#">2 Agents</a>
 hbase1	 Bad Health	<a href="#">6 RegionServers, 1 Master</a>
 hdfs1	 Bad Health	<a href="#">1 SecondaryNameNode, 1 NameNode, 1 Balancer, 6 DataNodes</a>
 hive1	 Good Health	<a href="#">1 Hive Metastore Server, 1 WebHCat Server, 6 Gateways</a>
 hue1	 Good Health	<a href="#">1 Beeswax Server, 1 Hue Server</a>
 impala1	 Bad Health	<a href="#">6 Impala Daemons, 1 Impala StateStore Daemon</a>
 mapreduce1	 Bad Health	<a href="#">1 JobTracker, 6 TaskTrackers</a>
 oozie1	 Good Health	<a href="#">1 Oozie Server</a>
 sqoop1	 Good Health	<a href="#">1 Sqoop Server</a>
 zookeeper1	 Good Health	<a href="#">1 Server</a>

<input type="checkbox"/>	Name	IP	Rack	CDH Version	Cluster	Roles	Status
<input type="checkbox"/>	Any Na	Any IP	Any R	All	All	All	All
<input type="checkbox"/>	linus100	192.168.109.100	/default	CDH4	Cluster 1 - CDH4	► 25 Role(s)	 Good Health
<input type="checkbox"/>	linus40	192.168.109.40	/default	CDH4	Cluster 1 - CDH4		 Good Health
<input type="checkbox"/>	linus50	192.168.109.50	/default	CDH4	Cluster 1 - CDH4	► 5 Role(s)	 Bad Health
<input type="checkbox"/>	linus60	192.168.109.60	/default	CDH4	Cluster 1 - CDH4	► 5 Role(s)	 Bad Health
<input type="checkbox"/>	linus70	192.168.109.70	/default	CDH4	Cluster 1 - CDH4	► 5 Role(s)	 Bad Health
<input type="checkbox"/>	linus80	192.168.109.80	/default	CDH4	Cluster 1 - CDH4	► 6 Role(s)	 Concerning Health
<input type="checkbox"/>	linus90	192.168.109.90	/default	CDH4	Cluster 1 - CDH4	► 5 Role(s)	 Good Health

## 3.3.Flume

In this project, we use Apache Flume as our main tool to bring data from twitter streaming API into HDFS. The piece of data we collected from Twitter source will be packed into events at first, and then they will be processed by three primary structures: sources, channels, and sinks (shown in the following figure) before forwarding to HDFS.



## Source

The source is from where Flume receives the data. The main function of Source in Flume is to gather the discrete data and pack them into individual event. Therefore, after that, channel could process the event as a batch.

The configuration of source part regarding the Flume Agent can be seen in following table:

Flume Agent- Source Part Configuration	
<b>TwitterAgent.sources</b>	Twitter
<b>TwitterAgent.sources.Twitter.type</b>	com.cloudera.flume.source.TwitterSource
<b>TwitterAgent.sources.Twitter.channels</b>	MemChannel
<b>TwitterAgent.sources.Twitter.consumerKey</b>	[Required]
<b>TwitterAgent.sources.Twitter.consumerSecret</b>	[Required]
<b>TwitterAgent.sources.Twitter.accessToken</b>	[Required]
<b>TwitterAgent.sources.Twitter.accessTokenSecret</b>	[Required]
<b>TwitterAgent.sources.Twitter.keywords</b>	hadoop, big data, analytics, bigdata, cloudera, data science, data scientist, business intelligence, mapreduce, data warehouse, data warehousing, mahout, hbase, nosql, newsq, businessintelligence, cloudcomputing

In order to use Twitter Streaming API, we register the consumerkey, consumerSecret, accessToken and accessTokenSecret and then apply them in the configuration part. In addition, to parse the appropriate tweets from Twitter, we set the filtered words related to “Big Data” in the field of keywords.

## Channel

Channel is acted as an intermediary between Source and Sink. Events are put by source and later removed from channel by Sink. In this project, we only use master

machine to run Flume to acquire the data from Twitter source. Therefore, we only configure one channel although in Flume dataflows can actually support multiple channels. Following table is the configuration regarding the channel part.

Flume Agent- Channel Part Configuration	
TwitterAgent.channels	MemChannel
TwitterAgent.channels.MemChannel.type	memory
TwitterAgent.channels.MemChannel.capacity	10000
TwitterAgent.channels.MemChannel.transactionCapacity	100

Here, we set the “type” equals to “memory”, which means the events will be stored in memory in the channel before they are written into sink. With doing so, we expect the dataflows can achieve high throughput considering that the possibility of losing data due to agent failure is low in our case.

### Sink

The final part of the Flume is Sink which takes events and stores them to configured location in HDFS. The configuration is shown in below:

Flume Agent- Sink Part Configuration	
TwitterAgent.sinks	HDFS
TwitterAgent.sinks.HDFS.channel	MemChannel
TwitterAgent.sinks.HDFS.type	hdfs
TwitterAgent.sinks.HDFS.hdfs.path	hdfs://192.168.109.100:8020/user/flume/tweets/%Y/%m/%d/%H/
TwitterAgent.sinks.HDFS.hdfs.fileType	DataStream
TwitterAgent.sinks.HDFS.hdfs.writeFormat	Text
TwitterAgent.sinks.HDFS.hdfs.batchSize	1000
TwitterAgent.sinks.HDFS.hdfs.rollSize	0
TwitterAgent.sinks.HDFS.hdfs.rollCount	10000
TwitterAgent.sinks.HDFS.hdfs.rollInterval	600

As we can see from the above table, we set the path to the IP address of the master machine (192.168.109.100). In addition, we set the file size to equal to 10000 tweets (defined in the field of rollCount) and 10 minutes (600 here) waiting time before rolling the current file. Finally, every 10 minutes the Flume will output one new file and put it into HDFS. Due to structure of sink path, it creates the individual directory for each hour.

## 3.4.Hive

Cloud manager was used to deploy the Hive service. As amount of data was not enormous we just deployed the Hive metadata server on Linus100. It stores all the metadata information about Hive database.



Considering that the data Flume parses from the Twitter source is JSON format, we select Hive, which is very effective for dealing with semi-structured or unstructured data, as our main tool to process the data collected from Twitter. Moreover, to support real time operation and get up-to-date data, we establish the **automatic scripts**, which will import data from HDFS into Hive table every hour. Loading the data into hive data means it just updates metadata information about location of file residing on HDFS. To control the execution time we created the table with limited information and not all information from Twitter data, description of table (tweets) is shown in below figure.

```
hive> describe tweets;
OK
id          bigint   from deserializer
created_at  string   from deserializer
source      string   from deserializer
favorited   boolean  from deserializer
retweet_count int      from deserializer
retweeted_status struct<text:string,user:struct<screen_name:string,name:string>> from deserializer
entities    struct<urls:array<struct<expanded_url:string>>,user_mentions:array<struct<screen_name:string,name:string>>,hashtags:array<struct<text:string>>> from deserializer
text        string   from deserializer
user        struct<screen_name:string,name:string,friends_count:int,followers_count:int,statuses_count:int,verified:boolean,utc_offset:int,time_zone:string> from deserializer
in_reply_to_screen_name string   from deserializer
datehour    int
```

The other advantage of using Hive is that we can easily interact with it by using SQL-like statements (HQL – Hive Query Language). More than that, when any query runs in Hive is actually executed as the sequence of MapReduce jobs. Therefore, Hive is very suitable for dealing with massive data sets. Following is one example we query Hive to see top 15 accounts with number of followers.

```
SELECT
    user.screen_name,           // Select the field of "screen_name" in the twitter account
    max(user.followers_count)   // based on number of followers in the field of "followers_count"
    t                           // and stored the result in "t"
FROM tweets
GROUP BY user.screen_name
ORDER BY t DESC                // Show Top 15 accounts which have the most followers
LIMIT 15;                     // and show the list in descending order
```

The result is as follows:



Rank	screen_name	#followers	Rank	screen_name	#followers
1	nytimes	8649179	9	om	1345831
2	hootsuite	5036660	10	RealTonyRocha	1299845
3	nfl	4715463	11	SAI	1282956
4	TechCrunch	2802977	12	johncusack	1257021
5	wired	2042481	13	ForbesTech	860890
6	guardiantech	2033699	14	bhogleharsha	811145
7	timoreilly	1706688	15	FastCompany	763604
8	sacca	1412093			

The automation script to load data into Hive or to update its metadata information is scheduled in CRON to execute every hour. The entry is shown in below figure:

```
root@linus100:~# crontab -l | grep data
15 00-23 * * * /var/lib/hadoop-hdfs/data-load.sh
```

### 3.5.RESTful API

We have built the RESTful API, which could be integrated with external services or applications to gather required data from our implementation. Considering the test setup we have provided limited capability to get the data, but this setup can be extended to get any kind twitter data or if source is changed other services could be provided.

To implement the RESTful API we have used Apache2 and Slim framework on Ubuntu. This API currently accepts two variables: Type of data and number of top rows belonging to that kind of data. Following figure shows the query for RESTful API.

[192.168.109.100/slim/?fn=tweets&cn=8](http://192.168.109.100/slim/?fn=tweets&cn=8)

In the above link tweets is the value of the first variable and 8 is the value of second variable. API is nothing but PHP interface, which executes respective shell script on web server, which in turn executes Hive queries to get required data. Usage of this API is demonstrated with test application in section 5.

The result is provided in JSON format, example result for above query is shown below:

```
"TwitterAccount\tTweetsCount\nPageViral\t3457\nTruOptik\t1847\ncontextualdata\t566\nbigdatameetups\t461\nMinneapolisPR\t343\nwebsite_metrics\t327\nCWBadger12\t238\nBITechWatch\t238\n"
```

## 4. Testing and Result

### 4.1. Data Collection

The data captured from Twitter source (represented as JSON blobs) contains a myriad of useful information. However, in this project, we offer six services which will only use parts of the information in the data fields. Thus, we focus on following fields, which are relevant to the services we provide to the users through our system; also, the following table explains the meanings of the fields we use:

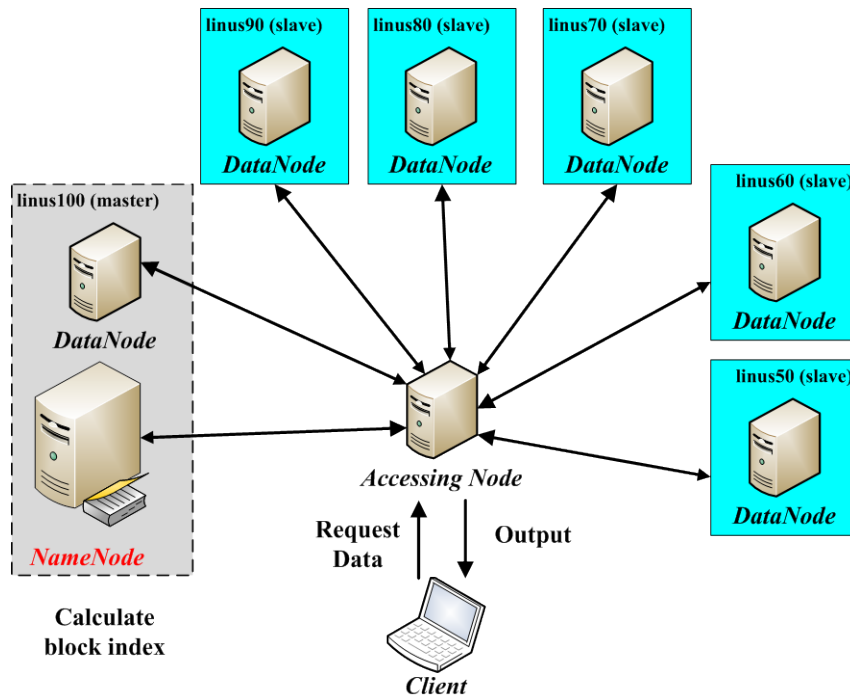
Field Name	Description
<b>entities.hashtags</b>	<ul style="list-style-type: none"><li>➤ A prefixed phrase which provide the means of grouping the messages and allow the Twitter subscribers easily to find the interested information just by searching the Hashtags.</li><li>➤ With this information, we can know which topics related to the big data industry are most common.</li></ul>
<b>created_at</b>	➤ Show the posting time of that specific tweet.
<b>user.screen_name</b>	➤ Show the account of Twitter subscriber who posts the tweet related to the keyword we are targeting.
<b>user.time_zone</b>	➤ Show the Twitter subscriber's belonging time zone.
<b>user.followers_count</b>	➤ Show how many followers follow that specific subscriber.
<b>user.friends_count</b>	➤ Show how many friends that targeted subscriber has.
<b>user.statuses_count</b>	➤ Illustrate how many tweets posted by that specific subscriber.

Combining the information of these fields, we are able to retrieve and vivify the results which response the requests of users.

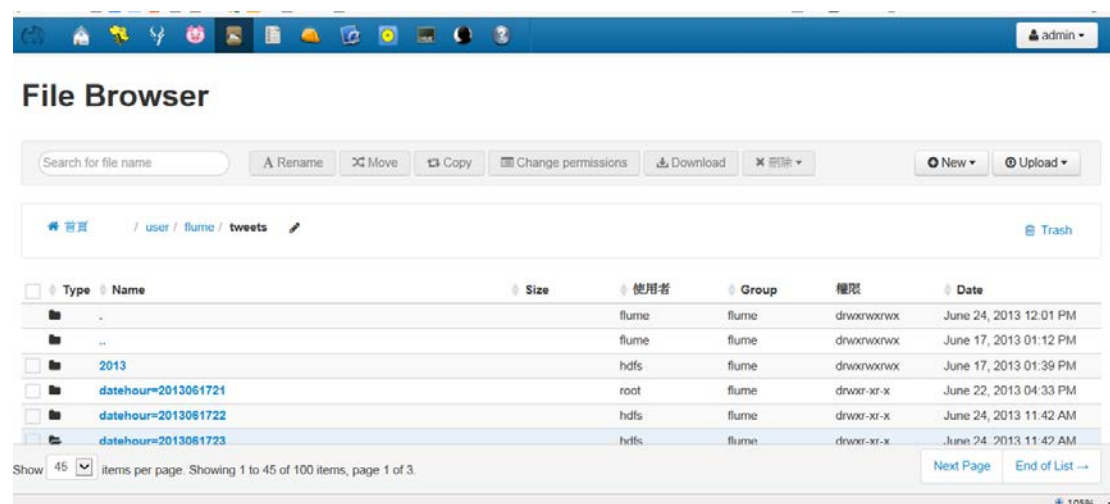
### 4.2. Multi-nodes Data Computation

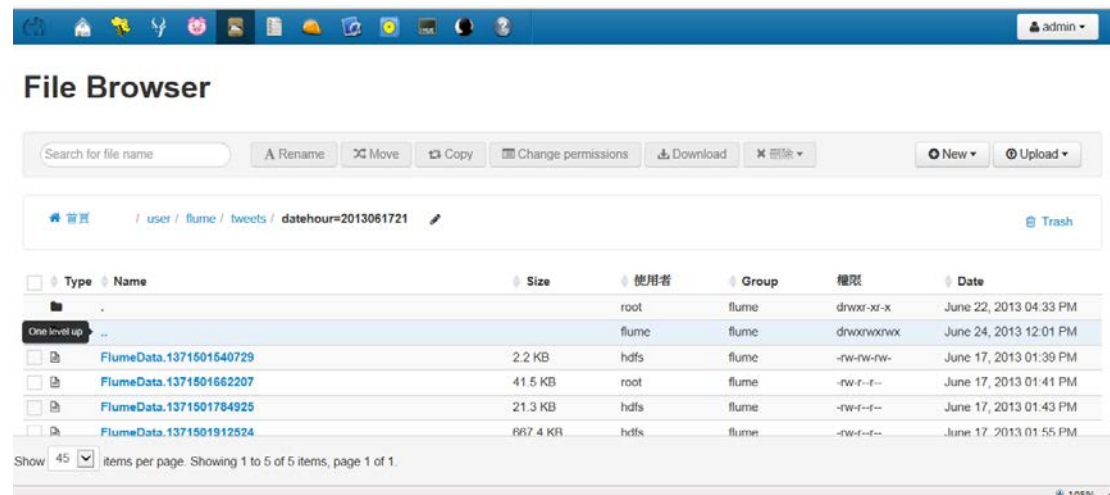
As mentioned before, we use six machines in our system. Also, we configure these machines as the multi-nodes in single cluster. Thus, one of them is the

master (“linus100”) and the rest five machines are slaves (“linus50,” “linus60,” “linus70,” “linus80” and “linus90”). The machine “linus100” (master node) is composed of a JobTracker, a TaskTracker, a NameNode and a DataNode while each slave is composed of a DataNode and a TaskTracker. The whole infrastructure regarding these six machines is shown in the following figure.

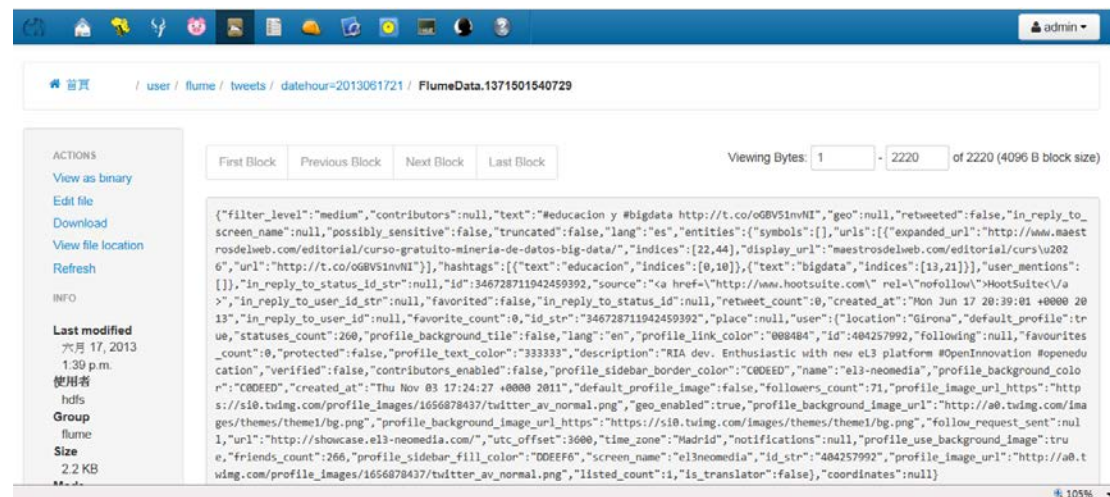


Following two figures show the file directory in the HDFS.





The file blocks parsed from Twitter source in the HDFS are shown as following figure:



## 5. Achievement and Conclusion

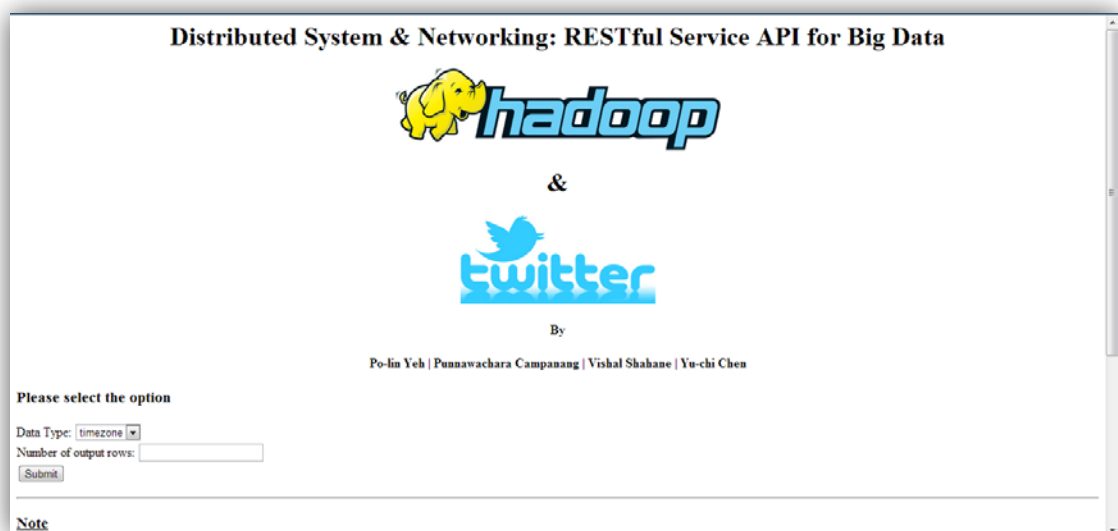
### 5.1. Implementation of Big Data Service

In this project, we provide six service options for the users who are interested in the big data industry. The six service options are:

- Top popular **time-zones** for big data industry
- Top **hashtags** for big data industry
- Top **user accounts** which related to the big data industry with **maximum followers**

- Top **user accounts** which related to the big data industry with **maximum friends**
- Top **user accounts** who have generated **maximum tweets**
- Top **user accounts** who are generating **maximum tweets for targeted keywords** in collection period and their **overall tweets, followers count and friends count**

We provide the web UI (interface) for users to interact with the system. The web UI will look like the following figure:



In the service option selection columns, the users can choose the service in the “Data Type” drop-down list as shown in the following figure. In addition, the user can also choose the number of output data rows:

**Please select the option**

Data Type:

Number of output rows:

At the bottom of the web UI, there is a note for explaining the corresponding service option in the “Data Type” column:

**Note**

**Data Type**

1. **timezone:** To get timezones for number of tweets.
2. **hashtag:** To get hashtags for number of tweets.
3. **follower:** To get user accounts with maximum followers (with followers count).
4. **friends:** To get user accounts with maximum friends (with friends count).
5. **tweets:** To get user accounts who are generating maximum tweets for targeted keywords in collection period.
6. **tweetsall:** To get user accounts who are generating maximum tweets for targeted keywords in collection period and their overall tweets, followers count and friends count.

**Number of output rows:** Rank the results from the top to the number you input

For example, if we want to understand the most active 10 user accounts that generated maximum tweets for the targeted keywords during the collection period, we could then set the “Data Type” column as “tweets” and put 10 in the “Number of output rows” column as shown in the following figure:

**Please select the option**

Data Type:

Number of output rows:

After finishing all the columns, the user should click on the “Submit” button to send out the request to the system. By doing so, the system will retrieve the data based on the user-defined query; thus, the output of the system would be a sting of characters. To enhance the readability, the interface we provide will reformat the output results and put them into a table as illustrate in the following figure:



TwitterAccount	TweetsCount
PageViral	3405
TruOptik	1824
contextualdata	566
bigdatameetups	461
MinneapolisPR	337
website_metrics	321
BITechWatch	236
CWBadger12	234
jkozak84	220
jg21	206

Therefore, the user would realize that the most active twitter account “PageViral” generated 3405 tweets which related to our targeted keywords during the collection period.

The information could be extremely helpful and make the significant contribution in several areas. For example, David Kirkpatrick [6] provided some instructions and ways to adopt the information from Twitter data in forecasting the trends and sentiment and identifying the influencers in marketing areas.

## 5.2. Performance Analysis and Discussions

In this section, we evaluate the performances of the system based on the number of nodes used in computation and the time of obtaining the results from the system.

The first analysis evaluates the performance based on the different number of nodes we used in the computation. The dominance part of the whole computation procedure is the MapReduce mechanism which is implemented in the Hive. Hence, we analyze the performance of the system and the services based on measuring the computational period of MapReduce process.

In the performance analysis, we adopt one of the possible services based on the Twitter data we gathered: “What are the top 15 popular hashtags in the big data industry on the Twitter?” One of the critical components which affect the

computation time of MapReduce is the number of nodes used in the process; thus, we vary the number of the nodes used in the example application to test its performance. In the following table, we first set the number of node as 1 and retrieve the data from the HDFS via Hive:

```
hive> set mapred.reduce.tasks=1;

hive> SELECT LOWER(hashtags.text), COUNT(*) AS total_count FROM tweets
LATERAL VIEW EXPLODE(entities.hashtags) t1 AS hashtags GROUP BY
LOWER(hashtags.text) ORDER BY total_count DESC LIMIT 15;
```

After obtaining the results, we could also learn the time CPU spent on the MapReduce task by using one node as following:

```
Total MapReduce CPU Time Spent: 57 seconds 760 msec
```

Here we set the second test as the performance of using three nodes in the MapReduce task. It's noted that the data we retrieve should be the same. Therefore, the comments of the test would be like the following:

```
hive> set mapred.reduce.tasks=3;

hive> SELECT LOWER(hashtags.text), COUNT(*) AS total_count FROM tweets
LATERAL VIEW EXPLODE(entities.hashtags) t1 AS hashtags GROUP BY
LOWER(hashtags.text) ORDER BY total_count DESC LIMIT 15;
```

Then, we could learn the time CPU spent on the MapReduce task by using three nodes as following:

```
Total MapReduce CPU Time Spent: 1 minutes 4 seconds 500 msec
```

Similarly, we set the third test as the performance of using four nodes in the MapReduce task. Therefore, the comments of the test would be like the following:

```
hive> set mapred.reduce.tasks=4;

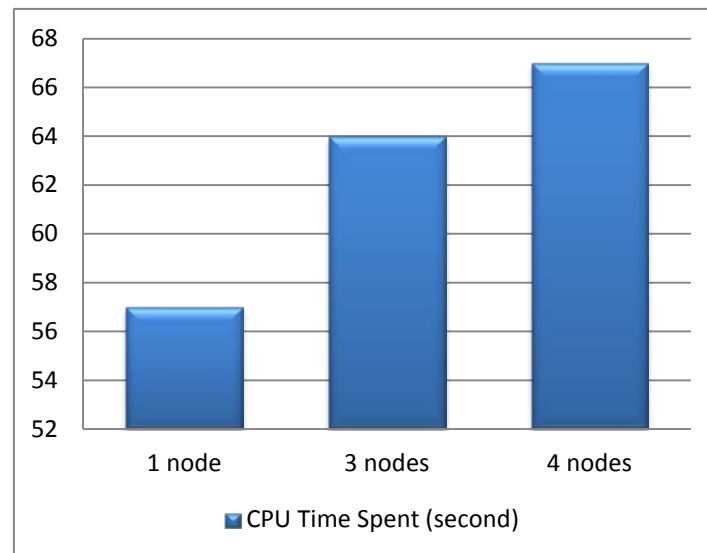
hive> SELECT LOWER(hashtags.text), COUNT(*) AS total_count FROM tweets
LATERAL VIEW EXPLODE(entities.hashtags) t1 AS hashtags GROUP BY
LOWER(hashtags.text) ORDER BY total_count DESC LIMIT 15;
```



Then, we could learn the time CPU spent on the MapReduce task by using four nodes as following:

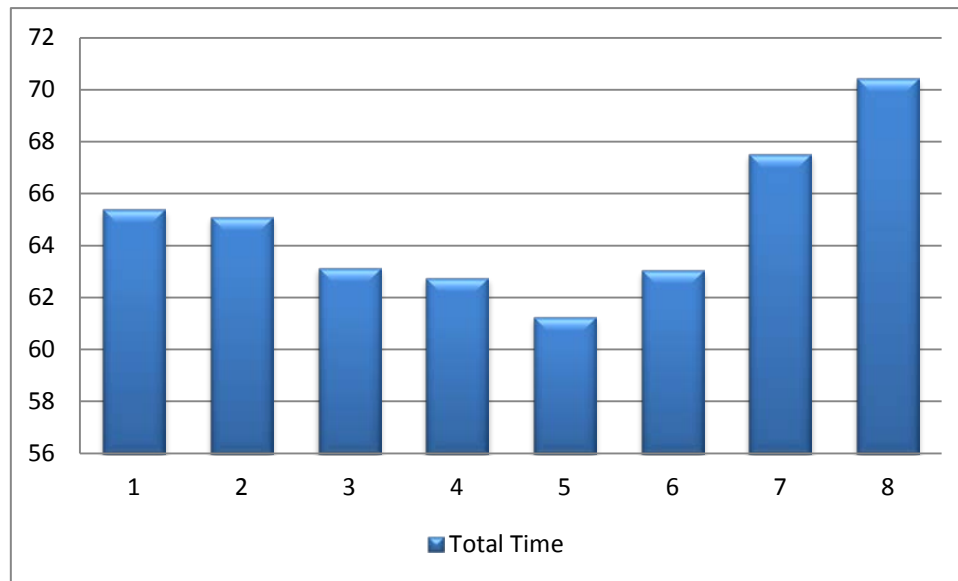
Total MapReduce CPU Time Spent: 1 minutes 7 seconds 910 msec

We can then obtain the performance of the MapReduce tasks under the situation of varying the number of nodes used in the system as the following figure:



Theoretically speaking, the computation time for CPU should decrease along with the number of nodes involved in the system since the procedure is distributed. However, in the above figure we realize that the CPU time spent increased. The reason which explains the phenomenon is that the amount of data we used in the system is relatively small compared with the terabytes or petabytes, thus, the time for allocating the computation units to nodes and retrieving respective data is dominant in whole process, while the influence of reduced time by using distributed computing is too subtle to demonstrate the benefit of distributed computing. Along with the increasing amount of the data, the benefits of distributed computing would be gradually dominant in the whole process of the system; thus, the results of CPU time spent would be reduced significantly with increase in nodes.

The second performance test is to measure the time it takes to obtain the results. In this evaluation, we use the same service option to retrieve the results eight times. The results are provided in the following figure:



From the results we can learn that even the same service option will take different time to retrieve the same results. The reason of the variation is mainly caused by the random allocation and retrieving data of MapReduce procedure. Therefore, even we execute the same query, the total time spent on allocating and retrieving data of MapReduce procedure. From the results, we can know that the average total time for this query is around 64 seconds. But this could increase significantly with increase in amount of data.

## 6. Future Goal

### ✚ Test on Other Sources

For this application, we only focus on the Twitter data source. However, there are lots of useful data sources from other social networks such as Facebook and LinkedIn. Besides social networks, other categories of resource such as finance (Yahoo Finance), specific forum (Stack Overflow) and news (NYTimes) could also provide great deal of meaningful data. Since the amount of data sources is far beyond one can image, the way we gathering and retrieving the data become much more important. Therefore, due to we already have the data gathering mechanism, one of the possible future development direction of the system is the applying the system on multiple data sources.

### ✚ Include Multiple Clusters

In the distributed file system, instead of using multiple nodes in single cluster, the

other common-used system architecture is the multiple clusters file system. By leveraging the benefits of multiple clusters, the computational time by using multiple clusters would further be reduced since the tasks assigned to each cluster would be relatively small. Also, the scalability of the distributed file system by adopting multiple clusters would be much higher than multiple nodes in single cluster.

### **Implementation of New Applications**

The services we provide in this project are only few possible use cases. There are bunches of applications by combining different elements of data fields we gathered. Furthermore, if the data could come from different data sources, one definitely can further develop various services based on the information revealed from combining, comparing and computing of the sources. Thus, one of the possible further developments of the system is the new applications discovery.

### **Integration of Opinion Finder to do sentiment analysis**

By incorporating the tool “Opinion Finder,” the system may further provide useful information. The tool “Opinion Finder” is used to analyze the opinions, both positive and negative opinions, by using the words presents in the opinion sources. Therefore, by integrating the tool, we may further analyze the information we collect as sentiment information; moreover, we could further forecast the trends of the target keywords one may interest in.

### **Performance analysis for large amount of data**

In this project, we only use a comparably small set of data and analyze the system performance based on them. As we mentioned before, the behaviors of the performance analysis results are different from the theories due to the dominance of procedure is the data allocation and retrieving. However, one of the significant advantages of distributed computing is the efficiency of the computation. Thus, instead of analyzing the small amount of data, our future work should address the benefits of distributed computing on big data.

## 7. Reference

- [1] Apache Flume, available at: <http://flume.apache.org/>
- [2] Apache Hadoop, available at: <http://hadoop.apache.org/>
- [3] Apache Hive, available at: <http://hive.apache.org/>
- [4] Cloudera's Distribution Including Hadoop (CDH), available at: <http://www.cloudera.com/content/cloudera/en/products/cdh.html>
- [5] *CDH4 Data Sheet*, Cloudera available at: <http://www.cloudera.com/content/cloudera/en/resources/library/datasheet/cdh-4-datasheet.html>
- [6] David Kirkpatrick, "*Social Media Marketing: Data mining Twitter for trends, sentiment and influencers*," available at: <http://sherpablog.marketingsherpa.com/social-networking-evangelism-community/twitter-data-mining/>
- [7] *Slim Framework*, available at: <http://www.slimframework.com>
- [8] *Twitter Statistics* from Statistic Brain. Available at: <http://www.statisticbrain.com/twitter-statistics/>
- [9] Crontab main page, Linux