COMP90024 Cluster and Cloud Computing

Semester1-2020 Assignment2

Alcohol Tweets and Australian Cities-Analytics on the Cloud

Team 5

Student Name	Student ID	Email
Ruiqi Zhu	939162	ruiqiz3@student.unimelb.edu.au
Zhengyang Li	952972	zhengyangl3@student.unimelb.edu.au
Jianxin Xu	1014840	jianxinx1@student.unimelb.edu.au
Qiuxia YIN	1017231	qiuxiay@student.unimelb.edu.au
Fang Qu	1070888	fqqu@student.unimelb.edu.au

CONTENTS

1 Introduction

Alcohol takes a significant place in Australian culture and is consumed in a wide range of social circumstances. According to Australian Institute of Health and Welfare, the majority of Australians aged 14 years and over consume alcohol. Alcohol consumption can be said to be a reflection of social and cultural norms [1] and the relationship between alcohol and social behaviours is an enduring topic worldwide.

Traditionally, alcohol usage is detected with the use of surveys or interviews with a limited number of people [2]. But with the rapid and vast adoption of social media, a tremendous amount of data is generated every second, from which significant information and knowledge can be retrieved. In this project, the big data power of Twitter, a ubiquitous microblogging service with 4.6 million subscribers in Australia in 2019 [3], is leveraged as the source of a large amount of data for us to dig out the correlations between alcohol-related tweets and some demographic and behavioural characteristics in Australia cities.

Specifically, a cloud-based analytic system is built on Melbourne Research Cloud (cited as MRC hereafter), and it consists of two harvesters for tweets collection, a three-node CouchDB cluster for data storage, an analytics module for data analysis and a front-end website with ReSTful design for visualization of the results under several scenarios. Over 5 million tweets have been collected, and real-time tweets are being continuously collected and corresponding analysis outcomes are updated on the website. Scalability and automation are also featured in the system.

Demos, codes and website can be reached by the following links:

- Project GitHub repository:

  https://github.com/cxhugh/COMP90024_A2
- Ansible deployment demo:

  https://youtu.be/a_tzDuSSN2A
- Web interface demo:

  https://youtu.be/WA4PTjUQX_Y
- Web frontend server address (only accessible from Unimelb Intranet）

  Xxxxxxxxxxxxxxxx

2 System Design and Deployment

2.1 Architecture and design

The overall architecture of the system is illustrated in Fig 2-1. The system is deployed on MRC, with four instances running based on Ubuntu 18.04. One harvester and one CouchDB node are installed on instance1 and instance2. Instance3 hosts another CouchDB node and a data analytics module. And these three CouchDB nodes are set up into a cluster mainly for data availability. Instance4 is used as the web server.

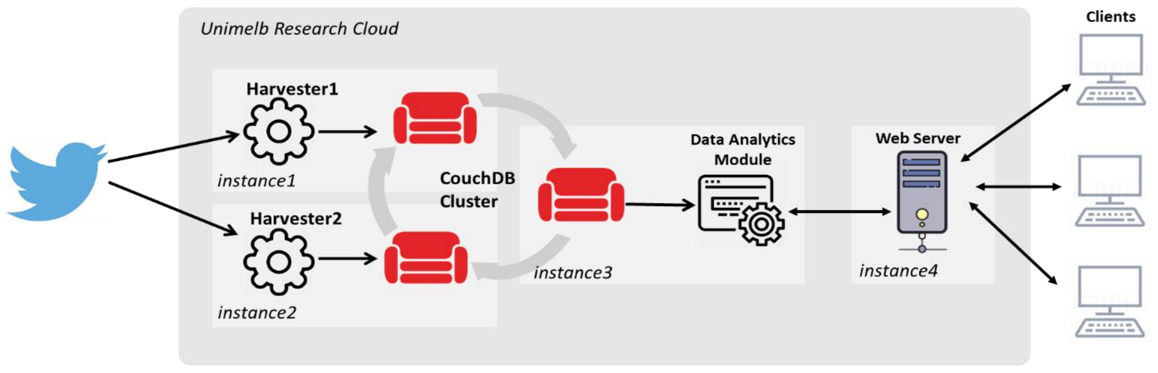


Fig2-1 System Architecture

Table 2-1 shows the resources allocated to the instances. Taking consideration of the availability of the VCPUs, RAM and volumes, together with functions and potential hardware requirements of each instance, all instances are assigned to the same VCPUs and RAM, and instances which are used for data storage are allocated with larger size of volumes. Volume scalability is also considered, with 45G being used as backup.

Instance NO.	Instance Name	VCPUs	Volume size	RAM
1	Harvester 1	2	65	9GB
2	Harvester 2	2	65	9GB
3	dbServer	2	65	9GB
4	webServer	2	10	9GB

Table 2-1 Instance Resource Allocation

The functionalities of the system main components can be summarised as below:

- Twitter Harvester

  A substantial amount of data is required for our analysis, and thanks to Twitter's free public API, we are able to deploy both search API and stream API to collect historical data and real-time data. By applying nohup shell command, we are able to run the scripts in the background on our server endlessly. Given the data retrieved from Twitter, we did not save all the information from one tweet. We saved 'id_str' as a unique identifier for each tweet to avoid duplication. Geojson information is used to determine which state and suburb is tweeted from, which helps us do analysis for each scenario.

- CouchDB Cluster

  CouchDB is a widely-used open-source document-oriented NoSQL database and it is deployed in this project for big data storage and processing. CouchDB can be said to be the "backbone" component in the system: data from the harvesters is stored into CouchDB, and its embedded MapReduce capabilities are utilized for data analysis; the front-end website also interacts with CouchDB for data analysis outcomes for visualization.

As for the design, clustering instead of replication is chosen to enhance the performance of the system. In CouchDB clusters, each database (and secondary index) is split into shards and is stored on every node in the cluster, and data is still available even if one node is unavailable. With more than one harvester working on different instances simultaneously, deploying a cluster to allow a single logical database server on all three instances can provide better data redundancy and higher availability.

- Data Analytics Module

  After the harvested twitter data have been preprocessed and stored into the database, we can do some interesting analysis along with the collected Aurin data. MapReduce function provided by CouchDB will be used to carry out the analysis. Multiple views will be created based on different scenarios and analysis scope. The view query results well then be further processed and saved to the database, which can be later consumed by the front end to display visual results. As our twitter harvester will be running all the time to crawl real-time streaming data, the query result of the view will be updated every 30 minutes to obtain the newest information.

- Web Server

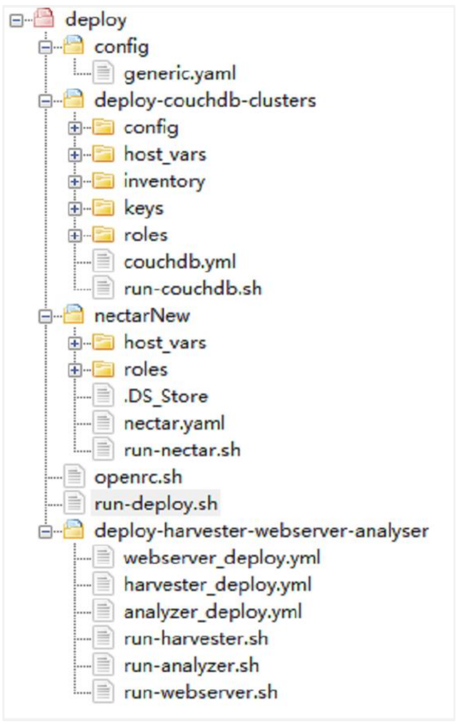
  A front-end, implemented with HTML, CSS, Javascript, and a backend of a web-based visualization application is deployed on an independent instance by using Flask and Nginx. The web application uses pre-defined ReSTful APIs to access the data that need to be shown on the website from one node of the CouchDB cluster, which is “dbServer” in our case.

2.2 System deployment

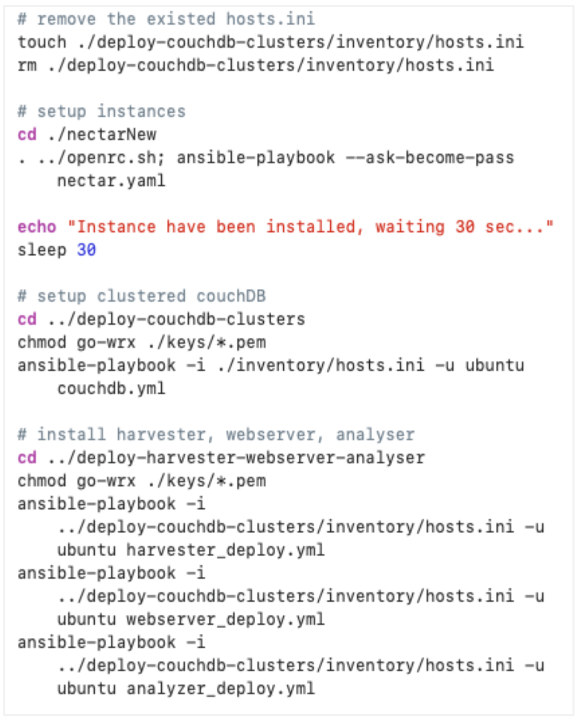
The entire system is required to have scripted deployment capabilities; thus, automation is the major concern in the deployment. To that purpose, ansible is utilized to create instances on MRC and automatically set up harvesters, CouchDB cluster and the web server.

Ansible is an open-source software tool that provides simple but powerful automation. During our project, we found it very useful in making the systems and applications easier to deploy and scale with ad-hoc commands and easy to read/edit playbooks. Besides, it is declarative and not procedural as once the final state is defined, it takes all necessary steps to fulfil the task, which means that playbooks can be executed many times and only necessary steps are taken.

Fig2-2 illustrates the overall structure of the playbook (repository tree of the project) for installing the whole system all at once. After *run-deploy.sh* is executed, the script will automatically complete the applications installation tasks as the instance info (mainly the IP addresses) is stored to the *config/generic.yaml, host_var* and *host.ini* docs under each application’s implementation script, thus avoiding any manual input.



```
deploy
  config
    generic.yaml
  deploy-couchdb-clusters
    config
    host_vars
    inventory
    keys
    roles
    couchdb.yml
    run-couchdb.sh
  nectarNew
    host_vars
    roles
    .DS_Store
    nectar.yaml
    run-nectar.sh
  openrc.sh
  run-deploy.sh
  deploy-harvester-webserver-analyser
    webserver_deploy.yml
    harvester_deploy.yml
    analyzer_deploy.yml
    run-harvester.sh
    run-analyzer.sh
    run-webserver.sh
```



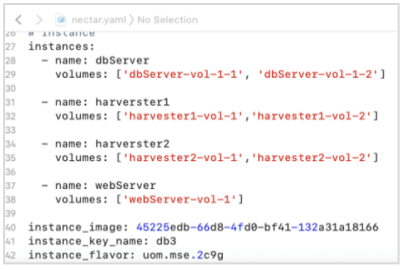
```
# remove the existed hosts.ini
touch ./deploy-couchdb-clusters/inventory/hosts.ini
rm ./deploy-couchdb-clusters/inventory/hosts.ini

# setup instances
cd ./nectarNew
. ../openrc.sh; ansible-playbook --ask-become-pass
    nectar.yaml

echo "Instance have been installed, waiting 30 sec..."
sleep 30

# setup clustered couchDB
cd ../deploy-couchdb-clusters
chmod go-wrx ./keys/*.pem
ansible-playbook -i ./inventory/hosts.ini -u ubuntu
    couchdb.yml

# install harvester, webserver, analyser
cd ../deploy-harvester-webserver-analyser
chmod go-wrx ./keys/*.pem
ansible-playbook -i
    ../deploy-couchdb-clusters/inventory/hosts.ini -u
    ubuntu harvester_deploy.yml
ansible-playbook -i
    ../deploy-couchdb-clusters/inventory/hosts.ini -u
    ubuntu webserver_deploy.yml
ansible-playbook -i
    ../deploy-couchdb-clusters/inventory/hosts.ini -u
    ubuntu analyzer_deploy.yml
```

Fig2-2 Overall playbook structure illustration

Fig 2-3 workflow of ansible playbook

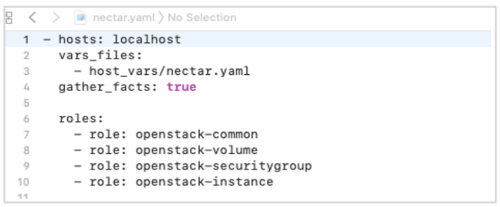
2.2.1 Create instances

Our first step is to create four instances on MRC. As shown in Fig 2-4, four instances are created by a single ansible script, with four roles implemented:



nectar.yaml 〉No Selection

```
# instance
instances:
  - name: dbServer
    volumes: ['dbServer-vol-1-1', 'dbServer-vol-1-2']

  - name: harverster1
    volumes: ['harvester1-vol-1','harvester1-vol-2']

  - name: harverster2
    volumes: ['harvester2-vol-1','harvester2-vol-2']

  - name: webServer
    volumes: ['webServer-vol-1']

instance_image: 45225edb-66d8-4fd0-bf41-132a31a18166
instance_key_name: db3
instance_flavor: uom.mse.2c9g
```



nectar.yaml 〉No Selection

```
- hosts: localhost
  vars_files:
    - host_vars/nectar.yaml
  gather_facts: true

  roles:
    - role: openstack-common
    - role: openstack-volume
    - role: openstack-securitygroup
    - role: openstack-instance
```

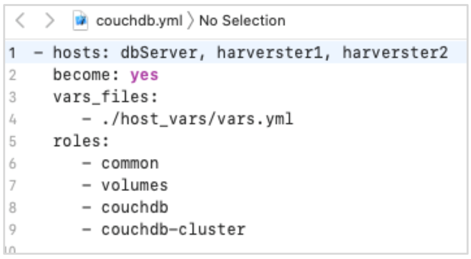
Fig 2-4 instance creation and roles

- Openstack-common: install pip and openstacksdk
- Openstack-volume: create volumes
- Openstack-security-group: open ports for applications.
- Openstack-instance: create instances and transmit the IP addresses into *config/generic.yaml, deploy-couchdb-clusers/inventory/host.ini* and *deploy-couchdb-clusers/host_vars/vars.yml*

After the instances are created, a 30s' break is taken to make sure the instance is on and the IP addresses are transmitted to CouchDB host_vars and host.ini for the next step.

2.2.2 Set up CouchDB cluster

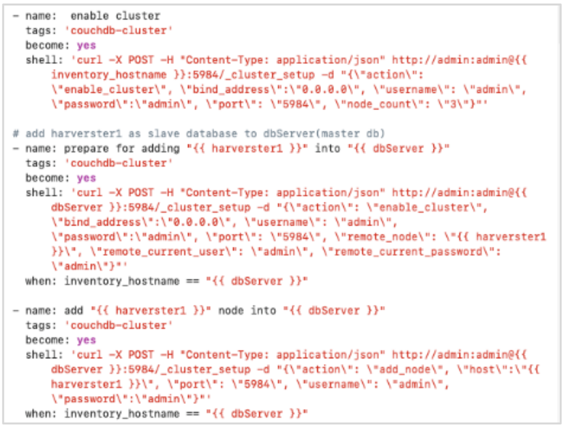
The second step that run-deploy.sh takes is to change the working directory and start to install CouchDB and then CouchDB cluster.



couchdb.yml 〉 No Selection

```
- hosts: dbServer, harverster1, harverster2
  become: yes
  vars_files:
    - ./host_vars/vars.yml
  roles:
    - common
    - volumes
    - couchdb
    - couchdb-cluster
```

Fig 2-5 CouchDB cluster installation roles



```
- name:  enable cluster
  tags: 'couchdb-cluster'
  become: yes
  shell: 'curl -X POST -H "Content-Type: application/json" http://admin:admin@{{
      inventory_hostname }}:5984/_cluster_setup -d "{\"action\":
      \"enable_cluster\", \"bind_address\":\"0.0.0.0\", \"username\": \"admin\",
      \"password\":\"admin\", \"port\": \"5984\", \"node_count\": \"3\"}"'

# add harverster1 as slave database to dbServer(master db)
- name: prepare for adding "{{ harverster1 }}" into "{{ dbServer }}"
  tags: 'couchdb-cluster'
  become: yes
  shell: 'curl -X POST -H "Content-Type: application/json" http://admin:admin@{{
      dbServer }}:5984/_cluster_setup -d "{\"action\": \"enable_cluster\",
      \"bind_address\":\"0.0.0.0\", \"username\": \"admin\",
      \"password\":\"admin\", \"port\": \"5984\", \"remote_node\": \"{{ harverster1
      }}\", \"remote_current_user\": \"admin\", \"remote_current_password\":
      \"admin\"}"'
  when: inventory_hostname == "{{ dbServer }}"

- name: add "{{ harverster1 }}" node into "{{ dbServer }}"
  tags: 'couchdb-cluster'
  become: yes
  shell: 'curl -X POST -H "Content-Type: application/json" http://admin:admin@{{
      dbServer }}:5984/_cluster_setup -d "{\"action\": \"add_node\", \"host\":\"{{
      harverster1 }}\", \"port\": \"5984\", \"username\": \"admin\",
      \"password\":\"admin\"}"'
  when: inventory_hostname == "{{ dbServer }}"
```

Fig 2-6 adding node to dbServer to form a cluster

Four roles are executed:

- common: install dependencies and update pip and populate proxy setting
- volumes: mount the volumes, assign the directory for database storage; this step is very important; once done, the data stored in the volume is safe and can be reloaded into a rebuilt cluster when the initial cluster crashes or is re-installed.
- couchdb: single CouchDB is installed on each of the three nodes
- couchdb-cluster: set up a cluster of the single nodes. Specifically, dbSever is treated as the first node and other nodes are added into a group with dbSever by *curl -X POST -H…* (shown in Fig 2-6). In this way, Scalability of CouchDB cluster can be easily achieved by making some changes of the ansible scripts.

2.2.3 Deploy Harvester/Analyzer/Webserver

Having the instance and Couchdb cluster been built, the following three applications can be easily deployed to the instances:

- Deploy harvester

  A single playbook harvester_deploy.yml is designed to install the harvester automatically on any of two instances, one for search API and another one for stream API. The script performs the following tasks in order: copy harvester function package to cloud server, install dependencies such as couchdb, mpi4py, tweepy library and use nohup command to run the harvesters in the server background.

- Install data analysis module

The process of deploying is similar to harvester, we created an ansible script, analyzer_deploy.yml to finalize this task. The script uploads three packages that are needed for the analysis module, and then starts the module to generate data views every 30 minutes.

- Deploy web server

There are 5 steps in script to make the web app available on cloud server, including upload the web package to the server, install dependencies, update Nginx config file to keep listening to port 80, start Nginx by giving config path using “nginx -c /home/ubuntu/web/nginx.conf”. And finally, run the flask app to activate the web server.

2.3 Melbourne Research Cloud (MRC) review

Melbourne Research Cloud (MRC) is a cloud infrastructure that provides free on-demand computing resources to researchers and students at the University of Melbourne. The functionalities of MRC are similar to its commercial counterparts like Amazon Web Services, Microsoft Azure, and the services enables researchers/students to get access to scalable remote computational resources to store, access and process large volumes of data or deploy other applications such as web-hosting.

While creating and managing the environment on MRC, there are several aspects that needs special attention. First, choose a suitable image. Some images may require special version of software like Python. In our project, we used the pre-provided ubuntu 18.04. Second, make sure the volume is correctly mounted to the instance in order to permanently keep the processed data. Third, all ports that will be used in the projects should be open in security group properly.

As the platform on which our system is built, MRC has proven to provide robust IaaS cloud computing services. There are many advantages we have observed as well as some disadvantages or to be more precise, challenges, especially for the new users.

Advantages:

- Availability: researchers/students with eligibility can build virtual machines to take use of the computing resources the moment they need without planning in advance and purchase. Users are not responsible for the maintenance of the system, and it’s easy to rebuild or recreate servers if there is any issue in the current sever that might be take longer to track and fix. It is ideal for testing configurations like in our project.
- Flexibility and Scalability: it is dynamic to manage the environment. With settings of image, snapshot, security-groups etc, users can scale easily to meet the particular requirements. Being able to choose from various pre-provided images and flavours or creating one’s own also make it flexible to deploy a specific environment. Backup and sharing are also easy to realize by cloning the computation environment.

- Accessibility: firstly, MRC is accessible from anywhere of the world, which is especially helpful this year for the ones who have to work overseas due to the COVID-19 lockdown. Secondly, MRC can be accessed over API. We can use scripts to deploy and manage the virtual machines, which is a great advantage to make the whole work easier and more efficient.

Disadvantages/challenges:

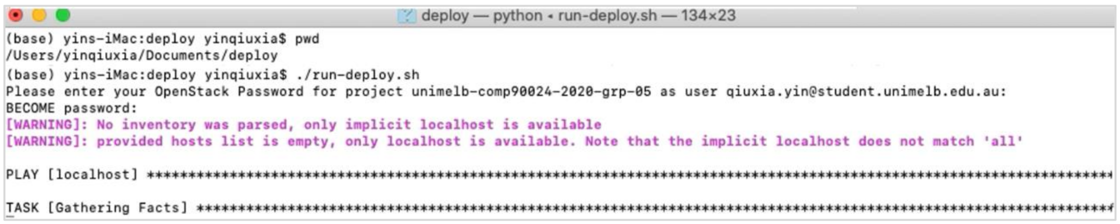
- Limited public IP resources: we are only allocated to private IP addresses; therefore, we have to do most of the operations via VPN, which is not stable and sometimes can be inaccessible. Private IP addresses also makes some features more difficult if not impossible to utilize, e.g. replication from one CouchDB to another (not clustering).
- It requires some extra attention to make good use of the platform and deploy applications. For example, in order to correctly use the volume, several steps are to be taken. After the volume is created and attached to the instance, it is very important to format and mount the volume, and in order to prevent missing the volume (not the data) after reboot, further commands are required. It's very likely for new users to miss some steps and get stuck.
- Debugging is tricky. Although info/warn/error/debug logs and action logs are available, it is still not enough for new users to figure out causes of some problems.
- Potential risk of instance failure and security vulnerability. Like other on the cloud system, the virtual machines might fail and then lead to loss of data. Thus, it is vital to ensure constant data backup offsite, which requires extra resources. Besides, as everything is run over the internet, it is possible that the data be compromised or leaked.

2.4 User guide for one-key deployment

Operational steps of deploying the whole system are shown as below:

1) Download the OpenStack RC file and OpenStack API password from MRC; everyone has his/her own RC file. The file is named as openrc.sh and saved under the first-level directory in the repository (refer to Fig 2-2)

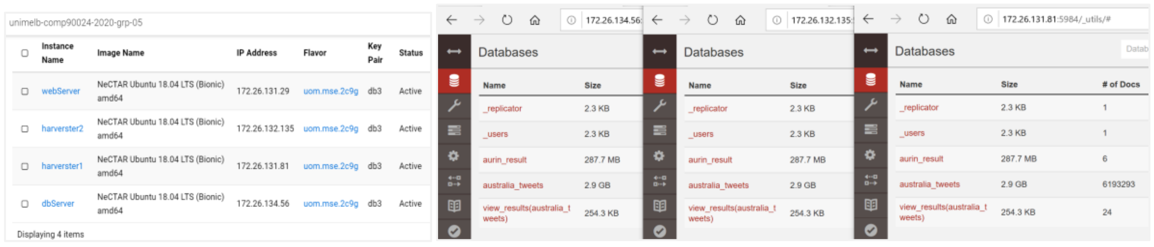
2) Use ansible to automatically deploy the entire system: open the terminal and set the folder with the scripts as the working path, change *run-deploy.sh* as executable and then run the script by commanding *./run-deploy.sh*, input the OpenStack API password and localhost password (shown in Fig 2-7).



```
(base) yins-iMac:deploy yinqiuxia$ pwd
/Users/yinqiuxia/Documents/deploy
(base) yins-iMac:deploy yinqiuxia$ ./run-deploy.sh
Please enter your OpenStack Password for project unimelb-comp90024-2020-grp-05 as user qiuxia.yin@student.unimelb.edu.au:
BECOME password:
[WARNING]: No inventory was parsed, only implicit localhost is available
[WARNING]: provided hosts list is empty, only localhost is available. Note that the implicit localhost does not match 'all'

PLAY [localhost] ************************************************************************************************************

TASK [Gathering Facts] ******************************************************************************************************
```

Fig2-7 commands to run the script

3) The outcomes: the instances can be seen on *dashboard.cloud.unimelb.edu.au* and all three nodes of the CouchDB clusters are accessible with same databases on each node and #o of Docs continuously growing, indicating that the harvesters are working as well. And the webserver can be reached at *172.26.131.29.*



unimelb-comp90024-2020-grp-05

Instance Name	Image Name	IP Address	Flavor	Key Pair	Status
webServer	NeCTAR Ubuntu 18.04 LTS (Bionic) amd64	172.26.131.29	uom.mse.2c9g	db3	Active
harverster2	NeCTAR Ubuntu 18.04 LTS (Bionic) amd64	172.26.132.135	uom.mse.2c9g	db3	Active
harverster1	NeCTAR Ubuntu 18.04 LTS (Bionic) amd64	172.26.131.81	uom.mse.2c9g	db3	Active
dbServer	NeCTAR Ubuntu 18.04 LTS (Bionic) amd64	172.26.134.56	uom.mse.2c9g	db3	Active

Displaying 4 items

172.26.134.56:

Databases

Name	Size
_replicator	2.3 KB
_users	2.3 KB
aurin_result	287.7 MB
australia_tweets	2.9 GB
view_results(australia_tweets)	254.3 KB

172.26.132.135:

Databases

Name	Size
_replicator	2.3 KB
_users	2.3 KB
aurin_result	287.7 MB
australia_tweets	2.9 GB
view_results(australia_tweets)	254.3 KB

172.26.131.81:5984/_utils/#

Databases

Name	Size	# of Docs
_replicator	2.3 KB	1
_users	2.3 KB	1
aurin_result	287.7 MB	6
australia_tweets	2.9 GB	6193293
view_results(australia_tweets)	254.3 KB	24

Fig2-8 instances and CouchDB cluster of grp-05

4) Every segment in the system can be deployed separately: execute the run-xxx.sh under the folder with its name, e.g., *run-couchdb.sh* under *deploy-couchdb-clusters* to install CouchDB Cluster.

3 Twitter Harvester

3.1 Harvester and topic overview

The harvester utilizes two different crawlers to extract tweets: Search API and Stream API provided by Twitter. They are implemented with the support of tweepy(cite) package and deployed on two different cloud servers. Nohup shell command is used to let the harvesters run in the background of the server instance to continuously fetch tweets and save to CouchDB database.

The harvester has a data cleaning process to remove redundant information retrieved by APIs and keep the important keys, namely unique ID, original text, user's screen name, and coordinates information.

As scenarios needed, our harvesters aim to find the tweets which relate to the alcohol topic. Thus, we created a list of drinking-related terms by searching from some web searches (the top brands of liquors/wines/beers etc), and some online dictionaries which list dozens of words/slangs associated with drinking alcohol.

Given that users usually do not tend to share their detailed location when tweeting, it is quite hard to find coordinate-enabled tweets. To solve this issue, we restricted search API to search tweets based on some of the major cities in Australia and stream API to find tweets if the place bounding box is not null. We also tried to work out problems of duplicate data, harvester shut down and recovery, and twitter API rate limitation, details will be given in section 5.2.

Until 26th May, the harvester had collected roughly 6 million tweets data. A more detailed description of how we designed our harvester system will be delineated in the following sections.

3.2 Stream API

Given twitter's official document, there is no rate limit for stream API, we designed our scripts to be able to extract new tweets “forever”. We put our scripts onto our harvester server and allowed it to keep the HTTP connection persistently. We restricted the tweets to have either coordinate information or having roughly bounding box information so we can have a rough idea on where these tweets come from. Per Tweepy's official document, we cannot simultaneously track topic and location, so we had two individual scripts to find tweets:

1) Streaming globally to find tweets that are relevant to our topic by restricting topic filters. If this tweet contains coordinates or bounding box information, we will figure out whether this tweet is posted in Australia. If it is indeed from Australia, we will first label the location tag as '1’ for Australian tweets ('0' otherwise).

2) Streaming locally within Australia's bounding box to find all sorts of topics that contain coordinates or bounding box information. Since tweets from these scripts are all coming from Australia, we label the location tag as '1'.

3.3 Search API

3.3.1 Search functions

Unlike the stream API which is able to extract tweets endlessly, Twitter limited the request times of the search API to a 15-minute window. To retrieve as many tweets as possible, this API contains two functions:

1) The first one is to search the tweets within 7 days in 15 major cities in Australia, such as Melbourne, Sydney, by specifying the geocode to API requests. And we extracted the coordinates format location information for tweets from the return value “coordinates” or “bounding box” if exists, otherwise, the coordinates of geocode will be treated as its location information.
2) Search tweets from user timeline, which means fetching the previous 1000 posts by a particular user, specified by a given user id. A user list is built up and updated continuously by storing the user occurred in the fetched tweets in the first function. After fetching, we filter out the tweets without location information and save the rest to the database.

3.3.2 Parallelize

MPI is a protocol that enables the system to get higher capability, therefore, we use an MPI based python tool, mpich, to run 6 search API functions (three for each function) simultaneously to retrieve twitter data faster. Each function will get different tasks by splitting the geocode list and user list.

3.4 Data cleaning and storage

Filter useless data

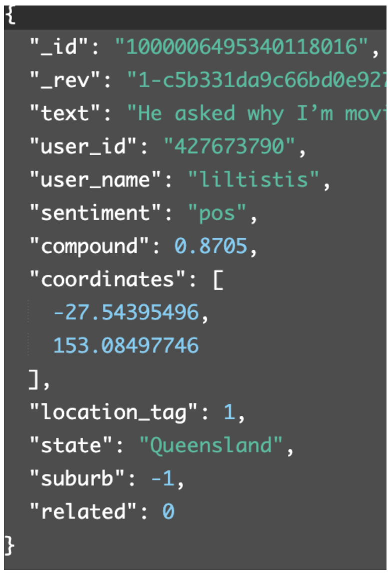
Since the tweet retrieved back from APIs contains many original information and it costs large storage to store irrelevant things, we perform the cleaning process to only keep information that is necessary for data analysis. Thus, unique id string for each tweet, original tweet text, user's unique id_string, coordinate information if not null are considered to be important to record. We also remove urls, "@" tag in the text for sentiment analysis, and ignore the tweet started with “RT” as they are not meaningful in scenario analysis.

Extract features

Some important features are extracted for data analysis and visualization. We use python package vaderSentiment Analyzer, which is a lexicon and rule-based tool, (cite) to evaluate the sentiment of each tweet. Vader analysis will yield results for each tweet as the format below:

{‘neg’:0.0,’neu’:0.196,’pos’:0.804,’compound’:0.6249}

We save the tag of sentiment (“positive", "negative" or "neutral") as well as a “compound” value (range from -1 to 1) for future sentiment analysis in the designed scenarios. In order to make the MapReduce task easier to implement, we included a location tag ("1" for Australian Tweets, "0" otherwise) to find the ratio of Australian tweets and Global tweets that are relevant to our topic. Furthermore, the information of which state and suburb this twitter location belongs to is determined based on geographic data, and the flag of whether this tweet is related to our topic is stored. All of these data are stored in a key-pair format, the example is shown in Fig 3-1 below.



```
{
  "_id": "1000006495340118016",
  "_rev": "1-c5b331da9c66bd0e92
  "text": "He asked why I’m mov
  "user_id": "427673790",
  "user_name": "liltistis",
  "sentiment": "pos",
  "compound": 0.8705,
  "coordinates": [
    -27.54395496,
    153.08497746
  ],
  "location_tag": 1,
  "state": "Queensland",
  "suburb": -1,
  "related": 0
}
```

Fig 3-1 the format of tweets stored in database

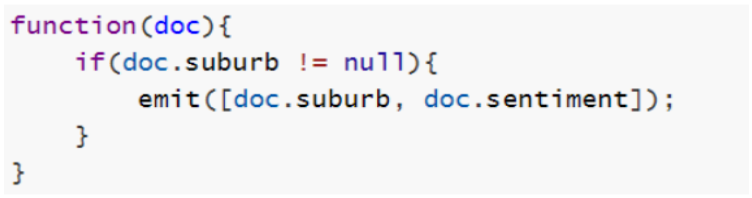
4 Tweets Data Processing and Analysis

4.1 Data analysis module

In order to analyze the harvested twitter data, we utilize the MapReduce functions offered by CouchDB. All documents in the database will be processed and filtered by the Map() function which will produce a list of key/value pairs, and the result will be sorted by key. The result can be then aggregated and summarized by Reduce()

function. This technique is efficient when processing a large amount of data, as the data can be processed in parallel using Map() function. Although it takes a relatively long time to build the B-tree for a view when the query is called the first time, after the first call, unless the Map() function changes, only the new and changed documents will be called.

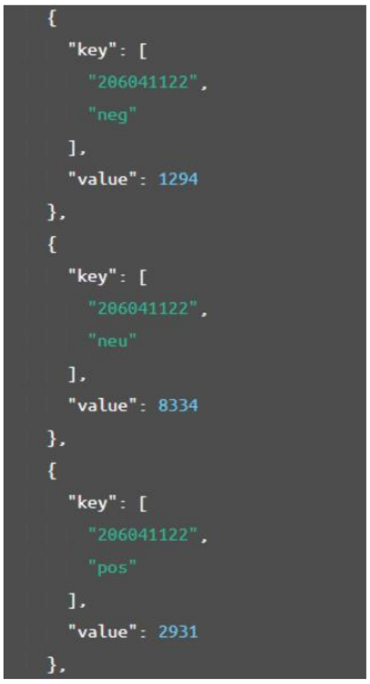
In this project, MapReduce is used to count the number of tweets each city and suburb have, and the average sentiment score of an area. An example of a map function to count the number of positive, negative, and neutral tweets of each suburb is shown in Fig 4-1:



```
function(doc){
    if(doc.suburb != null){
        emit([doc.suburb, doc.sentiment]);
    }
}
```

Fig 4-1 an example of map function

The function goes through each document to find if the suburb is within greater Melbourne. If it is, the function emits the key of [suburb, sentiment]. Using reduce function “_count” and group at level2, the results of the same key will be aggregated and counted. Fig 4-2 shows the aggregation result for the suburb with SA2 code 206041122.



```
{
  "key": [
    "206041122",
    "neg"
  ],
  "value": 1294
},
{
  "key": [
    "206041122",
    "neu"
  ],
  "value": 8334
},
{
  "key": [
    "206041122",
    "pos"
  ],
  "value": 2931
},
```

Fig 4-2 part of the query result

4.2 Front-end design

The web architecture contains two main components of our web visualization which are the basic visualization process and the web application design. In general, the web application in the instance named WebServer is developed by using Bootstrap along with Flask. In addition, libraries like Leaflet and ECharts are used to visualize our data. They are free and open-source web frameworks, which are suitable for a novice to develop rapidly and design pragmatically. Finally, the most important part is to query data from the server, and ReSTful API facilitates the communications between CouchDB and the web. It allows us to pass the data to different charts or maps, which ensures the data integrity of the entire web.

The web architecture is illustrated below:

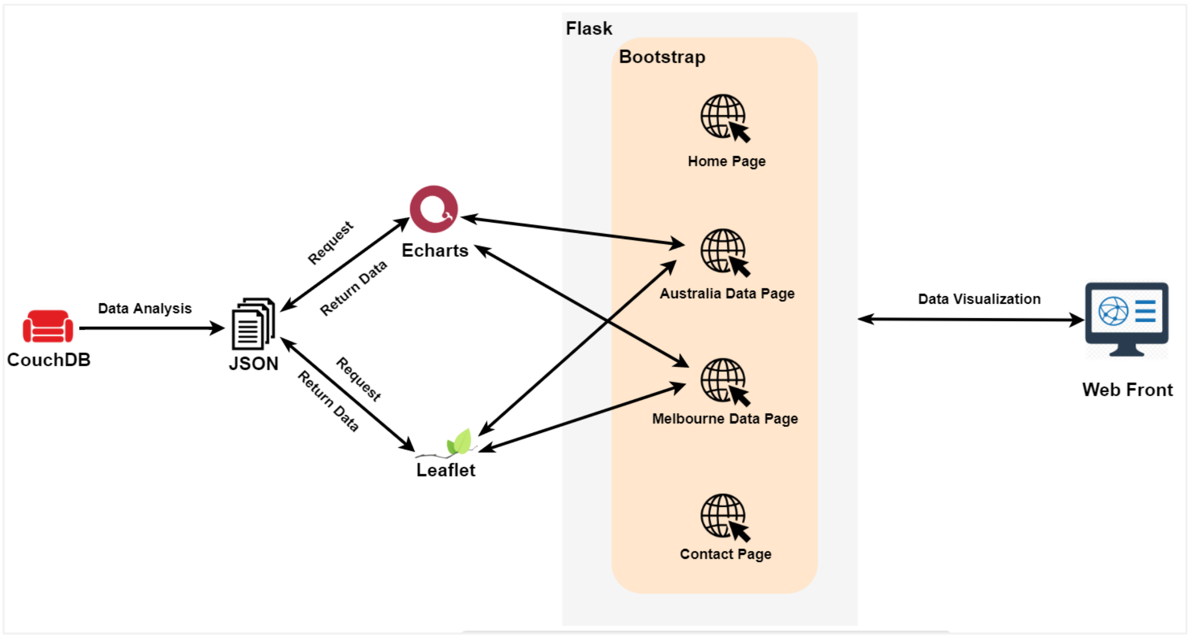


Fig 4-3 The web Architecture

4.2.1 Brief introduction of the technology stack

Bootstrap is a free and open-source CSS framework directed at responsive, mobile-first front-end web development. It contains CSS- and (optionally) JavaScript-based design templates for typography, forms, buttons, navigation, and other interface components.

Flask is a micro web framework written in Python. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself.

Leaflet is the leading open-source JavaScript library for mobile-friendly interactive maps. It has all the mapping features most developers ever need. Leaflet is designed with simplicity, performance and usability in mind.

ECharts, an open-sourced, web-based, cross-platform framework, supports the rapid construction of interactive visualization. The motivation is driven by three goals: easy-to-use, rich built-in interactions, and high performance. The kernel of Echarts is a suite of declarative visual design language that customizes built-in chart types.

4.2.2 Web application design

There are three components of our web: the home page, the data display pages and the contact page. More detailed information about each page is described below.

- Home Page

Home page shows three pictures related to overall Australia, suburb of Melbourne, and team members respectively, and there is a navigation button that allows users to enter the data pages, contact page and back to home page. The main purpose of the home page is to make users directly know each child component, which improves the user experience.

- Data Visualization Pages

There are two pages called Big Cities and Melbourne to show our data. In order to find out the correlations between aurin and twitter data in different regions, Leaflet and ECharts are used for data visualization.

For maps, it directly shows the number of data in different regions, the depth of the color indicates the different number of data which shows in the bottom-right corner of the map. When the mouse hovers over a region, the top-right corner of the map will show the information of data. Also, some emojis can indicate the different sentiment of data. For charts, different types of charts, such as pie charts and bar charts, are used to display the detailed information of data in some way. Users can interact with the chart, and the information will be changed if we hover over different parts of the chart. Filter and sort functionality is embedded inside charts.

- Contact Page

This page shows the information of team members.

4.2.3 Basic visualization process

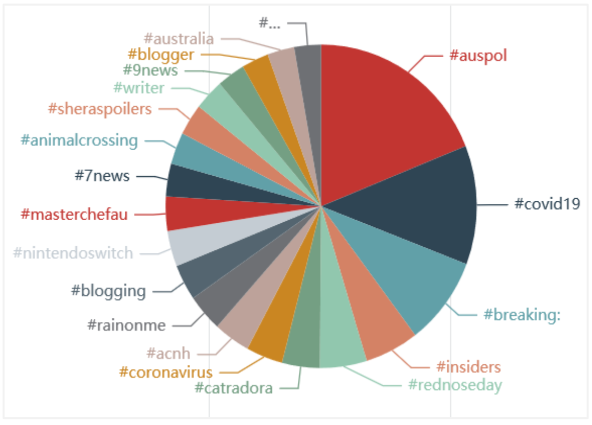
Visualization process is an important part to display the collected and analyzed data. In this process, our web server retrieves json format data from CouchDB by using RESTful API. Meanwhile, an asynchronous technique, ajax, is used while requesting the data to avoid long waiting time of accessing the website. Web front will always show the latest data because data is requested from CouchDB for browsing the web every time. Thus, users can view relevant and latest data in the front-end website.

4.3 Tweets scenarios and visualization

The data is analyzed according to three different aspects: entire Australia, the big cities of Australia, and the suburbs of Greater Melbourne.

4.3.1 Australia overview

Focused on Australia, the most popular 20 topics among all the tweets as well as alcohol-related tweets are obtained, and Fig 4-4, 4-5 present the results.



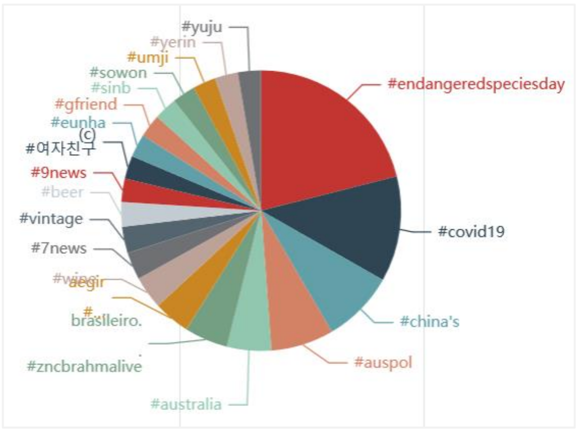


Fig 4-4 Top 20 hashtags of all tweets

Fig 4-5. Top 20 hashtags of alcohol-related tweets

The most common hashtags are '#auspol', short for Australian politics, and '#covid19'. And hashtags like '#breaking', '7news' all suggest Australia people like to discuss the latest news on twitter.

In terms of alcohol-related tweets, some types of liquor can be seen on the chart, such as wine and beer. An interesting finding is that people also talk about Gfriend, a South Korea girl group, and its group members a lot.

4.3.2 Big cities

Greater Capital City Statistical Areas (GCCSAs), which represent the functional extent of each state, is selected to conduct analysis along with Aurin data.

Population and number of tweets

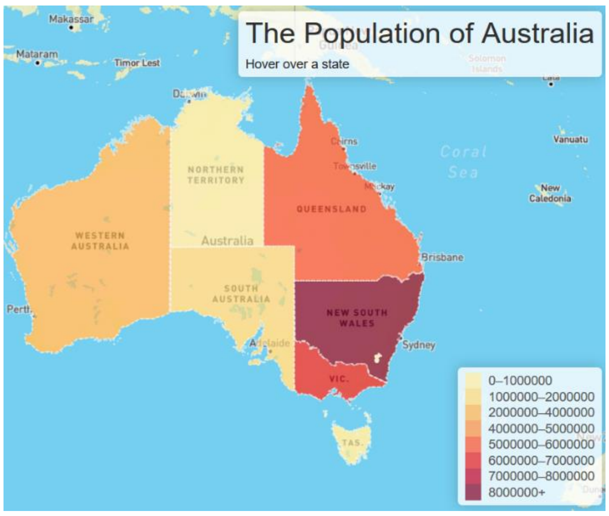


Fig 4-6 population of Australia

New South Wales (NSW) has the largest population, followed by Victoria (VIC) and Queensland (QLD). But both VIC and QLD have far more tweets than NSW, with QLD having the largest number of tweets, suggesting people in NSW less prefer to post tweets.

While the number of tweets of QLD is about 18% larger than that of VIC, the latter has the largest number of alcohol-related tweets (20,894 in VIC versus 19,565 in QLD), indicating people in VIC tend to tweet about alcohol more

Sentiment of tweets

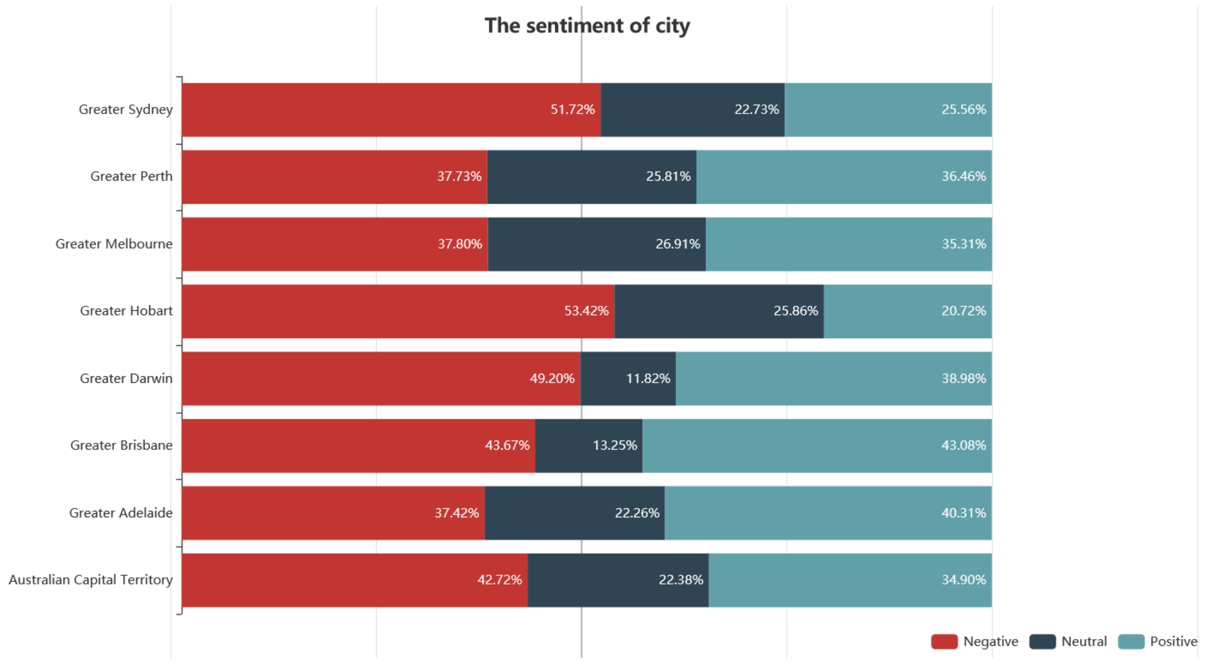


Fig 4-7 the alcohol-related tweet sentiment of cities

For the sentiment of each city, Greater Hobart and Greater Sydney have the highest negative tweets percentage in terms of alcohol-related tweets. There is no strong correlation between the sentiment and age, gender ratio of each city, and the result is not shown here.

4.3.3 Melbourne suburbs

Focused on a specific city - Greater Melbourne, analysis is carried out with respect to its suburbs. Statistical Area Level 2 is used to get the suburb level of Greater Melbourne, and among all the 309 suburbs, the number of tweets posted and the average sentiment for each suburb are summarised.

Income and sentiment

The median household weekly income of each suburb is used to explore the correlation with the average sentiment of the area.

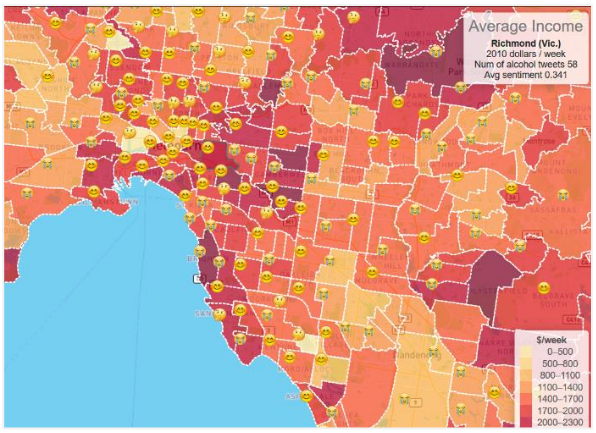


Fig 4-8 Richmond

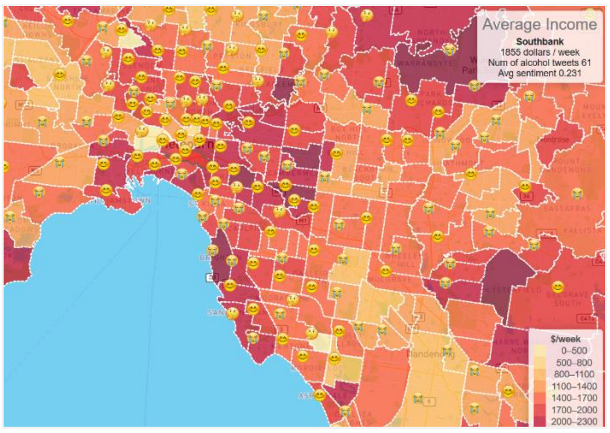


Fig 4-9 Southbank

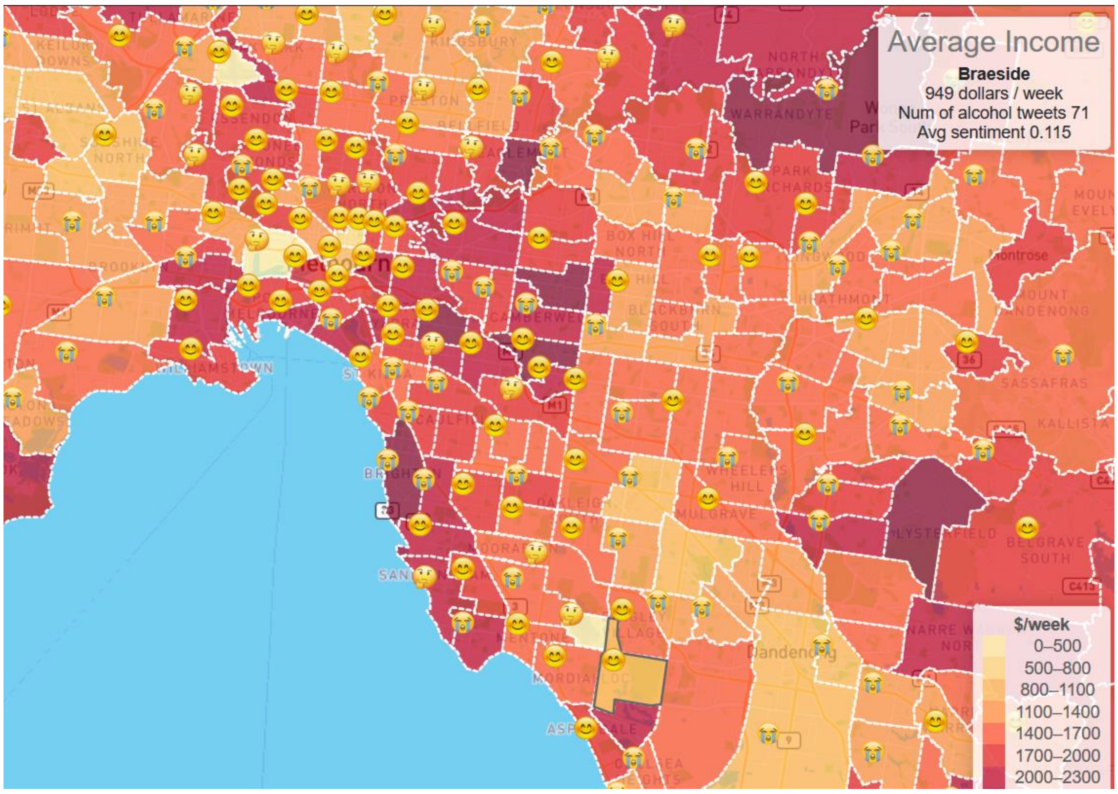


Fig 4-10 Braeside

The above three figures show the median income of Richmond, Southbank, and Braeside, as well as the number of tweets and the average sentiment of these areas. Having a similar number of tweets toward alcohol makes them comparable. These three suburbs show a positive correlation between the income and the average sentiment, possibly because when wealthier people mention alcohol, they talk more about bars and having a drink with friends, while people under economic pressure are likely to feel more upset. However, this is not the case with some other suburbs, and the reason might lie in income being not the only factor that influences a person's emotion.

The above three figures show the median income of Richmond, Southbank, and Braeside, as well as the number of tweets and the average sentiment of these areas. Having a similar number of tweets toward alcohol makes them comparable. These three suburbs show a positive correlation between the income and the average sentiment, possibly because when wealthier people mention alcohol, they talk more about bars and having a drink with friends, while people under economic pressure are likely to feel more upset. However, this is not the case with some other suburbs, and the reason might lie in income being not the only factor that influences a person's emotion.

Education and sentiment

Fig 4-12 shows that there is not much correlation between diploma rate and income. This is understandable as many other import factors determine a person's income besides his/her education level.

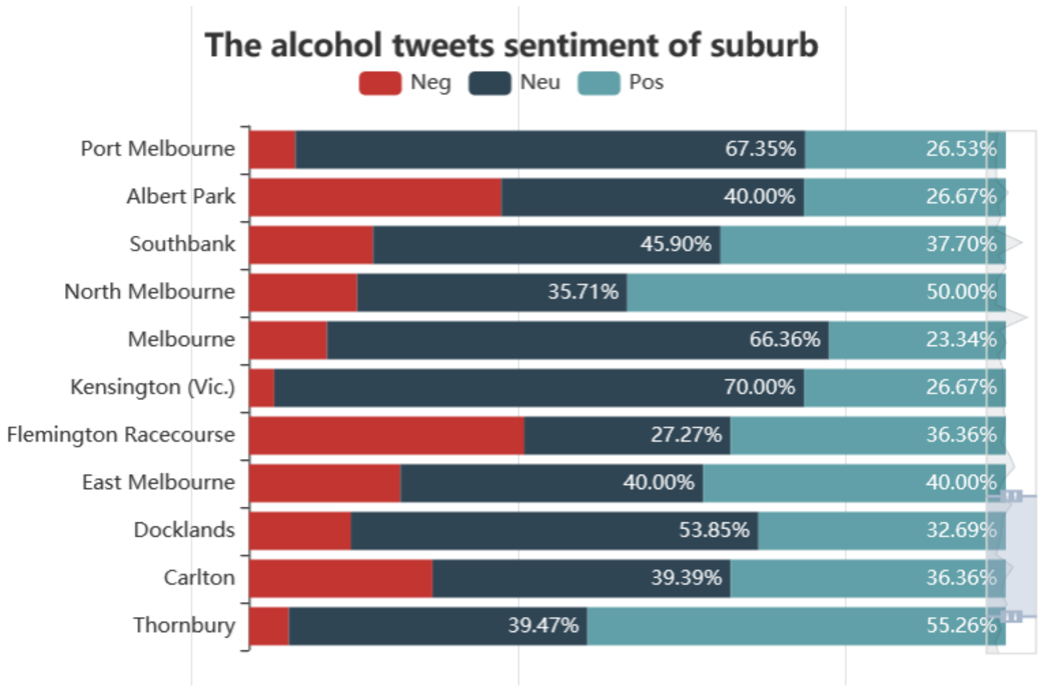


Fig 4-11 The alcohol-related tweets sentiment of suburbs

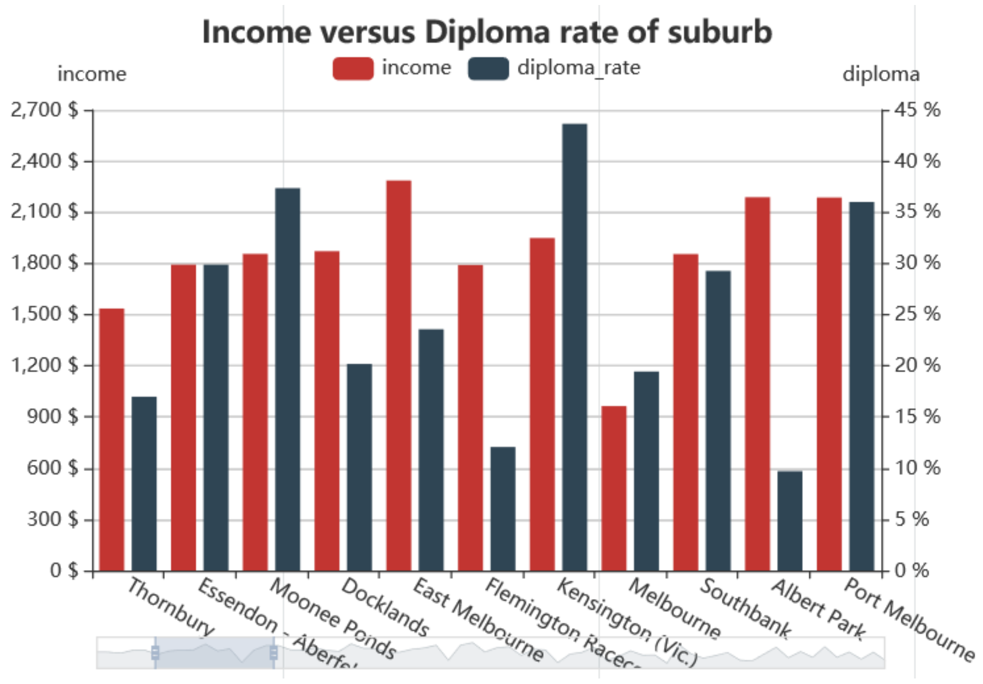


Fig 4-12 Income versus diploma rate of suburbs

When the above two figures are combined, It can be seen that Kensington has the highest diploma rate, and it also has the lowest percentage of negative alcohol-related tweets, while Flemington Racecourse and Albert Park has the lowest diploma rate, with both districts having the highest ratio of negative alcohol-related tweets. This seems to imply that when a person has a lower education level, he/she is more likely to have some negative feelings when mentioning alcohol.

5 Error Handling and Fault Tolerance

5.1 Error handling with CouchDB

We have encountered a number of challenges through the process of deploying the CouchDB clustering. Besides the scripts errors that can be fixed by studying and looking carefully into the script, there are two typical errors that could be handled:

CouchDB cluster with no "_replicator" "_user":

there were no "_replicator" "_user" databases in CouchDB after the cluster was deployed. The cluster seemed to work well with the correct membership and when adding/deleting/updating some databases on one of the instances, the other two showed the same result. It turned out later that without these two databases, there would be some issues when reloading the data. From the CouchDB log, we can only see that these two databases fail to launch. The solution that we used was to clean up the /var/lib/couchdb directory, to remove the old database metadata.

CouchDB cluster setup failure：

In the ansible automation deployment, we encountered abnormal cluster build-up. One reason is the time interval between the three CouchDB instance start up and cluster build is short. We enlarge it to 3 minutes; it works in most cases. But during some peak time, longer time is needed. Finally, we use 5 minutes. It seems to work all the time.

Data backup:

CouchDB server may be down or have other potential issues, thus, we use scripts to regularly back up tweets data and view data from databases to json files. And these large backup files will be compressed every night and downloaded to our local machine to make sure the database is recoverable.

5.2 Error handling with Twitter Harvester

Remove duplication

As the search API tends to return the nearest posts for each request, for example, the user's newest tweets or the newest tweets in a given city, there might be duplicated data between the results of every request. In order to avoid getting the same records as return, we set max-id parameter to the "_id" value of the last tweet to notice the API return tweets whose id is smaller than those we already have.

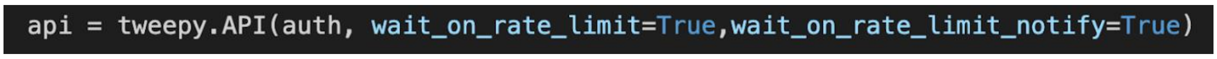
Besides, twitter id is used as "_id" in CouchDB and whether it is already in the database is checked, as the twitter id is unique, this method will filter the duplicate tweets for both search API and stream API. Another situation that may cause duplication is Re-tweet, so all of the text started with "RT" will not be stored into the database.

Tackle API rate limitation

According to the Twitter API document, there are lots of rate limitations on request API usage. For a basic developer account, the search API could only get the tweets posted within 7 days, and the function of searching by geocode can only return 100 tweets/per request, while the user timeline API can only fetch 200 records. On the other hand, each API application of twitter can only send 180 search requests and 900 user timeline requests every 15 minutes.

To get more request times in every 15-minute window, we used two accounts to create 3 developer authentication applications. As mentioned in the parallelize section, the harvester system has multiple processes running by MPI, thus, these authenticated key pairs will be distributed to these processes to improve the data collected speed. For search API, the fetch rate was extended to 54,000 tweets from location function and 540,000 from user timeline function per window from 18,000 tweets per window. However, since most of the twitters don’t have precise location information and the response time cost of user timeline API, we can hardly reach this upper bound.

Twitter will block API account which keeps sending HTTP requests after reaching the limit, in order to avoid this issue, two parameters "wait_on_rate_limit" and "wait_on_rate_limit_notify" of tweepy API object were set to True to conduct the harvesters automatically pause around 500-700 seconds while reaching the request limitation.



```
api = tweepy.API(auth, wait_on_rate_limit=True,wait_on_rate_limit_notify=True)
```

Checkpoint and restart

In order to prevent the possibility of the harvester program crashing, the search function will log the current max-id and user id of the search process to json file after every 10 requests sent, every time we restart the harvester, it will check whether the breakpoint exists first and load the search progress of last time. The checkpoint indicates which user’s posts should the user timeline function start to request in the user list. And we also save the current user list to the checkpoint.

We also received some errors like "http Incomplete Read error" and "urllib3_incompleteRead" which caused the harvester program to shut down accidentally. As the harvester was running in the background overnight silently, we could not notice this problem as soon as possible, in order to avoid the program stopping to fetch data, we imported respective error handling libraries, tracked these errors and forced the scripts to sleep for 10 sections and then continued working.

Limitation discussion of twitter harvester

In both stream and search API, we encountered the issue that not many tweets have coordination information. For the search API, since we set the radius of different cities, we do not have the precise coordinate information for each tweet. In other words, we label the same coordinates for tweets that are from the same city range, this will increase the number of tweets with coordinates information, but it will impact the topic analysis within Melbourne suburb. Another drawback is that twitter's official search API doesn’t support filter the return data by geolocation tag and topic words at the same time, and we don’t use any state-of-the-art Natural Language Processing technique to test if the text relates to one topic, so by exactly matching the keywords, there is only a small part of tweets among millions of data falls into the specific alcohol topic.

Since we changed the data format according to visualization needed during the project, some of the tweets data retrieved in early time have different types of characters to the latest data. To issue this problem, we created some scripts to merge the old data and new data to a new database with unified format. A CouchDB management API, Cloudant API, is used to implement the script to transfer and store the data fast.

5.3 Error handling with Webserver

“404” error

In web design, it is necessary to have a “404” error HTML. In practice, web servers may work unsuccessfully. If a visitor tries to find a non-existent page, the site’s server will throw up a “404” error and tell the visitor that the page can’t be found.

6 Conclusion

In summary, in this project a scalable cloud system is deployed utilising Melbourne Research Cloud, CouchDB cluster, MapReduce, Flask, Ansible, Twitter API. Over 6 million tweets have been collected to help find out some interesting relationship between alcohol related tweets and demographic characteristics in Australian cities.

The results show that population distribution is not closely related to the number of tweets posted, as NSW has the largest population, but QLD and VIC are the ones with the largest number of tweets of all topics and alcohol-related respectively. Suburb level analysis suggests that for some suburbs in greater Melbourne, there is a positive correlation between income and sentiment. Similarly, a negative correlation between the diploma rate and the percentage of negative tweets can be found in some other suburbs.

The system deployed has handled with errors from many aspects, and it continuously harvest tweets to combine with analysis with Aurin data, with results visualized and updated in the front-end. However, the design and functionalities are still a comparative simple. There can be many improvements in the future, like using more advanced algorithms in NLP and data analysis to deal with data collection and analysis. Besides, more security and backup plan could be deployed, for example, backup MapReduce files on web server to ensure the analysis outcome is available even when the CouchDB cluster is not available.

Reference

1. Saleem Alhabash et al, 140 Characters of Intoxication: Exploring the Prevalence of Alcohol-Related Tweets and Predicting Their Virality, https://doi.org/10.1177/2158244018803137
2. Paulhus, D.L., Vazire, S., 2007. The self-report method. Handb. Res. Methods Personal. Psychol. 1, 224–239.
3. Social media statistics-world + Australia, Demographics 2019, https://www.exalteddigital.com/social-media-statistics-world-australia/
4. Litt DM, Lewis MA, Spiro ES, Aulck L, Waldron KA, Head-Corliss MK, Swanson A, #drunktwitter: Examining the relations between alcohol-related Twitter content and alcohol willingness and use among underage young adults, Drug and Alcohol Dependence (2018), https://doi.org/10.1016/j.drugalcdep.2018.08.021
5. Melbourne Research Cloud documentation: https://docs.cloud.unimelb.edu.au/

Appendix Team member and Roles

- Cooperative parts: architecture and system design, topic & scenarios, general problem solving, report
- Personal responsibilities:

Team Member	**Responsibility**
Ruiqi ZHU	MapReduce; Data process and analysis
Zhengyang LI	Ansible to deploy web server/harvester/data analysis module; Web server frontend and backend; Harvester-search API; Videos
Jianxin XU	Web-based Design and Visualization
Qiuxia YIN	Instances &CouchDB cluster; one-key deployment; Aurin data
Fang Qu	Harvester-stream API; Database backup and merge