COMP90024 Cluster & Cloud Computing
Assignment 2
Australian Social Media Analytics - Team CCC2018-2 Report

Jiachuan Yu(867000), jiachuany
Mengyu Zhou(780956), mengyuz2
Chenxi Hou(811596), chou1
Surong Zhu(859160), surongz
Sicong Ma(884092), Sicongm

Abstract

In this report, we introduce an implementation of geo-spatial social network analysis system on cloud environment. Our main purpose is to harvest Twitter data using multiple nodes deployed on the cloud and analyzed the correlation between tweets characteristic and Australia statistics. The design of system is demonstrated, including system structure, twitter harvesting, AURIN data collecting, data analysis based on CouchDB View of MapReduce and Twitter analysis module, and visualization on web-based frontend. A highly automated process of deployment is described. Also, seven scenarios are analyzed using machine learning techniques on Matlab with the plotted figures. Finally, we have a discussion on the result including the scalability, fault-tolerance and the ability of MapReduce according to our specific implementation.

1. Introduction

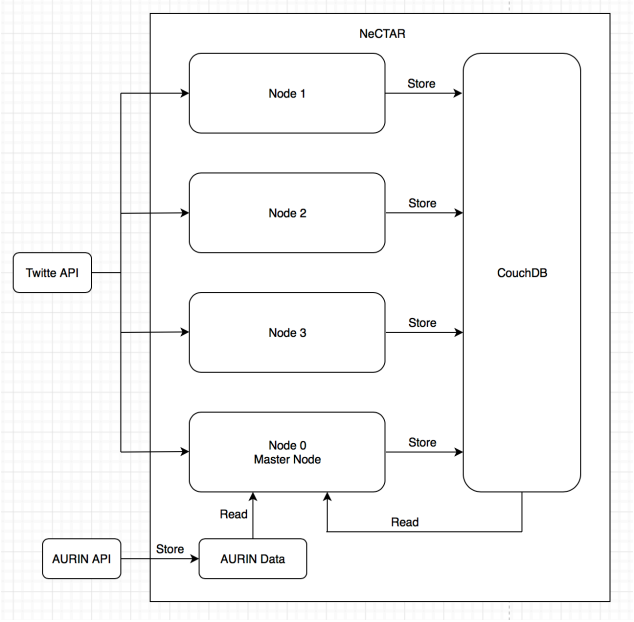
Twitter, as a social networking and microblogging service, has become a platform for millions of people to convey their ideas, thoughts and viewpoints in their daily life. According to [1], there were about 330 million active users on Twitter worldwide. In Australia, 32% of people used Twitter for their social networking[2] in 2017. Meanwhile, with the massive geographic information binded on tweets, Twitter has become the fertile ground for researches on various topics that can be combined with geographic information, such as diets, entertainments, sports and politics.

Many works have been done in this field. Stefanidis, Crooks, and Radzikowski focused on collecting large quantity of geospatial information from social media and illustrated data analytics in several cases[3]. MacEachren and et. al. harvested twitter with geographic information to perform geo-Twitter analytics for crisis management[4]. Ratkiewicz and et. al. developed a system for detecting political abuse in social media with geo-information[5], and Chae and et. al. performed spatio temporal analytics on data collected from social media for abnormal event detection[6].

In this project, a system is developed to perform Australian social media analytics under several scenarios, based on the data with geographic information harvested from Twitter, in order to find useful information hidden in the data for major Australian cities. The system includes data harvesting, data analytics and result visualizations. It is deployed on the NeCTAR Research Cloud for purpose of parallel processing to handle the huge amount of data from social media. These data are analyzed in comparison with the statistical data retrieved from the Australian Urban Research Infrastructure Network(AURIN)[7], in order to implement a deeper analysis with the combination of subjective and objective data sets. AURIN as a spatial and statistical modelling dataset from over 90 data sources, provides an access to a distributed network of aggregated datasets and information services, while it also serves a good vehicle to get different collection of data in Australia for all the researchers and institutes[7]. At last, the results is virtualized based on Google Map obtained through Google Map API.

2. System Design

The structure of the system is briefly illustrated in Picture 2.1 below.



Picture 2.1 System structure

The whole system is deployed remotely on the NeCTAR Research Cloud by a script, in order to make the whole deployment process simple and easy to be managed. Boto3 and Ansible Python package are involved here to invoke the system. Then users can perform different kinds of tasks with a script file. 4 instances or nodes are created on the cloud server, among which the node 0 is treated as the master node for tasks of data analysis and data visualization. Meanwhile, all 4 nodes run the tweet harvester parallelly. Details of the designs of each part are explained in the following paragraph, with the deployment part being discussed in section 3 due to its complexity and thus requiring a long piece of explaining.

2.1 Twitter Harvesting

Tweepy API[8] is chosen as the Twitter API because the main language used in this project is Python and it is easy to use. In terms of tweets capture, they are captured with a filtration of their coordinates, meaning that only tweets that are posted within given Australia coordinates grid are captured. Meanwhile, among the captured tweets, those possessing attributes of place name or coordinates are retained since the location where the tweet is posted is necessary in the later analysis. The rest tweets are discarded.

There are two tweets harvesting methods in Tweepy API, namely, the "stream" method and "search" method. They have several differences such as the rate limitation and queries searching ability. What we care most here is the posting time of the tweets captured. The search method is to get the tweets posted for the past 7 days, while the stream method is to get the tweets posted in near real time. According to this characteristic, the stream method is chosen in this project as the tweets harvested for analysis are those posted during the project development period, say, in the future several days. Meanwhile, the rate limitation of the search method also impairs its practicability since time for this project is limited.

In terms of tweets storage, four attributes of each captured tweet are stored in the CouchDB mounted on the NeCTAR Cloud server in the format of JSON, including text, place name or coordinates, created time and a coordinates box, where the coordinates box is only useful for tweets having the attribute of place name. The CouchDB is accessed by using the standard Python library *urllib* and the CouchDB uniform resource locator(URL).

The Twitter Harvesting process is running parallelly among all four nodes on NeCTAR. Each captured tweet is given a node ID calculated via modding the seconds of posting time by the number of nodes, which is 4 here. So literally, each node has even amount of workloads for tweets harvesting. As a result, there are four documents in the CouchDB containing the tweets harvested by each node, respectively. An interesting thing worth mentioning here is that originally, we used the last two digits of the tweet's id as the dividend to be divided by 4. However, Twitter owns a kind of unclear tweet id generation mechanism that makes the results only contain 0, 1, and 2, leaving the node 3 idol. It needs further effort of research since there is

also no clear explanation of this phenomenon online.

There are three main reasons of using multiple nodes for tweets harvesting here, of which the first one is to avoid capturing duplicate tweets. Provided the tweet numbering mechanism above, each tweet may be captured by several nodes but will only by stored by the one that has corresponding node ID. As for the second reason, the workloads for database access are separated evenly to all four nodes, which, though may not have a significant improvement in this system, is still necessary for the system's scalability. Last but not least, as mentioned above, the Twitter stream method has its rate limit that is not clear for the public. One assumption is that each Twitter app user will be assigned a certain percentage of tweets randomly at a point. Based on this assumption, we expect that more tweet harvester running at the same time can capture a wider range of tweets.

The Twitter harvester is running in the background of each node by nohup command for uninterrupted tweets harvesting. Running errors are handled by a try-except mechanism in the Python file, so the harvesting process will not be interrupted by any error. When an error occurs, the error message is recorded in a log file stored on the node. In addition, once a tweet is captured and stored, a message including the time of storing, the name of current module, the level of the tweet and the stored tweet data is recorded in the log file as well. Therefore, by reading the log file only, one can get a clear idea of the running status of the corresponding Twitter harvester.

Each Twitter harvester s managed by a configuration file containing the node ID as well as its unique customer key, customer secret, access token and access secret for Twitter API OAuth and connection. This makes it much easier for any alteration of keys and node IDs. The node ID is appointed with 0, 1, 2 and 3, respectively, to match the four different reminders of 4 as the id of each captured tweet. The scalability of the system can be guaranteed by adding other nodes with some modifications of the Twitter harvester file, including change the total node number and the divisor of the modding process. Since the workloads of the database access have been well managed by the parallelization design, there has no other changes that need to be done for adding more nodes, thus, endowing the system a good ability of scale and the accompanying configurations.

2.2 AURIN Data

Considering Melbourne is a typical area that has a great number of tweets, it is chosen as the main research area.

Melbourne has forty suburbs (such as Melbourne city, Port Phillip, Essendon, etc.). Therefore, the tweets collected are divided into these areas according to their coordinates.

Some kinds of data in these areas from AURIN are selected to help show the relationships between the results of analyzing the tweets and them.

The AURIN data includes the following aspects:

1. Median personal income per week in different suburbs;

2. The number of people who have Bachelor Degree in each suburb;

3. Median family income per week in different suburbs;

4. Median age of each suburb;

5. The number of people who have Postgraduate Degree in each suburb.

There are lots of education levels in Australia, these three main high degrees are chosen to assess the education level of the people in that suburb.

In order to show these suburbs obviously on the map, their coordinate sets which determine their boundaries are collected. Combining the data with the map according to the coordinate sets of these suburbs ensures that the specific data matches the corresponding suburb correctly.

2.3 Data Analysis

2.3.1 CouchDB MapReduce

The data for future analysis are pre-processed by CouchDB build-in MapReduce functionality. MapReduce derives from the map and reduce concepts from functional languages. It is used as a programming model for the processing of large data set. The Map method creates a set of intermediate key for the date input and associates other data that have the same key. In other words, it works as a filter that gathers all eligible data together to achieve the query functionality as in the relational database. The Reduce method merges the data with the same intermediate key together reaching a much smaller results of the data, which is usually a single value. This method can be used to handle the data that are too large for the system memory. Meanwhile, due to its intrinsic characteristics, it can guarantee the scalability to a large extent Therefore, the MapReduce has become an ideal method for super large dataset processing for cluster & and cloud computing [9].

There are also some limitations of this method. Some pointed out that the MapReduce system is built upon an unreliable communication module that may affect its performance on a sophisticated parallelization system, and the heavy workloads for processing a over large date set may lead to the master node a single failure. Ma, Z. and Gu, L. considered the intrinsic limitation of the MapReduce method is its design of one-way scalability that constrains its ability of handling small data set. As a result, it is very difficult to extent the system to a more general computation structure[10]. However, in this project, these problems may not have a considerable impact on the system, thus, the MapReduce method is introduced for data pre-processing.

All tweets stored in the database are catalogued based on their posting locations. There are totally 8 classes according to the 8 major cities in Australia, which are Melbourne, Sydney, Canberra, Perth, Darwin, Hobart, Adelaide and Brisbane, respectively. Tweets with place name are categorized directly by matching the place name, while those without place name are classified by matching the coordinates with the box coordinates provided by Twitter. After this MapReduce pre-processing, data can be retrieved much more effectively for future analysis on account of the geoinformation of the data and the comparison with the AURIN data. One thing need to be mentioned is, the build-in MapReduce view in CouchDB is far more powerful than the utilization in this project. In the future, more sophisticated MapReduce views can be programmed for data processing rather than reading the data and performing process in a separate process.

The data pre-processed by MapReduce can be accessed by the corresponding URL. For example, if one wants to get the tweets posted in Melbourne, the URL will look like *http://115.146.85.216:5984/twitter/_design/coor/_view/tweet_in_melboure*, where *115.146.85.216* is the IP address of the NeCTAR server, *5984* is the port number, */twitter/* is the name of the database, */_design/coor/* means the design document of is named as ‘coor’ and */_view/tweet_in_melbourne* represents the name of the view is ‘tweet_in_melbourne’.

2.3.1 Tweets Analysis

Tweets analysis part is implemented through Python 3. After using MapReduce view function running in Nectar server to divide all tweets including location information into 8 cities, analysis module takes data of specific cities from CouchDB and yields GeoJSON files which will be directly used at front-end presentation. This part is located at one instance of Nectar server, being invoked at predefined time stamps. Parallelizing code has been considered at this point, which could be implemented by allocating same module at four instances at Nectar server with different sets of data of cities. However, one-node process only takes about 3 to 5 minutes to complete and it only provides pre-calculated results instead of real-time analysis; there is no need to deploy the parallel function.

The whole module consists of two main functions—extracting data from CouchDB, which is

implemented by accessing the url of database as a user; analysis such as sentiment analysis, keywords filtering or the hybrid process of keywords filtering then calculating the relative support rate based on sentiment analysis. Also, when the data is coming from Melbourne view, this module also be responsible for locating each tweet at different suburbs around Melbourne once the data get inputted. Meanwhile, the sentiment analysis is supported by a pre-existing Python library called TextBlob, which provides a simple API for diving into common natural language processing (NLP) tasks such as sentiment analysis. We used the pre-trained model from TextBlob to analysis the sentiment of each tweet then divided them into either positive or negative tweet group based on the polarity value.

In addition to pure sentiment analysis to find out the happiness level of people living in different areas, for exploring the data set, we chose several scenarios closely relative to Australia life matters such as fitness and entertainment activities. The analysis of those scenarios simply filter the tweets connecting to it based on a corpus and calculate the percentage of people who like sports or afternoon tea. The reasons why we chose those scenarios not only due to them closely relative to people normal life but also because the limited tweet data we harvested and collected. Our first hunch of scenario is marriage equality or animal rights which we thought are drawing people's concerns; however, after processing the data, we found the number of relative tweets is small, and cannot present the common perspectives. So, we jump into the entertainment area which is corresponding with enough data.

There are multiple pros and cons of the method being used during this part. Firstly, we use the pre-existing library to do the sentiment analysis, which reduces the workload while may miss some important part of tweets analysis such as analysis based on emoji. Secondly, the filtering function filter relative tweets depends on the corpus manually created by us; the corpus may not be thorough, and filter use string matching method to find needed texts, which means the synonyms using, typos in tweets, whether searching by whole word all will influence the results of counting. Last but not the least, as mentioned before, the number of tweets we could use(contains location information) is limited; the results of analysis may not be extremely precise.

2.4 Visualization

A front-end web application like Fig. 2.4-1 is used for visualizing all AURIN data into different scenarios. The application has been deployed at a server supported by Apache, which makes web app can be accessed through URL via any web browsers. There are four scenarios corresponding with AURIN data, two are related to the income issues which are personal weekly income and family weekly income, the other two are related to the education level issues-bachelor degree and postgraduate degree.

At first, React JavaScript is introduced to implement those four scenarios, because it can support online interaction of data visualization, which means that twitter data can be captured by it and then update the visualization, and it is easy to frame the web page because of it has own frame. However, React JavaScript is not used commonly as the HTML and the most important function from Google Map API cannot be used. Consequently, all those four scenarios are shown in the google map via HTML.

The map is implemented using Google Map API and a GeoJSON file which is a SA3 level suburbs division information in Melbourne., which also supports different level division using Google GeoJSON. Within the Google Map API, there is a call-back function to implement the shape of each area once loading that GeoJSON file. Then, each polygon divided by Google Map is colored based on the AURIN data we got; for example, as for personal weekly income scenario, the greener area shows higher personal weekly income in that area and more red illustrates lower.

Also, each option within the selection bar at the top of Google Map can deliver each corresponding scenario, which is a switcher for four scenarios, and the name of area will be shown

in the information box next to the selection bar. In addition, when users click a polygon, it will pop out one information box containing the data attaining from twitter analysis module, including positive, negative sentiment tweet counts, the number of tweets relative to fitness/entertainment/food & drinks locating in the chosen area. The web-page get the statistic twitter data via accessing URL that redirect to the address of CouchDB storing processed data in our configured Apache server. Statistic twitter data will be generated periodically by twitter analysis module and be written into the database CouchDB, the connection between front-end and twitter analysis module.

Beside the main page, we also have a button floating at the right-top corner of the map, redirecting users to a detail page showing further analysis of relationships between the AURIN data such as income and tweeter analysis data such as positive sentiment ratio as regression curve graphs; along with situation descriptions. The animation presentation (Carousel component) of this analysis detail page is supported by pre-existing JavaScript files from Bootstrap, which significantly reduced our workload for coding Html files.

In a nutshell, the web-page can deliver the analysis of relationship between statistic twitter data from CouchDB and the data from AURIN by using Google Map API compiled by the HTML.

Based on our web-page, AURIN data of four scenarios and statistic twitter data for different area can be easily visualized, and one analysis of total relationship can also be shown when it is required to be shown. Also, the page is simple and support awaken mouse event. On the other hand, our page cannot support the online interaction that data should be updated by person periodically instead of being updated in real time. The visualization is presented in Fig. 2.4-1.

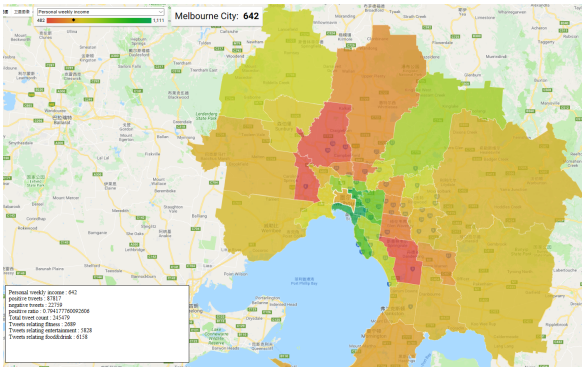


Fig. 2.4-1 Data visualization in Melbourne areas

3 Deployment

3.1 Description

To prepare a ready-to-run cluster environment can be tedious manually. Several automation methods can be applied to finish all the deployment jobs remotely with a single script.

Our process is as follow:

First, create instances and volumes on Nectar. The ids of the created instances are recorded and saved in a host.ini file for later use in Ansible script. Second, attach each instance to an available volume. Third run the Ansible playbook to setup all requirements in all instances.

We used boto3 to manage instances and Ansible python package to invoke the playbook. Finally, the user can have all tasks done with only one single script.

Key pair and security group creation is not included in the script, because we argued that these operations involves steps beyond the deployment process(such as all members create a key pair) and must be done with caution. In addition, these operations does not scale according to the size of the cluster. A cluster with 4 node, we only need to do it once. A cluster with thousand nodes, still once is all. Thus, we decide to manage key pairs and security groups manually.

3.2 Instance Management with Boto3

Boto3 in python is used to create instances and volumes, attach volume to instances. Potentially, there are functions for user to get a summary of all resources or terminate some of the resources.

The default routine in our final script is to check status of all instances and volumes, create new ones, and attach volumes. There are circumstances when user wanted to terminate some of the instances, or he/she doesn't want to remove the volumes. These can be achieved by small adjustments in the script. We have functions to terminated certain id of instances, but it seems not making things easier than doing it on nectar dashboard.

One hit everything done is pretty cool. So we only put a Terminate All function in the main routine but by default disabled since it's too risky. Only when for demonstration or the need to start from scratch, then we enable this function

3.2.1 Script Variables User Guide

The script can be applied to different project on nectar with tiny changes. Each project on nectar has unique credentials.

We save these variables as .py file, and import into the main script.

If you want to test the script on other projects, just edit a .py file with 'key' and 'secret', then import in the main script as credential:

```
import credential_demo as credential
```

3.2.2 Connection

boto3 in python has two main utilities to manage instances.

Client: the client class is used to manipulate all existing resources. Such as get a description of a resource, terminate a resource, or relating resources(attach volume).

Resource: the resource class is to create new resources, such as create instances and volumes

The first thing is to initiate the two utilities, check the nectar dashboard and retrieve a summary of the current instances and volumes used. If successful, it indicated that the connection is good, utilities are initialized correctly. If any error raised here, it would be caught at once and the process will be aborted with a brief message.

The summary includes the instance list and the volume list. If there are useful instances running, the user can abort here by interrupting the script.

```
meowmetal@ubuntu:~/cluster/ASMA2018/deploy$ python deploy.py
[{'ip': '115.146.85.130', 'id': 'i-88626bec'}, {'ip': '115.146.86.7', 'id': 'i-e
9a31da4'}]
```

3.2.3 Create Instances

Boto3 provided the function to create multiple instances in a single function call. A max count is set to specify how many instances we want. If there are not enough resources for instances, it will create only what it can do and give a warning of insufficient resource, but will not raise an exception.

The instance creation is wrapped in try catch blocks, if anything wrong, further steps must be aborted with an error message

Instance creation process is asynchronous. The create instance call returns immediately, but it takes time for the instances to get ready. If the instances are still initiating, further steps like attaching volume will fail. In addition, we need to retrieve the IP address of the instances automatically, and it is only available when the instance is up and running.

```
spawning instances
New instance i-27c35fb6 has been created
instances initializing
instances initializing
instances initializing
instances initializing
New instance i-cd0bb828 has been created
instances initializing
instances initializing
instances initializing
instances initializing
```

The script will check the status of the instances and loop until it is ready. When the instances are ready, their id and IP address will be saved. A host.ini file is created containing IP address of the created instances, one of them is in [master] section others are in [slave] section. This is file is used in running Ansible playbook.

```
1  [master]
2  115.146.85.238
3  [slave]
4  115.146.85.238
5  115.146.85.130
```

3.2.4 Create Volumes

The instance size we are using is 'm2.medium', which has 30G disk involved. Since instances can break down and data could possibly be lost on the instance storage, we attach an additional 50G volume to each instance. We suppose that these volumes are much more persistent and can hardly be lost.

We have 250G volume available, 200G were allocated to instances, the rest 50G is used for backup. Since we only save CouchDB data in these volumes, and CouchDB clustering with sharding and replication is applied in our later deployment, we only need to create a snapshot of one of these volumes every 2-3 days manually. In this way, our data become much safer.

The use cases can be very complicated on volume creation and attachment.

If there are volumes existing, the newly created instances can attach to existing volume or create new ones. Which volume to attach depends on the certain requirements. A certain rule is needed to thoroughly automate.

In our script, we suppose very new instance should be attached to a new volume. If there aren't enough volumes, then no volumes are attached. It can be accomplished that new instances are attached to existing volumes since we already have the list of volume ids. However, we prefer to leave the volumes alone and don't do much with script at the risk of losing data and demonstrate this functionality on some empty Nectar project.

If there are not enough volume resource available, the volume operations are skipped with a warning message.

```
creating volumes
An error occurred (OverLimit) when calling the CreateVolume operation: VolumeSiz
eExceedsAvailableQuota: Requested volume or snapshot exceeds allowed gigabytes q
uota. Requested 50G, quota is 0G and 0G has been consumed.
```

Each new instance will try to create a new volume. If successful, the next step will be attaching the volume to the instance.

To attach an instance, instance id and volume id are required, which are returned by *create_instance* and *create_volume* calls respectively. One important thing is that the volume attaching operation requires that the instance is in running status. Attaching volume to initiating instances will fail. In our script, volume operations are all called till the instance status checking is done.

3.3 Ansible Playbook

3.2.1 Methodology

To ssh into a node, install everything and configure everything is quite tedious. When the system scale, the task becomes impossible. We used Ansible to automatically finish these tasks.

Ansible playbook is in yaml format, containing operation routines for different sets of instances. It can be invoked by Ansible-playbook command by indicating host.ini and yaml playbook.

Though a single additional command is not much effort, but the real problem in this method is that, instance creation and initiation takes a randomly long period. One has to run the instance script first, and check regularly to see if

it is done, and then fire the command to configure the instances. This process is not so elegant for automation.

We managed to integrate the Ansible command into the deployment script. The simplest way is to use sysexec in python to run the exact command that we type in the system prompt, but it will have less control in the parameter and settings. Therefore, we used the Ansible python package to invoke the Ansible playbook programmatically.

3.2.2 Usage

To change the operations on instances, edit the yaml playbook. We hard-coded the file name as 'config.yaml' in the script, so the file name can only be 'config.yaml' if running from our deploy script. An alternative way would be passing the file as parameter, but we preferred concision and simplicity to flexibility

The yaml playbook can be run independently by Ansible-playbook command, but you'll have to manually edit host.ini to specify which servers you want to manipulate. A private key to ssh into the server is also required. Upon initiating, each of our instance will have a default public key. We included the corresponding private key in the repository.

In our default routine, the yaml will be invoked along with the deploy script. Inventory file is also hard-coded as 'host.ini'. One thing changeable is the private key name. You can specify your private key name in credential.py as introduced in instance management section.

In the deploy script, we set a 30 seconds timer between instance operations and Ansible playbook run. The reason is when instance is initiated, ssh will be refused for a while for some reason. The exact time of this process is unknown, and we didn't find a way to check the status. 30 seconds is just enough for all our test runs.

3.2.3 Permissions

The inner implementation of Ansible is to ssh into an instance and do all defined tasks. We can only perform everything as user 'ubuntu' on instances. Some operations however, requires the root role to run. Therefore, we configured Ansible executor with become method and become role. This is equivalent to make Ansible do it's tasks with sudo privilege.

3.2.4 Inventory

Ansible needs the inventory to indicate which servers to run on, and some additional parameters. In our project, it is just the host.ini file generated when creating instances.

There is one special parameter in the inventory that never changes.

Ansible_ssh_common_args

This is added because system will ask for adding new server fingerprint which will interrupt the process. By setting it to

StrictHostKeyChecking=no

there will be no more prompt asking for permission, the whole process will be non-blocking.

There are two kinds of servers: master slave. All servers are used as tweet harvester and CouchDB cluster nodes. Master server is used as web server and tweet analyzing node.

Therefore, in our Ansible yaml file, there are two routines to take, one for every server, one for the master server. The routine for all server is invoked first, to install all basic requirement. After that master server routine is invoked to configure tasks specific to master node.

3.2.5 General Server Tasks

3.2.5.1 Authorized_Keys

Upon initiating, the instance has only one indicated *public_key*.To grant all members permis-

sion to ssh to the servers, their public keys must be stored in *~/.ssh/authorized_keys*.

Ansible copy module is used for this task. We have the 'authorized_keys' file in our local repository, it is used as the src file in an Ansible copy command.

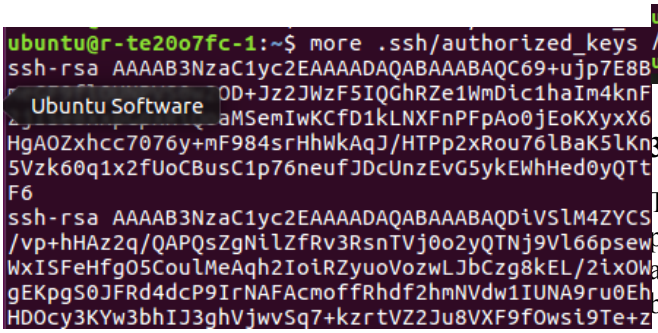
If other keys are needed, just add them to the local 'authorized_keys' file.

However, it is strongly recommended that this part is commented out, because if something in the key is wrong, nobody can ever log in the instance anymore. This part is used only for the need of my team members.

The task is successful

```
TASK [Copy members key to Authorized_keys]
changed: [115.146.85.238]
changed: [115.146.85.130]
```

Now the instance has multiple 'authorized_keys'.



3.2.5.2 Volume Mount

Each instance is attached with a volume. However the volume must be mounted to the file system and formatted before use.

We use Ansible to create a folder as mount point, format the volume and mount the volume to mount point.

First create a directory named 'database' as the mount point. This is accomplished by Ansible file module. Specify that file type is directory

```
- name: Copy members key to Authorized_keys
  copy:
    src: authorized_keys
    dest: .ssh/authorized_keys
```

Format the volume with Ansible filesystem module. Volume id is */dev/vdb/*, file system type is ext4

```
- name: fomat the volume
  filesystem:
    fstype: ext4
    dev: /dev/vdb
```

Mount the volume with Ansible mount module. Specify the state as mounted

```
- name: Mount volume to ~/database
  mount:
    path: /home/ubuntu/database
    src: /dev/vdb
    fstype: ext4
    state: mounted
```

After that, we can see the volume is mounted to 'database' directory

```
ubuntu@r-b5g6qjd5-2:~$ cat /proc/mounts |grep vdb
/dev/vdb /home/ubuntu/database ext4 rw,relatime,data=or
ubuntu@r-b5g6qjd5-2:~$
```

3.2.5.3 Install Software

The software needed in all our instances include python3, pip, CouchDB. Master node will need apache server in addition. Software is installed by Ansible apt module.

Sometimes the dpkg in ubuntu system is locked when the instance is created. This could be a result of some broken system initiating process, which will prevent all succeeding apt-get commands from getting the lock. To solve this, our script explicitly removes the dpkg lock before installing software.

The most complicated one is CouchDB installation, because, CouchDB package is not in default ubuntu source list. It must be added to the sourcelist file by Ansible lineinfile module.

Add CouchDB package site to ubuntu package list with Ansible lineinfile modyke.

```
- name: prepare install CouchDB
  lineinfile:
    dest: /etc/apt/sources.list
    line: deb http://apache.bintray.com/couchdb-deb xenial main
    insertafter: EOF
```

The CouchDB installation will prompt a pre-install configuration interactive panel, which will block the automation process. It can be skipped by setting the environment value of DEBAIN-FRONTEND: noninteractive. CouchDB is the very first package to install in our routine, so a update-cache is also performed during installation.

```
- name: install CouchDB
  environment:
      DEBIAN_FRONTEND: noninteractive
  apt:
    name: couchdb
    update_cache: yes
    force: yes
    allow_unauthenticated: yes
```

Likewise, pip3 can be installed on all node, and apache server can be installed on master node.

3.2.5.4 Install Dependencies for Python

Now that we have installed pip3, the dependencies can be installed with pip with Ansible pip module. The system default pip version is for python2. To run pip3, we need to explicitly specify the executable name

```
- name: install tweepy
  pip:
    name: tweepy
    executable: pip3
```

3.2.5.5 CouchDB Configuration

The CouchDB is in single node mode after installations. Our script can automatically configure CouchDB into cluster mode and nodes to the cluster. This involves two parts: first is to do basic setting, change storage directory and enable cluster mode on every instance; Second, the master is responsible for adding all other nodes into the cluster.

1. Change Node Name with Its IP address.

This must be done or the master node will not be able to reach members. It is achieved by Ansible lineinfile module to change one line in CouchDB vm.args config file. The IP address however, is different for each node, so we used the Ansible built in magic variables to get which exact node it is working on.

2. Change CouchDB storage to persistent volume storage.

We have attached a volume for each instance on deployment. These are designed to be storage for CouchDB, so that data will not be lost even the instances break down. The default storage of CouchDB is on instance storage.

The *database_dir* and *view_index_dir* are defined in CouchDB default.ini config file. We change these to the mount point directory we have created in former steps.

Then the current content of CouchDB must be moved to the specified directory*(~/database*). Since Ansible document says Ansible copy module does not support directory copy, we had to accomplish this by raw command.

CouchDB will not be able to access the new storage if the owner is not ‘couchdb:couchdb’. The script changes owner of storage with Ansible file module

1) Enable Cluster Mode

CouchDB provides config utilities to change mode by sending http request to database server. With Ansible uri module, the script will post an enable cluster request to DB server and check success on status code 201

2) Restart DB

To apply all changes. The DB server must be restarted. This is accomplished by a simple raw command

3) Master

Master node need two request for each node to add them into cluster. In former steps, the nodes are all enabled with cluster mode, within which a user-password must be set. Therefore the master node request must be authenticated in

Ansible URL module. Since the request must be sent for each member node, the script used Ansible built in magic variable {{groups}} to access the node names of all instances.

To finish the job, master node will send a single *finish_cluster* request to the DB server and create a database table named 'twitter' for further use

3.2.5.6 Apache Server Configuration

Upon installation the apache server will serve a default web site in */var/www*. The script will customize a new directory for our own web page. There aren't any new technical in this process. The script add new directory definition in *apache.conf* to allow server access, alter the existing default website serving directory to 'mysite' created at */home/ubuntu*, then restart the apache service.

At this point, we have a running cluster of several members and a server serving */mysite*

3.2.5.7 Harvester Deploy

The twitter harvester is deployed to every instance with different configurations. A harvester source code is all the same on all instances, but the 'twitter_key' file containing twitter access key and current node ID, distributed to each node will be different.

When the source code of harvester is on instances, Ansible command will launch it in background.

3.2.5.8 Tweet Analyze Script Deploy

The tweet analyze script takes about 5minutes to finish analyzing 300,000 tweets. It doesn't have to be called every now and then. So we used ansible CRON module to set a scheduled task on the master node. The task is to run this python code every hour. The result is saved in server readable path.

4. System Functionalities & Scenarios

In this section, MATLAB is used to produce general models and figures according to the data of the JSON file named "conclusions" which is produced by tweets analysis and different kinds of data from AURIN.

LATEX is used to produce different formulae related to corresponding scenarios according to the general models produced by MATLAB.

4.1 Sentiment Positive Ratio vs Personal Income

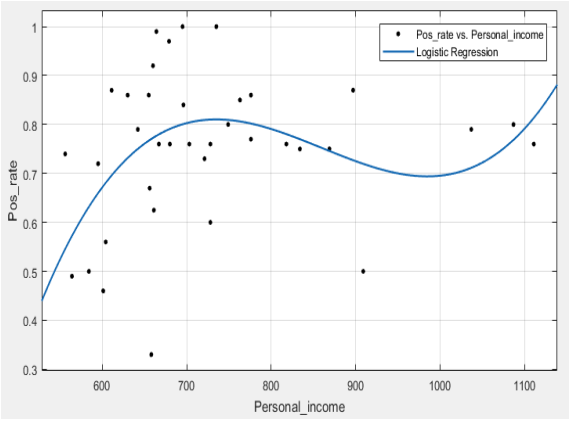


Fig 4.1-1

According to the polynomial regression curve (Fig 4.1-1) with personal income being X axis and positive rate being Y axis, there are unobvious correlations between those two sets of data.

When personal income is in range of $550 to $750, the percentage of positive sentiment tweets is keeping increasing and is growing very fast between $550 and $650. It indicates that in range of $550 to $750, people's happiness level is higher when they earn more money, especially in range from $550 to $650 per week. The result is not surprising; people's sentiment will be influenced by higher burden to support their life, and without this depressing reason, they will get happier. After $650 point, the curve start rising slowly, showing that with

enough income to support basic needs, the income is not dominant anymore. Meanwhile, the most tweets data we got are locating at areas where people' income is from $550 to $800 per week; this correlation should be reasonable and based on sufficient data. Nevertheless, the Y value of the curve slightly decreases when the X value hits 750 and finally increase again at the end of the curve around the point that X equals 1000. The correlation here is vague due to there is no enough data after the point that personal income per week equals 800. With limited number of data, the regression curve may not be accurate and will be overfitting to those single points. Besides that, when we did the polynomial regression analysis, we removed several points before X equals 550, which be assumed as outliers and will give the regression result a huge bad influence when they were included.

4.2 Sentiment Positive Ratio vs Education Level

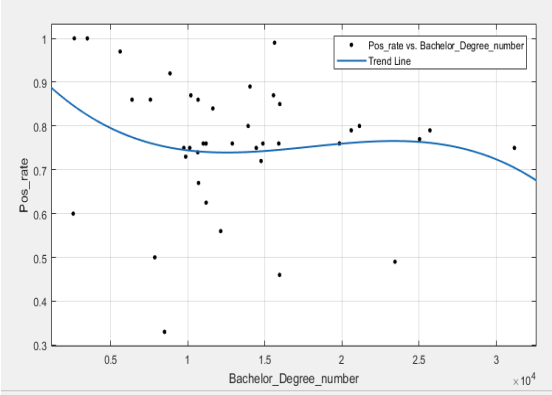


Fig 4.2-1

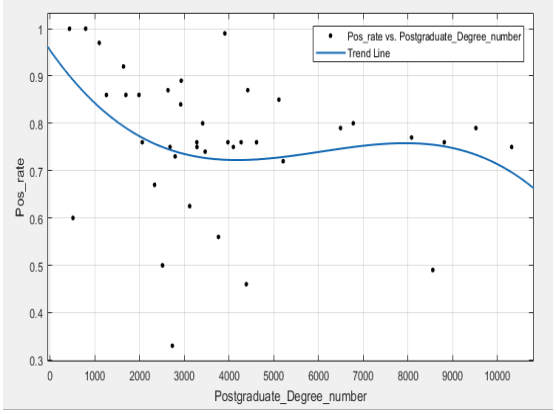


Fig 4.2-2

As those two graphs show, two education level cases (postgraduate degree and bachelor degree) has similar polynomial regression curves when they are analyzed with the ratio of positive sentiment tweets. From a global perspective, the main trend of rate of positive sentiment tweets is decreasing as the number of people have higher education level increases. The result here is worth to thinking about; why the people having higher degrees are more likely unhappy than people having simpler academic background? In our personal opinions, those people seem have critical thinking and higher perspective as well, they also realize more problems occurring around their life. Those things may give them more pressure than basic life matters and they may even have higher threshold of feeling happiness.

4.3 Spots Ratio vs Weekly Income

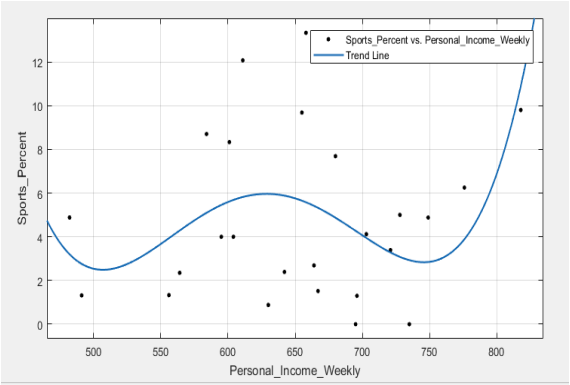


Fig 4.3-1

According to the Fig.4.3-1, when weekly personal income varies from 480 to 850, the percentage of tweets related to the sports is fluctuating. Between 510 and 630, the percentage is increasing slightly, which indicates that people prefer to pay more attention on sports as people have higher weekly income within the that range. Surprisingly, there is a slight decrease between 630 and 750 and have similar slope as the slight increase, people have that level of income is so busy and they do not have enough time to focus on sports. When people have more weekly income, over than 750, they can pay more attentions on sports as increased income. The highest percentage is located in the Whitehorse East, where far from the Melbourne City,

the residents there are more likely to care about the sports, compared with the situation in the Melbourne city, where has around 2 percentage of sports tweets and similar weekly income, which can show people who live in city have less time to focus on sports.

4.4 Entertainment Ratio vs Personal Income Weekly

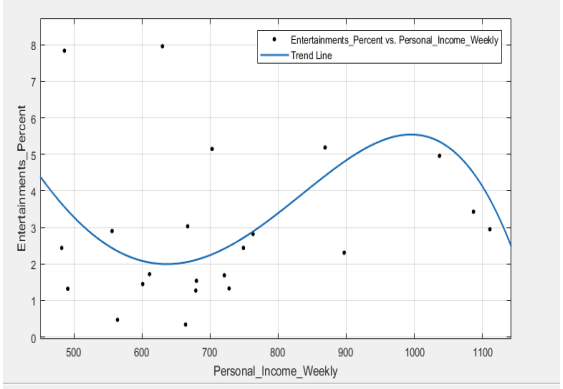


Fig 4.4-1

As shown in the Fig.4.4-1, when weekly personal income varies from 480 to 1200, the percentage of tweets related to the entertainment is fluctuating around 3.5, showing worker in Melbourne are willing to spend their remaining time on entertainment and there is no direct correlationship between entertainment post and how much they can earn weekly. However, by observing the slight change between 640 and 1000, people would like to pay more attention on entertainment as the income increased, and when the weekly income is over than 1000, people might focus more on the other things instead of entertainment. According to the observation of clustering of points in the Fig.4.4-1 , almost tweets related to the entertainment are distributed between 480 to 800, which indicates that people who have not high income are willing to spend their own time on entertainment.

4.5 Sentiment Positive Ratio vs Median Age

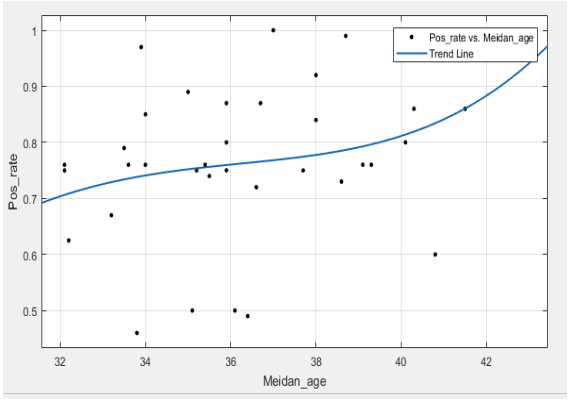


Fig 4.5-1

According to the diagram, from an overall perspective, the elder the people are, the higher positive ratio is. This possibly because when a person has experienced more, his outlook on life has become more mature and he doesn't fuss about many trifles in his life, which makes him know negative attitude does no good to anything.

In addition, we can see that the people who are in the suburb whose median age is between 33 and 39 seem to be more active to present their positive attitude than others. People in this range of age are usually the main power of the society, which means most of them are familiar with the social forums and pleasant to express their viewpoints on them.

Moreover, the positive ratio of people who have positive attitude is usually over 70%. This means the people in Melbourne are usually positive. Melbourne has been named the world's most liveable city by The Economist several times, receiving a perfect score for healthcare, education and infrastructure, which maybe lead to this result.

4.6 Sports Ratio vs Median Age

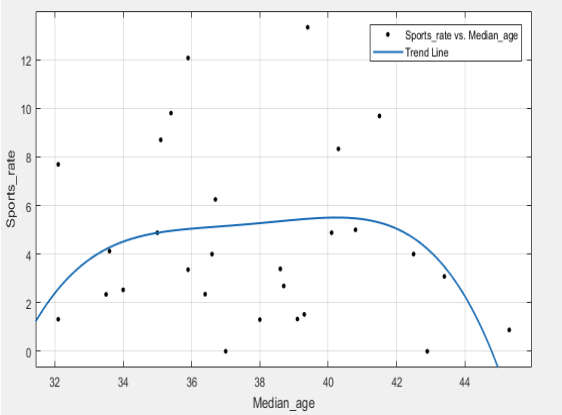


Fig 4.6-1

From the figure, the people whose age is between 34 and 41 give more information about sports on Twitter and people whose age is under 34 or higher than 41 put less focus on sports.

Some assumptions can be proposed.

People whose age is under 34 are usually considered as young people relatively. Therefore, many of them are students that they have to focus on their study and don't have much time on sports. Many of them can be new employees, which means they just come into the market and have heavy stress on making progress to make good impressions to their managers or bosses. Besides, blending in with their workplaces may also occupies their time.

Sport is often considered as strenuous exercise. When a person gets older, he usually moves his attention to other areas, such as planting and painting. We can see the trend starts to go down where the median age is over 41.

Most people whose age is between 34 and 41 have stable jobs and they may start to pay more attention to their health in middle age. Exercising are proved to be an effective way to reduce the rate of getting a cancer, which maybe a cause why people in this range focus more on sports.

4.7 Food&Drink Ratio vs Personal Weekly Income

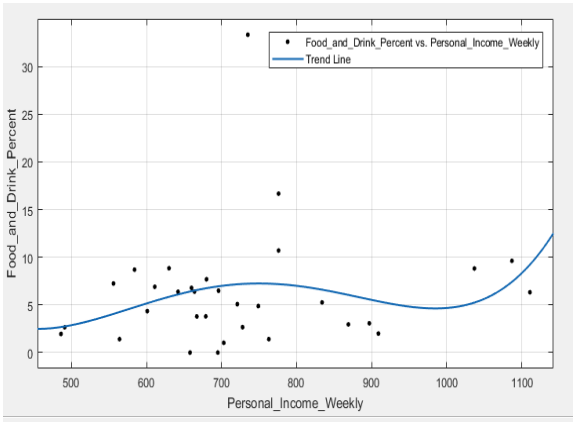


Fig 4.7-1

As illustrated in the Fig.4.7-1, when weekly personal income varies from 480 to 1200, the trend of percentage of tweets related to food and drink is increasing slightly from 3 to 12.5. Between 480 to 580, there is a big increase of percentage of food&drink tweets from 3 to 7, showing that people get the higher weekly income focus more on the discussion of food and drink. Conversely, when people get weekly income between 580 and 900, they are likely to spend less time on talking about the food and drink as increased income. By observing the clustering in the Figure, almost points are clustering between 580 and 750, which indicates that discussion on food and drink is much more popular within the group who have income varies from 580 to 750. The percentage of food&drink tweets is increasing quickly is increasing from 3 to 12, which shows that people pay more attentions on food and drink as the more income and they have more salary to pay for food and drink they want, which is same as the increase between 480 and 580. In addition, because of over-fitting issue, the highest percentage, 33.33 is removed, but it is located in Manningham East, the people here they are really eager to discuss food and drink.

5. Usage

5.1 Preparations

Before deployment several setting must done manually.

A key pair must be created for instance creation, which allows one administrator to access with ssh. Create security groups , to open 22 port for ssh, 80 port for http request, 5984 port for CouchDB cluster.

In our deployment script, instances are created with security group 'default' and 'SSH'. For creating instances in new Nectar group, *22/80/5984* must be added in these two groups.

5.2 Deployment Script

The deployment script creates instances and setup the environments. But, there might be interruptions between instance creation and ansible playbook run. This may happen when an expired server fingerprint exists in you system. Type yes if this happen.

After the deploy script finishes, CouchDB cluster and server will all be ready. Harvester will be running in background and the web page can be accessed by IP address of the master node from browser. Tweet analyze script is scheduled as CRON job in background.

5.3 Analysis Script

The analysis program is run by scheduled task. Deploy script will setup the schedule automatically. It doesn't need to be invoked explicitly. However, if you wish to get a new result at once, just run the tweetAnalyze.py script. In the *__main__* part of the script, customized keywords can be defined in the dictionary structure.

5.4 Data Import

The front page load GeoJSON data for display. The python script import *Geo.json* is used to transform shapefile to GeoJSON and add to database. It is invoked with one argument indicating the .shp file name.

More statistic data can be shown on website, by saving the GeoJSON style JSON data as downloaded from AURIN in GeoJSON directory and add an option in index page to select box element.

6. Results & Discussion

6.1 Nectar Cloud

The experience to operate most of the tasks remotely is very good. Since Nectar can be very slow to load the views, it is nearly impossible to do heavy deployment on the dashboard.

Good side is that almost all process we needed have a remote utility to manipulate. Bad side is that the UI of Nectar dashboard is too slow. The reason might be that loading these data about a view takes very long. But when remote API can actually get a summary of instances, volumes etc within a very short time, it is not reasonable for UI to take so long. This could be improved by using pre-loading or caching of unchanged status strategies.

We worked out a very smooth automatic deployment script. It can be further improved by adding parameters for the user to decide customized deploy process.

6.2 CouchDB

CouchDB guarantees availability and partition tolerance at the cost of occasional temporal inconsistency. It has eventually consistency by version control.

CouchDB MapReduce facility is a good tool to filter the data, and transform data into wanted format. However, the functionality is very limited, and debugging the scripts is painful.

We were considering using CouchDB map function to do sentimental analysis to each tweet, but end up find out external packages are not supported in view JavaScript.

There are tools like in grunt to compile and push view definitions to design document with commands. We also worked out own python version to do the same job, but still not quite flexible to debug. For example, in our original design, geometries are pushed to DB separately, then use CouchDB list function to filter geometries in a certain area and combine into a complete GeoJSON. However, we failed to make the view return a standard http response which our front page can access.

The limitation in MapReduce function is frustrating. It's even confusing for experienced JavaScript programmer, since what one used to know would not be working in this circumstance. Switching to more specialized framework providing MapReduce feature like Hadoop might be more rational.

6.3 Error Handling

During the tweets harvesting, errors is handled by a try-except process that when error occurs, it will be recorded in the log file stored in the node. In this way, the tweet harvesting process is not interrupted by any occurred errors. Meanwhile, at the stage of writing data into the database, there is another try-except mechanism that ensure the data storage process is not disturbed if any error occurs. The error message will also be kept in the log file. By reading the log file, one can have a clear idea of the exact process where the error occurs as well as when and why.

The fault-tolerant ability of the tweets harvesting process is guaranteed by the multi-nodes parallelization design. If one node fails, the other nodes does not get affected and will continue harvesting tweets as these nodes are physically separated. If one node stop collecting tweets due to some kind of failure, the log file will stop recording as well, so that the system manager can notice it by regular inspection and locate the position where a failure happens.

6.4 Scalability

The scalability of the tweets harvesting system is guaranteed by the parallel design, too. Since every node is physically separated and have no influence on each other, if more nodes are added, it can be achieved by simply assigning node IDs to new node and changing the total number of nodes in each tweet harvester to allocate even workloads to them. The subsequent data analysis and visualization are also not affected because both of them are done in the master node alone. The storage can easily be scaled by adding more nodes to the cluster.

7. Conclusions

In this project, we implemented a social network analyzing system on cluster environment. We come to the conclusion that deployment cost complexity can be constant, not growing proportionally to cluster scaling. Processes can be distributed with some synchronization at the loss of efficiency to balance load, parallel resource consuming tasks and promote robustness of system. Storage can be arranged in several ways to cope with potential need growing extra data size and improve availability and fault tolerance.

Appendix

1. Git repository address:

https://github.com/yjclegend/ASMA2018

2. YouTube demo:

https://www.youtube.com/watch?v=03y_i1ZzCGo&feature=youtu.be

Reference

[1] Number of monthly active Twitter users worldwide from 1st quarter 2010 to 4th quarter 2017 (in millions), Retrieved from 'https://www.statista.com/statistics/282087/number-of-monthly-active-twitter-users/'

[2]Sensis Pty. Ltd. (June, 2017) Sensis Social Media Report 2017, Retrieved from 'https://www.sensis.com.au/asset/PDFdirectory/Sensis-Social-Media-Report-2017.pdf'

[3] Stefanidis, A., Crooks, A., & Radzikowski, J. (2013). Harvesting ambient geospatial information from social media feeds. GeoJournal, 78(2), 319-338.

[4] MacEachren, A. M., Robinson, A. C., Jaiswal, A., Pezanowski, S., Savelyev, A., Blanford, J., & Mitra, P. (2011, July). Geo-twitter analytics: Applications in crisis management. In 25th International Cartographic Conference (pp. 3-8).

[5] Ratkiewicz, J., Conover, M., Meiss, M. R., Gonçalves, B., Flammini, A., & Menczer, F. (2011). Detecting and tracking political abuse in social media. ICWSM, 11, 297-304.

[6] Chae, J., Thom, D., Bosch, H., Jang, Y., Maciejewski, R., Ebert, D. S., & Ertl, T. (2012, October). Spatiotemporal social media analytics for abnormal event detection and examination using seasonal-trend decomposition. In Visual Analytics Science and Technology (VAST), 2012 IEEE Conference on(pp. 143-152). IEEE.

[7] https://portal.aurin.org.au

[8] http://www.tweepy.org

[9]Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. Communications of the ACM, 51(1), 107-113.

[10] Ma, Z., & Gu, L. (2010, November). The limitation of MapReduce: A probing case and a lightweight solution. In Proc. of the 1st Intl. Conf. on Cloud Computing, GRIDs, and Virtualization (pp. 68-73).