Clustering Tweets Analysis

Kangbo Mo(526413), Yang Yu (578216), Jie Jin (652600), QingyangHong (629379), Ma Xiao (624053), Xiantian Luo (661898)

Abstract—The goal of this project is to create and develop a Cloud based system for extracting knowledge from a large number of tweets, research and analyze data i.e. tweets through 15 scenarios. The final implementation demonstrates all the 15 scenarios dynamically. We also integrate Google earth API into the visualization to show the location where the most recent 400 tweets were posted. This report consists of 5 sections: background introduction, design and architecture of system, orchestration and insights, web server and web site, and conclusion. We discuss about the pros and cons of NeCTAR in the section of orchestration and insights. And we discuss the research scenario and analyses in web server part. The process of this system is divided in 4 parts, and each part is running independent. Firstly, use 2 instances to harvest tweets from Melbourne and Los Angeles, then distributed to 4 computing instances. Second is each commuting instance maps and reduces the tweets for different scenarios. Third step is merging the results from all computing instances. Finally, visualization the result and displayed it on webpage. The process was mainly talked in the section of design and architecture of system.

Index Terms—Twitter, Cluster, Cloud Computing, Map Reduce, CouchDB, NeCTAR

1 Introduction

NOWADAYS, Microblog has become a very popular communication tool among Internet users, especially young people. Numerous messages are passing daily on popular websites that provide services for microblogging such as Twitter, Wechat and Facebook. People write about their life, share opinions on various topics and discuss current events, activities and issues. Easy access of microblogging platforms tends to make Internet users shift from traditional communication tools (such as traditional blogs or mail) to microblogging services.

As users increasingly post and express their political or religious opinions through those social websites or apps, microblogging websites become valuable sources of analyzing peoples opinions and sentiments. Thus, data from the social websites can be efficiently used for marketing or social media research. Twitter, was created in March 2006. The service rapidly gained worldwide popularity, with 500 million registered users in 2012, who posted 340 million tweets per day. In 2013 Tweeter was one of the ten most-visited websites, and has been described as the SMS of the Internet. Furthermore, twitter provides the function for users to follow and make friends with other users, which establishes a direct social graph for users.

1.1 Aims and Goals

In this project, we use data formed of collected messages from Twitter in order to examine NeCTARResearch Cloud capabilities and its use of OpenStack on the Twitter API, and CouchDB with its supported data MapReduce service. Our goals can be summarized as follows :

1. Harvesting tweets from Los Angeles and Melbourne city.
2. Pushing harvested data into CouchDB.
3. Conducting data analytic scenarios.
4. Developing a Cloud based solution for that purposes.
5. Exploring issues,challenges, advantages, and disadvantages in using the NeCTAR ResearchCloud and CouchDB.

2 System Design and Architecture

Due to the limited number of total instances(which is 8), the whole system is made up of two harvester instances, four computing instances, one merging instance and one web server. Only computing instance and web server contians CouchDB. The system design is focus on speed, the performance of analysis, instead of collecting more tweets. (See Fig. **??**)

2.1 Back-end Design and Implementation

The back-end program is implemented by Scala and its powerful library-Akka.
The system uses two harvester instances which are responsable collecting data from both cities. Then distributing tweets to four computing(slave) instances. The Merging(master) instance send a indexing request to computing instances per 5 minutes. Meanwhile it schedules a dispatcher which does a merge job itself every 10 minutes. After that, pushing merged data to web server's CouchDB. The reason for only using two harvester is that it is quite easy to get a full rate Twitter Developer ID, which means the harvester can obtain 100% tweets from Twitter without limitation. Futhermore, imagine harvesters collects 10,000 per minutes, the cost of doing Map reduce on massive data using single 4GB RAM machince(small instance) is extremely high. That is the main reason why the system deploys four instances with four CouchDB, utilizing a parallel computing enviroment to do the heaviest job of the whole system workflow.

The membership lifecycle (see Fig. **??**) of this system is described as follow: a instance begins in the joining state. Once all instances have seen that the new instance is joining (through gossip convergence) the leader will set the member state to up. If a instance is leaving the cluster in a safe, expected manner then it switches to the leaving state. Once the leader sees the convergence on the instance in the leaving state, the leader will then move it to exiting. Once all instances have seen the exiting state (convergence) the leader will remove the node from the cluster, marking it as removed. If a instance is unreachable then gossip convergence is not possible and therefore any leader actions are also not possible (for instance, allowing a instance to become a part of the cluster). To be able to move forward the state of the unreachable nodes must be changed. It must become reachable again or marked as down. If the instance is to join the cluster again the actor system must be restarted and go through the joining process again. The cluster can, through the leader, also auto-down a instance after a configured time of unreachability.
For simplicity, the harvester instance is not the part of the membership. Since it does not participate the computation tasks(only gather tweets).
The application uses tweet ID, which is unique, as the documents ID stores on the CouchDB. Once the harvester receives tweet, it would try to look up and delete existing documents with the same tweet ID on all databases before save them. That is the way of removal of duplicate tweets.

2.2 Fault Tolerance

"Akka is a toolkit and runtime environment for building highly concurrent, distributed and fault tolerant event-driven applications on the JVM." - from akka.io. High-level abstractions like Actors, Futures are used in the system. Akka Cluster provides a fault-tolerant decentralized peer-to-peer based cluster membership service with no single point of failure or single point of bottleneck. It does this using gossip protocols and an automatic failure detector.

2.3 Scaling Out

As Fig. **??** shows this system can be scaled out dynamically to meet the requirement. That is, if we have enough instances, deploying more harvester instances, computing instances or even merging instaces is relatively simple which only require to change a couple of configure files. In details, the system can collect tweets not only Mel and LA, but also user specified cities. Using one instance to merge all cities' database, let's say 10, is somewhat slow. In contrast, deploying more instances to merge

result parallelly would be a better solution. Similarly, the more computing instances are used, the indexing time cost are shared among them.
Please see the next section for more details.

2.4 Back-end Execution

∗ Please make sure Scala, SBT, CouchDB are installed correctly.
There are a few configuration files under **back-end/src/main/resources/** need to be noticed.

- ”application.conf” & ”stats.conf”: Akka Cluster Actor System
- ”location.conf”: the boundingboxes of the city
- ”Oauth.conf”: Twitter develop ID
- ”scenario.conf”: CouchDB views
- ”slave.conf”: the IP Address of computing instance
- ”webserver”: the IP Address of web server

2.4.1 Harvester Instance

1) Change ”slave.conf”
   Format: slave=IP1 IP2 IP3 IP4...
2) **sbt**
3) **runMain harvest.MelStreamer**
   or **runMain harvest.LAStreamer**

The Harvester can also collect tweets from user-defined city.

1) Change ”location.conf”
   Format: CITYNAME=BOUNDINGBOX
   E.g. la=-118.668167 33.703732 -118.155212 34.336781
2) **sbt**
3) **runMain harvest.LocationStreamer la**

2.4.2 Computing instance

1) Go to ”application.conf”, change host-name to the IP of that instance
2) Change seed-nodes to all the computing instances’ IP address
3) **sbt**
4) runMain stats.StatsComputeNode PORT
   The port number is depending on the ”application.conf”. By default, it’s 2551.
   i.e. **runMain stats.StatsComputeNode 2551**

2.4.3 Merging instance

1) Go to ”application.conf”, change host-name to the IP of that instance
2) Change seed-nodes to all the computing instances’ IP address
3) The merging node application supports adding new scenarios(views), the new views should be created under ”_design/analysis”. Then change ”scenarios.conf”.
   DBNAME_scenario=V:GL V:GL ... V:GL
   V for view name, GL for group level.
   e.g.**la_scenario=follower_friend_cmp:1 get_activiness_day:1 ...**
4) sbt
5) runMain stats.StatsMergeNode CITY1 CITY2 ...CITYN
   FYI, CITYs should be the name of the database stores on the computing instances. And CITYs are the DBs you want to merge. You can use one instance to merge all cities’ database. Or deploying more instances to merge result parallelly
   e.g. **runMain stats.StatsMergeNode mel la**

3 Orchestration and Insights

For better usability of the system, a series of scripts are developed to deploy environment of our system on the *NeCTAR* cloud. Such orchestrations include creating and deploying of virtual machines, installation of all necessary software and upload of all configuration files.
To have further insights into the *NeCTAR*, its pro’s and con’s is discussed in this part. As a research cloud to provide ICT infrastructure, *NeCTAR* has proper features to be keen on the research requirement. Simplification on the combining of instruments, data, computing, and analysis procedures makes the researchers have more concentration on the research tasks rather than orchestrating the research tools.

3.1 Environment and Deployment

The twittering system consists of 8 VM instances, which can be seen as 8 nodes in the cloud environment. Respectively, they have different configurations, i.e. software installation,

which are decided by the role of each node, namely server, slave, harvest or client.
As the first step, the grounding, some just initialized VMs, which would be customized as part of the cloud environment. *NeCTAR* provides a straightforward GUI, the Dashboard, to complete this job. Through the dashboard, we can not only create instances, attach volumes to them and authorize security access, but also can inspect status of all parts of the system. The interface provides handy and organised procedures to set up such infrastructure.
For more convenience in the deployement procedures, a sophisticated method is to have scripts to run all the installation and configuration operations.
Operations basically consist of two phases:

- VM setting up: Creation of all necessary instances; Attaching of volumes to corresponding instances; Authorisation on security success.
- Environment configuration: Installation of necessary software and packages on each VM; Uploading configuration and source files; Software configuration.

In the 1st phase, Boto, a Python interface to AWS, is used to complete VM setting up job. Amazon EC2 service API is provided on *NeCTAR*, which is also supported by Boto package. By accessing EC2 service, it becomes easier to obtain all kinds of information on *NeCTAR*, such as regioninfo, available image types, security groups etc. With complete control of the computing resourceyou can create instances and scale volumes quickly, what's the most important to the extent that just meets your demand. In such scenario, computing resources are paid as you use.
In the 2nd phase, Fabric, a Python command-line tool to deploy applications and administrate systems by SSH is used to install softwares and configure. Since the VMs are set up as different roles, the installations and configurations are customized on each of them. In terms of implementation, each VM would run a different suite of operations(shell commands) via *fab* command-line tool so that they would be orchestrated to the right conditions.
As can be seen, both tools used in the system are Python packages. Python is used, with the packages' features, to produce light-weight scripts for the best convenience on deployment. Besides, some auxiliary shell scripts are developed to organise python scritps in order to make them run on specific remote VM via SSH. Since the tools use SSH, IPs of the VMs are obtained through EC2 service and stored at the local directory. To apply the IP file, *scp* operation in a shell script would copy corresponding configuration files to a right IP host, a specific VM.
To get access to a instance, a personal keypair needs to be provided, which guarantees a secure access to authorised instance. By this means, only the creator of a VM with the right personal keypair can login in the VM.
A similar schema is used to access the EC2 API. Related credential files should be downloaded to provide corresponding key information to EC2 connection, i.e. aws_access_key_id and aws_secret_access_key.

3.2 Boto

Boto supports full kinds of Amazon Web Services, such as computing, database, deployment and management, networking etc.
In the system, Boto is used to invoke Elastic Compute Cloud service, with following steps:

- Creating a Connection: Create a connection to the EC2 connection object
- Launching Instances: Create instances in a specific region with proper parameters. In this step, both the personal keypair and EC2 security groups need to be set up.
- Checking Instances Status: Pull the status of instances to check if they are running correctly.
- Attaching a volume: Use EBS(Elastic Block Storage) for storage. The volume attached should be in the same region zone as the instance.

The first step to creating a EC2 connection is to decide the regioninfo, including the service endpoint.When makeing a connection, aws_access_key_id and aws_secret_access_key are provided, with region, port assigned.

```
r=RegionInfo(name='melbourne',
endpoint='nova.rc.nectar.org.au')

ec2_conn = boto.connect_ec2(
aws_access_key_id='...',
aws_secret_access_key='...',
is_secure=True,region=r,
 port=8773,path='/services/Cloud',
 validate_certs=False)
```

Launching Instances is based on the EC2 connection. The creation could be provided with several options. Some basic options are as follows. key_name and security_groups are both necessary for secure access.

```
reservation=ec2_conn.run_instances(
            image_id,
            key_name=myKey,
            instance_type='m1.small',
            security_groups=['default'],
            placement =
                'melbourne-qh2')

instance = reservation.instances[0]
```

Create a new volume in the same region as the instance that it plans to attach to. Attaching directory could be /dev/xvdx in some Linux kernels.

```
vol = ec2_conn.create_volume(5,
    "melbourne-qh2")
curr_vol =
    ec2_conn.get_all_volumes([vol.id])[0]
while curr_vol.status != 'available':
    print 'volume is %s' %
        curr_vol.status
    time.sleep(5)
    curr_vol.update()
    print "volume state: %s" %
        (curr_vol.status)
vol_add=ec2_conn.attach_volume
    (vol.id, instance.id, "/dev/sdx")
```

The setting-up procedures above are included in a connect.py script. As a result of these procedures, all VMs are initiated for the next phase's configuration.

4 FABRIC

4.1 Introduction

Fabric is a Python command line tool and a library to streamline SSH for system administration and software installation. It can let users execute Python functions through command line and shell commands through SSH, and also automate interactions with remote servers by combining the two functions.

4.2 Features

Fabric is a tool to let users to perform centralized application configuration. One can run system administrative tasks on all the machines at the same time. Its very easy to install and also quite fast and start using right away since there is not other configuration, just install and add tasks to the fabfile.

Fabric does not need anything else on your remote sever except a working SSH. If one can monitor a machine via SSH, one could also do it with Fabric. This is also the reason that lots of system administrators use this tool to perform their daily tasks.

4.3 Functions

In the same directory where one utilizes the Fabric tool, one can create a Python file called fabfile.py. Fabric will use this fabfile.py to perform tasks defined in it.
Following two steps will be used to execute Fabric funcions:
1. To establish an SSH session with the remote server, one should specify the remote sever host name and the key file in the fabfile.py.
2. To perform Fabric functions, defile a function the fabfile.py, e.g. hello(), one can run this function as follows: fab hello.

4.4 Application in Deployment

During application deployment, Fabric library will be used to script the execution of couchdb

and standard Linux commands will be issued to install packages such as Scala, sbt, git and tmux.
At the beginning of the fabfile.py, key file and remote sever directory should be defined. Following is the example:

```
env.key_filename=['my_new_keypair.pem']
remote_home_dir = '/home/ubuntu'
```

As multiple virtual machines need to be created on Nector, multiple IP addresses would be generated and saved to the working directory, e.g. ip_address.txt. In the fabfile.py, set_hosts() function is defined to read the IP address iteratively in the ip_address.txt as follows:

```
def set\_hosts():
    env.hosts = open('ip.txt',
        'r').readlines()
```

Then the deploy() function is defined afterwards. All the tasks the user needs to perform on the remote server should be included in this function.
By performing the following command, one can deploy the same tasks on multiple virtual machines.

```
fab set_hosts deploy
```

Inside the deploy() function, one needs to write the following command to specify the working directory on the remote server before issuing any Linux command.

```
with cd('/working directory'):
```

Before installing any package on the remote server, one can run the following command to simply make sure all the list of packages on the remote server is up to date to avoid dependency issues between different packages.

```
apt-get update
```

Scala is a modern multi-paradigm programming language designed to express common programming patterns in a concise, elegant, and type-safe way. It smoothly integrates features of object-oriented and functional languages. Scala is used for developing applications with Spark, which is an alternative to MapReduce. Once entering the working directory, make sure the working directory has write access to the directory and then install the scala. Fist download the scala package and unzip it, and move it to the right directory.

```
sudo('chmod u+w ../ubuntu')
sudo('wget
    http://www.scala-lang.org/files/archive
/scala-2.11.0.tgz')
run('sudo tar -xvf scala-2.11.0.tgz')
sudo('mv scala-2.11.0 /usr/share/scala')
```

To use the full flexibility of scala, sbt should also be installed as below. First the sbt package needs to be downloaded from the internet, then use Linux put command to upload to the remote server. In the put command, the upload directory should be provided and the use_sudo parameter should be set to True, which allows the command to be executed as the superuser.

```
put('sbt-launch.jar', remote_home_dir,
    use_sudo=True)

#Create a script to run the jar, by
    creating ~/bin/sbt with these
    contents:
SBT_OPTS="-Xms512M -Xmx1536M -Xss1M
    -XX:+CMSClassUnloadingEnabled
    -XX:MaxPermSize=256M"
java $SBT_OPTS -jar `dirname
    $0`/sbt-launch.jar "$@"

#upload the sbt file to remote server:
put('sbt',remote_home_dir,use_sudo=True)
sudo('mv sbt /bin')

#Add the directory /usr/local/bin to PATH
    variable:
sudo('echo -ne "export PATH=$PATH:~/bin"
    >> ~/.bashrc')

#Make the script executable:
sudo('chmod u+x /bin/sbt')

#Install Git, couchdb, tmux
sudo('sudo apt-get -y install git')
sudo('sudo apt-get -y install couchdb')
sudo('sudo apt-get -y install tmux')

# tmux tries to read .bash_profile
    instead of .bashrc
sudo('sudo echo -ne "source ~/.bashrc" >>
    .bash_profile')
```

```
#create .tmux.conf
sudo(sudo cp
    /usr/share/doc/tmux/examples/screen-keys.conf
    ./.tmux.conf')

#Configure couchdb,
#create the build dir
with cd('/mnt'):
   sudo('mkdir lib')
with cd('/mnt/lib'):
sudo('mkdir couchdb')
```

Use Linux sed command to change build dir from var to mnt, open the port 5984 for couchdb and change the bind address to 0.0.0.0, add database dir(Specifies location of CouchDB database files) and view index dir(Specifies location of CouchDB view index files) to local.ini, with cd('/etc/couchdb/'):

```
with cd('/etc/couchdb/'):
      sudo('sed -i "12s/;port = 5984/port
          = 5984/g" local.ini')
      sudo('sed -i "13s/;bind_address =
          127.0.0.1/bind_address =
          0.0.0.0/g" local.ini')
```

Change the content of couch.uri file which contains the full URI that can be used to access the instance of CouchDB with cd('/var/run/'):

```
sudo('echo "http://0.0.0.0:5984/" >>
   /var/run/couchdb/couch.uri')
```

4.5 Success and Insights

Using python Fabric to establish an SSH session with the virtual machines and script the execution of the database and application installation saves users a lot of time. Fabric could be applied to many tasks, suck as server restart and stop, process restart, application deployment and string replacement within the configuration files. One could define all the tasks in the python and Fabric will execute the tasks on all the devices the user specified. Hence, the user does not need to log into different servers to install the same packages over and over again.
As Fabric is easy to use and very powerful, it will become more and more popular among system administrators to perform their daily tasks. If same changes need to be applied to different severs, system adamin could just run one Fabric script to delploy all the changes to all the severs.

4.6 Pros and Cons of NeCTAR

Instances running on the NeCTAR Research Cloud are like real-life machines but in a remote location. The Research Cloud is used to start, copy and delete instances. Currently all instances are virtual machines.
The pros are: the user could select an operating system(a list of systems is provided), network access(a real IP address or user specifies) and also hard disk storage, the user does not need to maintain the hardware. The user could also copy snapshots and customize new machines easily and swiftly. Once the instance is created, the user could access to the information anywhere, where there is an Internet connection. The Dashboard is not the only way to operate the Research Cloud. There are many tools and interfaces (like nova, Fabric) that are compatible. Once the user gets used to this method of functioning, the whole system could be fully setup in a matter of a few minutes.
The cons are: Cloud computing make the tasks users wants to perform dependent on the reliability of both the Internet connection and NeCTAR system. When the Internet connection breaks up or the NeCTAR system is down, the user could not carry on the tasks. Security is another major issue. The user surrenders all the sensitive information to a third-party cloud service provider, which could probably put the user at great risk once the information is revealed to the public. As people are well aware, the Internet is not safe, some external hackers could easily attack the information stored on the Cloud system. The NeCTAR is also lack of technical support. When something wrong happened, the user should log a ticket and have to wait for the email reply, which might take a long time.

5 Web server and website

In order to exhibit our analysis results and scenarios, we developed a website to allow

anyone to access our work from anywhere via browsers. We set up an instance in our *nectar* cloud to specifically serve webpage requests. We tried to make our webpage simple and easy to use. With the use of figures, charts and map, we believe that users will be able to understand our analysis results and scenarios much easier.

5.1 Web server design and implementation

To ensure the web server to be able to serve requests whenever users want and avoid any possible resources competition, we allocate a single VM instance as web server. This server has a *couchDB* storing the final analysis results which is the combination of results from each slave node. We choose to use *apache* as the web server and PHP as our website development language. In order to ease the burden of *couchDB* and serve request instantly, we introduced a cache strategy that uses *memcached* to temparily store frequent query results in memory. If any following requests' results can be found in the cache, the results will be immediately sent back from memory without querying the *couchDB* again. This system works as shown in Figure **??**. What need to mention is that the content in the master couchDB is not controlled by the web server. It relies on the system mentioned in section 2 that regularly merge content in slave couchDB and put the result into master couchDB.

The design of request and response follows the rules of RESTful API. Traditionally, a request URL is something like *"http://domain.com/userinfo.php?name=jack"* that contains the path of target file (userinfo.php) and the arguments passed to it (name=jack). While, a RESTful URL is supposed to be like *"http://domain.com/userinfo/jack"*. The solution we adopt to change tranditional URL into RESTful URL is using *mod_rewrite*, which is provided by apache server to rewrite URL. Our rule for rewriting is *"RewriteRule ^scenario(.*)$ scenario.php?s=$1 [L]"*. It will rewrite any request URL matching *"^scenario(.*)$"* to *"scenario.php?s=$1"*. For example, if a user request *"http://domain.com/scenario/Language"*, the request URL will be rewrote to *"http://domain.com/scenario.php?s=Language"*. Then, the *scenario.php* will handle the request with the argument *"s=Language"* and return the query result in a JSON file. Because website users are only allowed to browse our analysis result. They are not grant the permission to delete, update or insert new data. Therefore, our web server only allow user to use *"GET"* method to obtain data. All the other methods, namely *"PUT"*, *"DELETE"*,*"POST"*, are not allowed and will return error message if user tried to use them.
When recieve a request from user, our PHP program will fist validate it and then fetch corresponding data from couchDB if the data cannot be found in cache. The communication between PHP program and couchDB is via HTTP connection. Some of our scenario requires data from more than one view in *couchDB*. The PHP program will retrieve all the required views and merge them into a single JSON file, and then send it back to users.

5.2 website design and implementation

Our website is based on a template provided by *Bootstrap*. We use *Bootstrap* for layout, *jQuery* for *ajax* communication, *google chart* for drawing figures and *google map* for presenting data on map. To make it easy to use, most of our scenarios are presented on a single webpage instead of opening new webpage for each scenario. When user select a scenario, the data will be fetched on the fly in background. User is free to do any other operation during the wariting time. Besides scenarios, user can find other information, like our report and teamates, on the website as well.

Figure **??** shows the layout of our website. The sidebar lists all the scenarios we have done. If user click on an item, *jQuery* will use *ajax* to retrieve related data from web server. The request URL is RESTful as discussed in section 5.1. When data successfully returned, the main content area will be updated to show corresponding analysis data of the scenario. To make the comparison between the two cities straightforward, we use different kinds of ways

for presentation, such as bar chart (Figure **??**), column chart, pie chart and map (Figure **??**).
Another feature of our website is that user can search the Tweeters of a particular user by putting its name in the search bar. The result will be shown in a list. Given the fact that some Tweeter users may have same username and we are unable to distinguish them, the result we give here is a collection from all users have that username. In other words, we assume username is unique and belongs to only one Tweeter user.

5.3 Discussion

The robust of our web system is largely depend on the software and frameworks we used. While, most of them are well-known software and frameworks in the industry. Their qualities to some extent can be guaranteed. The only part we do not use any framework is the PHP program. The reasons are two folds. First, it acts as a middleware in this system that fetch and merge data from *couchDB* and send back to users. Currently, there is no such framework that focuses on *couchDB*. Second, the function required to be done by PHP is relatively simple. There is no need to use burdensome frameworks.
In our web system design, we tried to make it dynamically scale out to meet large amount of users. The bigest challenge to serve a large number of users is about the burden on web server, especially on database. We adopt some strategies to relieve our server. First, we let webpage to retrieve javascript libraries from public resources, such as google. Those libraries are normally heavy and required for browsers by every user. By dispersing among other domains , our server just need to send a small amout of data back to front-end user. Meanwhile, this strategy shorten the request time so that our server can serve more users simultaneously. Second, we use *memcached* to cache frequent request in memory. It can largely decreases the times to query database. In addition to relieve database, it speeds up the request handling as well.
In conclusion, our web server which uses one instance is relatively robust and reliable. Due to resource and time limitation, we are not able to develop a multi-instances based server. If we could use multiple instances to work as web servers, the fault tolerance will be largely increased, and it will be able to serve more users simultaneously. This is the further work we could carry out in the future.

6 Research Scenarios and Analyses

6.1 Scenarios and Corresponding Views Introduction

In order to make a meaningful comparision about the life styles, individual characteristics and cultural diversity of Los Angeles and Melbourne, we have carefully designed several analysis (scenarios) topics that could objectively reflect the status of the above topics in both cities from different perspectives based on the tweets we harvested in the last two weeks. And a brief conclusion on which of the two cities is relatively more liveable would be generated based on the results of these scenarios. Since *couchDB* is used to store all the contents of tweets in this project, we chose to take advantage of the famous *MapReduce* views of *couchDB* to implement these analysis scenarios. In details, *Javascript* languages was used to code the *MapReduce* views for each scenario inside both databases for Los Angeles and Melbourne and a brief introduction about the design and function of each view (scenario) is listed as follows:
1. get_activeness_by_day: This is a simple view which counts the number of active Twitter users in both cities by days in a week. Since Twitter is probably the most prevalent social networking tool in both Los Angeles and Melbourne, we believe the sum of active users in a certain day of a city could well reflect the city's vigorousness on that day. To be more specific, the vigorousness of a city should be proportional to its number of Twitter users, say the more active users in a certain day, the more vigorous that city is. And in order to guarantee the statistic fairness of all the scenarios that related to dates or times, we have carefully restricted our harvesting server to harvest an entirely two weeks' tweets as

mentioned above.
2. get_create_year: This is also a straightforward view to count the creation time of each Twitter account in both cities by years. This could reflect their pace of pursuing the latest technical fashions by finding out the creation years of the majority of Twitter accounts in a certain city. Generally speaking, the eailer the majority of Twitter accounts in a city were created, the more fashionable the city is.
3. get_different_languages: This views aims at counting the types of languages that were used in the tweets of both cities and for each type of language, the number of tweets based on it would also be aggregated to reflect the popularity of that language in both cities. By analyzing the languages used in tweets, we could eaily tell the level of cultural diversity in both cities and which language is most popular (in this case, of course English is prevalent in both cities). A typical judgement criteria for this scenario could be the more types of languages were found in the tweets of a certain city, the higher level the cultural diversity in that city is.
4. get_highly_used_words: This views focuses on finding out the highly used words (frequently used words) in the tweets of both cities, the corresponding scenario is to take a general view about the speaking habits of people in both cities and to figure out what the people in these two cities concern the most. In order to better achieve the goals of finding out only meaningful words, we have introduced a list of ”stopwords” (e.g. ”the”, ”http”, ”and”, ”about” and so on) in our *javascript* code for this view to remove meaningless words. And in order to reduce the processing pressures in the graphic demo side, this *MapReduce* funtion would only emit words that appears more than 500 times and send them to the server that holds the demo website for further process (for instance, to only display the 100 most frequently used words).
5. get_eat_habits: This is a 2-level *MapReduce* view which aims at analyzing the dietary habits of people in both cities and two lists of words that consists of the names of foodstuff were introduced as the judgement criteria. One list contains the names of popular ”meat, fish and seafood” and the other list is a set of names for ”fruit and vegetables”. In correspondence, the first level of this view would only count the number of tweets in both cities that mentioned ”meat, fish and seafood” or ”fruit and vegetables” and the second level would count the occurence times of a specific type of food that appears in tweets. Through this 2-level view, a rough idea about whether people in the two cities eat in a healthy way (e.g. whether the proportion of meat and vegetable is reasonable) could be generated.
6. get_eat_habits_suburbs: This view is a complement to the above one and it aims at analyzing the dietary habits of people in both cities by suburbs (by refering to the location of where a certain tweet is posted). Through this view, we can not only compare the dietary habits of Los Angeles and Melbourne, but also find out which food people in each suburbs of both cities are interested.
8. get_positive_negative: The main purpose of this view is to roughly categorize twitter users' moods by days in a week or by hours in a day, we have carefully selected a list of positive words and another list of negative words to help us reasonably judge people's moods through the tweets they posted, which could be then used to decide which city tends to have a more positive life attitude. And in order to guarantee the quality of the scenario, we have carefully designed this view to be sensitive about any privative adverbs (like ”not”) that occurs right before a positive or negative word since the whole meaning of the tweets could be entirely adverse in this case. Moreover, this view has a 3-level structure so that we could easily categorize the mood by days in a week in level 2 (e.g. find out ”how many Twitter users have a positive mood on Monday”) or by hours in a day in level 3 (e.g. find out ”how many Twitter users have a negative mood between 5 and 6 o'clock of Tuesday”) conveniently as mentioned before.
9. get_positive_negative_suburbs: This view is a complement to the above one and it focus on counting the moods of people in both cities by suburbs, which could provide us more details

about a general life attitude in each city
10. follower_friend_cmp: This scenario aims at comparing the number of friends and followers of each Twitter account. Since a follower means that the account user is focused by someone, which is probably because the user is special, popular or full of wise thoughts, the number of followers an account have could reflect the popularity of the corresponding user. And in turn, the number of friends an account have might reflect whether the user is willing to care about others' opinions or to make friends (whether you are open enough or social enough). As a result, if an account has more followers than friends, that probably means that the user is very popular or even a celebrity. And if the account has more friends, that probably means the user is a ordinary person or keeps a low profile. By analzing the results of this scenario, we could roughly estimate which city is more popular or at least, has more celebrities.
11. get_public_interest_sports This view focus on counting the popularity of European Soccer (as the most popular sport in the world) in both cities, which would be used as an index to decide the public interests on sports showed by people in both cities. Moreover, this view also counts the popularity of Los Angeles' NBA teams among Melbourne people and the popularity of Melbourne's AFL teams among Los Angeles people. The point is to decide which city's major sport (sport teams) has a wider global influence.
12. get_interest_worldcup: Since the 2014 World-Cup is coming soon, we also designed a simple scenario to find out the popularity of WorldCup in both cities by counting the number of tweets that interested in WorldCup, which could also be used as an index to reflect people's interest on sports in both cities.
13. get_public_transport_satisfication: This scenario counts the number of tweets that showed a positive attitude on the local transport system in both cities and the point is to find out people's degree of satisfication on the local transportations, which could be a critical index to decide the liveable degree of a city.
14. get_twitter_terminals: This scenario focus on counting the number of different terminals that people in both cities used to post tweets. It could well reflect people's preferred ways of involving in social networks and it is also a good index on judging whether iPhone (iPad) or Android-based devices is more prevalent in the two cities.
15. get_twitter_locations: This scenario aims on getting the location of where the most recent tweets were posted (e.g. the coordinates of the 400 most recent tweets) and displays the mapreduced results by dots on *Google Map*, which could present us with a clear view about a sketchy distribution of Twitter users in both cities.

6.2 Analysis Results based on Views

6.2.1 Scope of the Anaylysis

Due to the different scales of populations in Los Angeles (large) and Melbourne (small), the number of tweets we could harvest from these two cities are significantly differnt – About 1.48 million from Los Angeles and only 0.1 million from Melbourne. Thus to ensure fairness in analysis, we would not use the absolute numbers to compare between two cities. Instead, the percentage of a certain kind of tweets over all tweets in a city is the key index that we concerns.

6.2.2 Analysis and Conclusions

Through the anaylysis against the results of the above views, some brief conclusions about the life styles, individual characteristics, cultural diversity and livable degree of Los Angeles and Melbourne could be generated and several of them are listed below. Besides, several charts or maps which displays some of these results are also listed below to support the analysis.
1. We have found that for Melbourne, the most active day in a week is Saturday (about 4800 Twitter users were active) while for Los Angeles, the most active days are both Friday and Sunday (about 51000 Twitter users were active on both days). Thus, it seems people in these two cities have different scheduls for celebrating weekends.
2. And in terms of the account creation year,

both cities have almost the same trends throughout the past 8 years. The first Twitter account in both cities were created in 2006, the birth year of Twitter Company while for both cities, the users of Twitter were gradually increasing since then and reached the peak in year 2009. Thus, a rough conclusion could be drew that both cities have the same level of fashion to pursue the latest technologies.

3. Based on the language analysis, we have found that both cities have a number of users from worldwide. For Melbourne, its tweets were posted in about 19 languages and for Los Angeles, XX different languages were detected. Thus these two cities both are very international and cultural diverse.

4. From the scenario view which count the hot words, the top 3 popular words in tweets from Melbourne are ”SOS”, ”luke” and ”Melbourne” while that of Los Angeles are ”lol”, ”love” and ”good”. From this we could conclude that it seems ”SOS” is a very hot pet phrase for Melbourne people and they seem to prefer to frequently ask someone for help in case of difficuties through posting tweets while Los Angeles people seem to be more positive since they ”always” mention about love, like to ”laugh out loudly” or say ”good”. One more interesting thing here is that it seems people in Melbourne concerns about their city more than people from Los Angeles since ”Melbourne” even ranks top 3 in hot words of Melbourne people.

5. Based on the results of *get_positive_negative* view, both cities arised more positive tweets than negative ones. In detail, for Melbourne, the proportion of positive tweets and negative ones are about 2:1 while for Los Angeles, the proportion are close to 1.16 : 1. From this result we may draw a rough conclusion that the proportion of people in Melbourne who shows a positive life attitude is relatively larger than that of Los Angeles. But since the ”hot words” analysis has showed that top hot words in Los Angeles are all positive ones, we may say that people from both cities are pretty positive towards daily lives.

6. From the results of the two views which analyze about ”public interests on sports”, we have found that people from both cities are very interested in soccer, especially famous European soccer teams. Moreover, as the hottest sports in Los Angeles, NBA teams and stars in Los Angeles has even drew some attentions by people from Melbourne while people in Los Angeles seem do not care about AFL, which is the top 1 sports in Melbourne. Thus, it seems that Los Angeles’ popular sports have a larger international influences. One more interesting thing is that as the hottest pageant around the world, the coming Brazil WorldCup seems do not grab deserved attentions from people in both cities. And this might simply because both cities have more popular sports than soccer.

7. From the results of ”dietary habits” views, we have found that people in both cities seem to prefer more about ”meat and seafood” than ”fruits and vegetables”, which is not a very ideal dietary habit. To eat in a healthier manner, people in both cities need to have more fruits and vegetables.

8. Based on the results of view *”13. get_public_transport_satisfication”*, the percentage of tweets that showed a positive attitude towards the local public transport over all tweets in Melbourne is higher than that of Los Angeles, which seems to show that Melbourne have a more satisfied public transport system. But since for both cities, the number of tweets that mentioned about public transport were very few, the result may only serve as a reference when judging the quality of the local public transport system.

Based on the analyses above, both cities seem to hold a pretty positive life attitude and are extremely cultural diverse. And people in both cities also like sports and have its own popular kind of sports, which even distract their attentions from the coming WorldCup. But they do not hold a very healthy dietary habit since they seem prefer to eat meat and seafood. But in all, a brief conclusion could still be generated that both Los Angeles and Melbourne are very livable cities for its good life attitude, cultural diversity and so on.

From Figure **??** to Figure **??** are some of the charts and maps from our demo site that displays part of the results of the above scenarios.

7 Conclusion

In this report, we have presented a pro-posed approach for analyzing Twitter user activities in Melbourne and Los Angeles using Nectar cloud and CouchDB capabilities.
We highlight the structure of a system for mining and analyzing relevant tweets. The system is designed as a cloud based solution; all analytics scenarios and corresponding results are already discussed in this report.

Appendix A
A working demo

http://115.146.86.86