COMP90024 Project 2 Report

Melbourne: The Most Liveable City...?

May, 2022

Team 31, Melbourne

Bingzhe Jin (1080774), bingzhej@student.unimelb.edu.au

Hongwei Chen (891632), hongweic1@student.unimelb.edu.au

Tian Hui (1054513), tian.hui@student.unimelb.edu.au

Zhen Cai (1049487), zhcai@student.unimelb.edu.au

Ziqi Zhang (1241157), zhazz4@student.unimelb.edu.au

Abstract

Cloud computing technologies make it simple for data analysis in a scalable manner. In this project, we examine the livability of Melbourne through the lens of Yarra Trams. A sentiment analysis of recent and past tweets (2014-2015) associated with trams in Melbourne, juxtaposed with AURIN statistics, may help shed some light on this topic. We use Ansible to automatically configure resources and deploy services on the Melbourne Research Cloud. Data are stored on virtual machines in CouchDB Docker containers, and several nodes across form a cluster that offers replication for fault tolerance. Following that, a map visualization is available for the public to view our latest, real-time analytical results. This report summarizes all of our efforts for Assignment 2.

Keywords Melbourne Research Cloud · Twitter · AURIN · Yarra Trams · Sentiment Analysis · Map Visualisation · Web Application · Ansible · CouchDB · Docker

GitHub repository: https://github.com/CccT31/comp90024
Demonstration on YouTube: https://youtu.be/zCwB1Lq7tCs

Contents

1 User Guide

Clone our git repository and change directory to *./ansible*.

1.1 Prerequisites

Download the OpenStack RC file[1] from Melbourne Research Cloud (MRC) as shown in Figure 1. Rename it to *openrc.sh*, put it under the current directory.

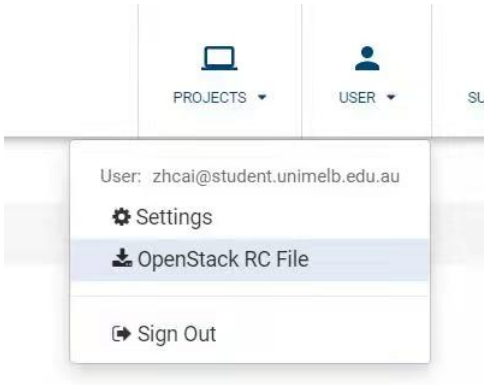

Figure 1: Download the OpenStack RC file from MRC.

Figure 2 generates a password for OpenStack API access in *Settings >Reset Password*.

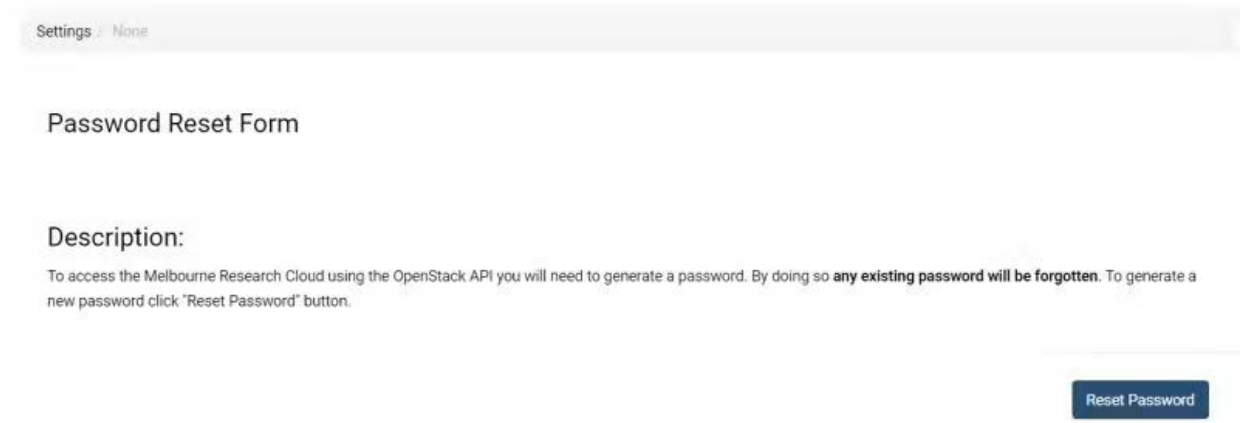

Figure 2: Generate an API password.

[1] A tiny bash script that contains environment variables that are necessary to run a command-line client.

1.2 Create and configure OpenStack instances

The system has scripted deployment capabilities using *Ansible* automation.

First of all, an example installation on Linux (Ubuntu) follows:

```
> sudo apt-get update && sudo apt-get install -y software-properties-common
> sudo apt-add-repository --yes --update ppa:ansible/ansible
> sudo apt-get install -y ansible
```

For other OS distributions, you also need to manually install dependencies *python3-dev*, *python3-setuptools*, and *python3-pip* after this.

The second step is to generate a (RSA) key pair, one should be named *cloud.key* and the other be *cloud.key.pub*. The public key will be uploaded to MRC by Ansible at the time of instance creation, and the private key will be used to login via SSH to those VMs after they are launched.

```
> ssh-keygen -t rsa -f cloud.key && chmod 600 cloud.key
```

Next is to 'tunnel'[2] into the campus network if from outside of the University of Melbourne network.

Once done, run scripts listed below. It will create instances and complete the related environment configurations. VMs clone/pull the latest code from our GitHub repository's main branch. Instances IP addresses are displayed in terminal output and kept at *./ansible/vars/setup.yaml*. Then this file is upload to all VMs, replacing the old one fetched from GitHub. The webapp will access db instance IP from here afterwards, so avoids hardcoding it.

```
> chmod +x run-mrc.sh && ./run-mrc.sh
```

1.3 Sync CouchDB Views

SSH onto any of instance db, harvester1, or harvester2.

```
> ssh -i cloud.key ubuntu@<instance-ip>
```

```
ubuntu@db:..$ cd ~/comp90024/harvester
ubuntu@db:..$ chmod +x view.sh && ./view.sh
```

[2] https://studentit.unimelb.edu.au/wireless-vpn/vpn

1.4 Upload Augmenting Tweets (Optional)

We are given 2.5 million historic tweets from 2014-2015 as augmenting data. To use them, link the data file and secure copy them to */home/ubuntu/* directory on any of 3 instances in Section 1.3.

```
> ln -s <your-data-path>/twitter-melb.json
> scp -i cloud.key twitter-melb.json ubuntu@<instance-ip>:/home/ubuntu/
```

After this finishes, run below on that instance and data will be added into *historic_melb* database.

```
ubuntu@db:..$ cd ~/comp90024/harvester
ubuntu@db:..$ chmod +x upload_hist.sh && ./upload_hist.sh
```

1.5 Start Harvesters

You can do Twitter search/streaming with some different queries on harvester1 and harvester2. Scripts are similar and harvesters can scale vertically by executing multiple .sh. Thus we just show starting one as a quick example. More on harvester tasks will be discussed in Section 4.1.

```
ubuntu@harvester1:..$ cd ~/comp90024/harvester
ubuntu@harvester1:..$ chmod +x search_by_*.sh && ./search_by_destination.sh
```

1.6 Start Web Application

```
ubuntu@webapp:..$ cd ~/comp90024/webapp
ubuntu@webapp:..$ npm install
ubuntu@webapp:..$ nohup npm start &
```

Visit the url:

```
<web-instance-ip>:3000
```

with your browser for your app.

* *nohup* is a POSIX command. Its purpose is to execute a command in the background without hanging up the command-line input, and keep it running when the user logs out.

2 System Architecture Design

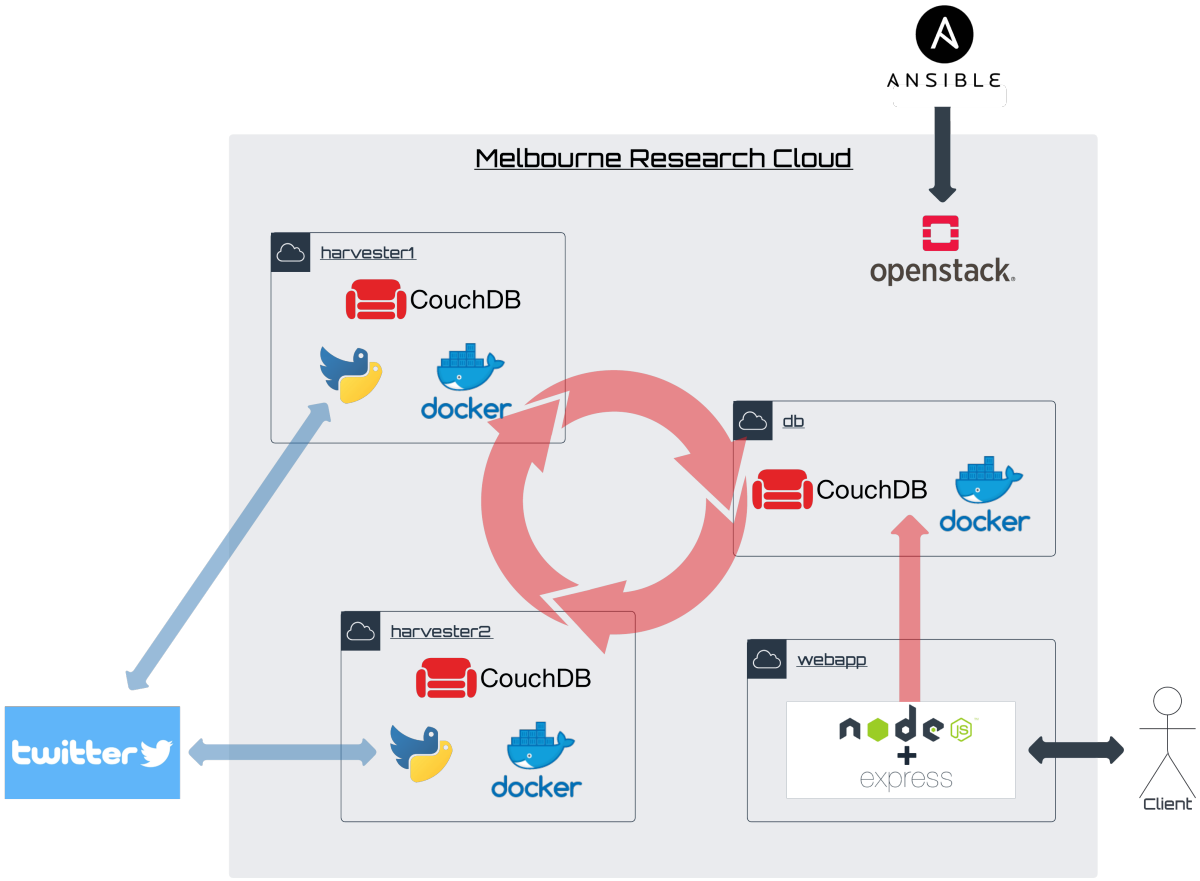

Figure 3: System architecture diagram. There are 4 instances deployed: db, harvester1, harvester2, and webapp.

2.1 Melbourne Research Cloud

The Melbourne Research Cloud (MRC) offers free on-demand compute power to researchers at the University of Melbourne and its affiliated institutions. It's a private cloud with similar functionality to those by commercial cloud providers such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform. [1] We don't pay anything for MRC, and staffs do all the jobs of hardware administration for us. It provides Infrastructure-as-a-Service (IaaS), which takes us to expand our DevOps skill stack. MRC is based on a family of open-source softwares collectively referred to as OpenStack, and hence exploits its portability and security during provisioning.

2.2 Resource Allocation

MRC specifies the limit of our project's resource utilisation, allows a maximum of up to 4 instances, 8 VCPUs, and 500GB of volume space (we use half). The allocation breakdown is as follows:

Instance name	Number of VCPUs	Memory size (GB)	Volume size (GB)
db	2	9	70
harvester1	2	9	70
harvester2	2	9	70
webapp	2	9	40

Table 1: Allocation of resources on MRC.

2.3 Key Components

2.3.1 CouchDB

Relational database management systems (DBMSs) are excellent for consistency and availability. The normalization model suggests fine-grained data, which should be rare in a world of big data. Our data is harvested from the social media platform Twitter, and the majority of it is semi-structured or unstructured, at which point veracity and variety cannot be taken for granted. These are the reasons why we choose CouchDB, a document-oriented database with Multi-Version Concurrency Control (MVCC). This algorithm ensures availability and partition tolerance. Concurrent updates are resolved by reconciling revision numbers.

We decide ahead of time to construct a cluster of three nodes rather than a standalone. Data are replicated, i.e. storing the same document on many nodes to make the database fault-tolerant. The three nodes are on three network-connected instances: db, harvester1, and harvester2. db runs the special setup coordination node, which is also the only one that answers queries. If a document isn't found here, db will request it from the two remote harvesters' shards (if available) and returns it to the client via webapp. Furthermore, db builds MapReduce views for data.

CouchDB uses the following ports [2]:

- 5984: Standard clustered port for all HTTP API requests.
- 4369: Erlang port mapper daemon (epmd).
- Random above 1024: Intra-cluster communication. We use 9100-9102 for 3 CouchDB nodes.

2.3.2 Docker

Containerization allows virtual instances to run on top of a single host OS that each container only holds the application and related binaries, sharing hardware resources. In practice, containerization helps keep the consistency between the production and deployment environment, and makes the setup process more manageable, especially when working with multiple OS distributions. For this project, we use Docker as a containerization solution.

Once volumes are mounted, Docker will be installed on the VMs. On db, harvester1, and harvester2, *docker-compose* starts containers from the *couchdb:3.2.2* image. Because data within a Docker container isn't persisted by default when the container is no longer running (e.g. a crash), and a container itself has limited storage volume; we make it bind mounts at */data/couch/data*.

2.3.3 Harvester

The harvester1 and harvester2 collect tweets from Twitter. We use Python library *tweepy* for accessing Twitter API v2 [3], from which we have the options to search tweets from the last 7 days, alternatively gather real-time in terms of streaming. In order to obtain more data, both ways are employed. Tweets are pre-processed before being saved into local *melb_db* database.

2.3.4 Web application

The Express.js framework is deployed on the webapp to assist with visualisation. webapp retrieves data from the database and renders web pages. The system functionalities will demonstrated in Section 5.

3 Deployment Automation Details

3.1 Scalability

Ansible is an open-source scripted configuration management tool that can be re-used to automate tasks, deploy applications and infrastructure. It only needs to connect to the control node through SSH (by default) to perform tasks. We used Ansible to create cloud instances, set up security baselines, configure environments, and install packages; make it simple to scale horizontally.

See Section 1.5 for solution of vertical scalability.

A bit of nomenclature:

- Control node: Any machine with Ansible installed and tasks invoked.
- Managed Node: The network devices you manage with Ansible.
- Inventory: A list of managed nodes.
- Role: Used to organize playbooks in a hierarchical and structured way. Can automatically load certain variable files, tasks and handlers accordingly.
- Playbook: An ordered lists of tasks, where a single task is the smallest unit of action in Ansible. Also contains variables.

3.2 Instances Creation

How to create and configure instances on MRC can be found in Section 1.2. The shell script *run-mrc.sh* runs *main.yaml* playbook with *cloud.key* selected as the SSH private key to use.

In instance creation step, 5 roles need to be run:

- openstack/common: Install *pip* and *openstacksdk* on control node.
- openstack/key: Upload the public key to MRC.
- openstack/security-group: Create security groups rules.
- openstack/volume: Create volumes.
- openstack/instance: Create instances but instead of one by one, submit tasks in parallel. After all async tasks are completed, add host into the in-memory inventory and write to */var/setup.yaml*.

The availability zone melbourne-qh2-uom is the same for all instances and volumes. NeCTAR Ubuntu 20.04 LTS (Focal) amd64 is the OS image. The network used is qh2-uom-internal, where only access from under UOM LAN can be made.

3.3 Security

We have discussed SSH in previous sections. Moreover, for all instances, we have another group *intra_cluster* that restricts remote access to ports 5984, 4369, 9100-9102, i.e. CouchDB ports, but only allows among themselves. Therefore, an outsider cannot simply tamper with our database unless the person can ssh onto one of these instances.

A third security group called *client* specifies port 80, 443 and 3000, where 80 and 443 are used as the HTTP/s default network port and express.js frontend takes 3000. It's open to the public by permitting access from any IP address. webapp is in this group exclusively.

3.4 VM Configuration

After creation, these 3 roles are run for all instances:

- setup/mount-volume: Mount device */dev/vdb* to the */data* directory. Configure the */etc/fstab* file so that the system mounts volume automatically on startup in case of a reboot.
- setup/docker: Install *docker*, *docker-compose* and their dependencies.
- setup/git: Clone or pull our GitHub repository, copy instance IP addresses from control node to all managed nodes.

The next role *setup/couch* is run except for webapp in order to set up a CouchDB cluster. The *docker-compose.yaml.j2* file uses Jinja2 templating to enable dynamic expressions and access to variables and facts before the task is sent to target machines. For ease of testing, we set both COUCHDB_USER and COUCHDB_PASSWORD to *admin*. This is actually unnecessary; we may define variables for them, and in a real-world scenario, such a pair must be replaced.

```
1  version: "3"
2  services:
3    couchDB:
4      image: couchdb:3.2.2
5      restart: always
6      ports:
7        - "5984:5984"
8        - "4369:4369"
9        - "9100-9102:9100-9102"
10     volumes:
11       - /data/couch/data:/opt/couchdb/data
12     environment:
13       - COUCHDB_USER=admin
14       - COUCHDB_PASSWORD=admin
15       - COUCHDB_SECRET=aiB8ooj9cheikie
16       - NODENAME="{{ node }}"
17     command: "-setcookie zaafaeKe7gaNal6"
18     container_name: couch
```

We wish to connect three CouchDB nodes together after container has been launched. First set up the cluster at each node to join. Then choose db as the setup coordination node, give it the IP addresses of other nodes. We need it to add nodes, and finish cluster after all have already completed the first step which means they've prepared.

Cross-Origin Resource Sharing (CORS) supports web pages to run JavaScript inside a browser to make XMLHttpRequests to other domain without compromising either's security. Since CouchDB disables CORS by default, a change in *local.ini* is required. [4] There's an ansible task for this:

```
- name: CouchDB CORS
  become: yes
  shell: docker exec -it couch sed -ibk -e 's/\[httpd\]/\[httpd\]\nenable_cors =
      true\n\[cors\]\norigins = *\ncredentials = true\nmethods = GET, PUT, POST,
      HEAD, DELETE\nheaders = accept, authorization, content-type, origin, referer,
       x-csrf-token\n/g' /opt/couchdb/etc/local.ini && docker restart couch
  when: node == "{{ db }}"
```

As a final step of ansible, pip requirements and apt dependencies are pre-installed by *setup/harvester* and *setup/webapp* for two harvesters and webapp, respectively.

4 Data Delivery

4.1 Collecting Tweets

Our goal here is to collect tweets associated with Yarra Trams, and a natural format of the query rule should look like a conjunction of these 4 filters:

```
"<tram-keywords> <geo-filter> lang:en -is:retweet"
```

Matching tweets (that have been classified by Twitter) as being English, and removing retweets[3] are not difficult. The real problems are deciding which keywords to use to target content of interest, and how to determine location correctly that we don't go outside Melbourne.

4.1.1 Trams

Keywords for this part are "myki", "tram", "trams", and "ptv" which stands for Public Transportation Victoria. We take a disjunction (OR) of them: any of which occurs in the text is enough to indicate its relevance. There's no concern about capitalization and non-whole-word matching because Twitter API evaluates queries in a case-insensitive, tokenized manner. [3]

4.1.2 Melbourne

The prevalence of geo-tagging is only ˜1-2% of all tweets. At the tweet level, two types of location information are available - an exact "point" location; or a Twitter Place with a "bounding box"

[3]Retweets are removed because some overly retweeted tweets overwhelms other data.

that represents an area of variable size. However, all of *has:geo*, *bounding_box* and *place* operators are advanced features that are only available for projects with academic research access. Thus our approach becomes inferring the location of each tweet from its text, by examining a range of keywords that people are most likely to mention when they tweet on a tram.

The Melbourne tram network on the PTV website [5] helped supply the idea for these keywords. One group of them can be tram route destinations such as "Docklands" and "Flinders", and another can be landmarks such as "Federation Square" and "Melbourne Cricket Ground". Not all of these words are used; in particular, we avoid names that can also be found elsewhere. "Kew", for example, is a Melbourne suburb as well as a small town in New South Wales. And, names of general places such as "Airport West" are also excluded. Keep in mind a single Twitter query should not exceed the length limit of up to 512 characters, including whitespaces.

When a tweet is retrieved, the database stores selected fields such id, text, and created_at. Several sentiment scores are calculated and stored as additional fields as well, to study a tweeter's sentiment. Section 4.2 will go over the specifics of how scores are calculated. An example CouchDB document:


```
{
 "id": "1525679406819074049",
 "key": "1525679406819074049",
 "value": {
  "rev": "1-0c16c9ec5580254352c7bf80692b45a9"
 },
 "doc": {
  "_id": "1525679406819074049",
  "_rev": "1-0c16c9ec5580254352c7bf80692b45a9",
  "text": "(gets on a tram) haha i'm from melbourne now",
  "author": 3304618931,
  "created_at": "2022-05-15T03:28:10+00:00",
  "geo": {},
  "polarity_score": 0.426,
  "introspection_score": 0.382,
  "temper_score": 0.06,
  "attitude_score": -0.025,
  "sensitivity_score": 0.055
 }
}
```


In our setup, we harvest about landmarks on harvester1 and destinations on harvester2, hence the chance of their "colliding", in other words, collecting duplicate tweets, is expected to be less. But we are still not claiming that a collision will bring the system down, where our measure uses the id of a tweet as the document's _id in CouchDB. This way, a tweet cannot be stored twice.

4.2 Sentiment Analysis

Sentiment analysis is required to gauge the emotions and attitudes conveyed by the tweets about trams. SenticNet [6] provides a range of APIs for this purpose. In our project, we followed the API's use of the Hourglass of Emotions [7] as a model of emotions. In this model, emotions are categorized into four aspects, which are introspection, temper, attitude, and sensitivity. Using the provided intensity ranking functionalities, a score of -1 and 1 can be obtained for each of the four aspects of the emotions when an input in the text format is given. The magnitude of the score represents the intensity of the emotion, whereas the sign reflects the direction of the emotion. For instance, "+1" represents the most positive emotion while "-1" means the most negative emotion. Besides the scores of these four aspects of emotions, an overall sentiment score, the polarity score, was also included.

For each raw tweet text, we first clean it to remove unnecessary and uninformative symbols like urls, emails, or other non-textual symbols. Then, we split the text into sentences. Iterate over each sentence, we use functions provided by APIs to calculate the five scores of each valid word in that sentence and keep track of the number of valid words in that sentence. After that, we aggregate the scores from each sentence and average them with the total number of valid words in the text.

Thereby, we obtain five scores for each tweet, and we store them alongside other attributes of the tweet into the database for further analysis.

4.3 Identification of Statistical Areas

Recall that we have two databases: *historic_melb* and *melb_db*. The location information for *melb_db* documents are not provided, while *historic_melb* did involve a nonempty *geo* field that gives us precise latitude and longitude of each historic tweet, and we classify its statistical areas (SA) 2, 3, 4, just before saving it into the database.[4]

The Australian Bureau of Statistics publishes ASGS GDA2020 digital boundary files. [8] A number of unnecessary columns, and rows of other cities like Sydney are included in the downloaded shapefiles. Thus we filter "GCC_CODE21 == 2GMEL", and only keep two columns - "SA4_NAME21" and "geometry", where the prior are the names of SA and each of the posterior is a polygon or multi-polygon indicating SA shape.

[4]Code in *./harvester/upload_hist.py*.

4.4 CouchDB Views

CouchDB views are MapReduce job definitions that are written to disk as B-tree indexes and become part of the database. As new data arrives, views are updated, and view results will help speed up lookups. This step does not require all mapped array keys and values to be reduced again; instead, such scenario is called a "re-reduce". To support re-reduces, an idempotent reduce function must be carefully written, which means that invoking reduce once versus one hundred times should not produce any difference. Also, *group_level* reduce query parameter works with array keys to support by-level aggregation. In general, if we have the knowledge of potential future queries, we can design MapReduce functions from the very start.

In total, there are 30 views: 15 in *historic_melb*, 15 in *melb_db*. For each database, there are 5 designs corresponding to 5 sentiment scores, each having 3 views - *sum_count*, *max_id*, and *min_id* for *melb_db*; but prefixed with *sa_* for *historic_melb*, e.g., *sa_sum_count*.

- sum_count: returns sum of a specific score in all documents and document count.
- max_id: returns the maximum of a score and that document id.
- min_id: returns the minimum of a score and that document id.
- sa_...: similar, but can aggregate by key levels.[5] Here's an example of how to fetch SA4 polarity scores:

```
ubuntu@webapp :..$ curl http://admin:admin@<db-instance-ip>:5984/historic_melb
    /_design/polarity/_view/sum_count?group_level=1
{"rows":[
{"key":["Melbourne - Inner"],"value":{"sum":75.195725,"count":573}},
{"key":["Melbourne - Inner East"],"value":{"sum":1.9080714285714,"count":52}},
{"key":["Melbourne - Inner South"],"value":{"sum":3.259216666667,"count":25}},
{"key":["Melbourne - North East"],"value":{"sum":0.3021833333333,"count":22}},
{"key":["Melbourne - North West"],"value":{"sum":1.5010666666666,"count":7}},
{"key":["Melbourne - Outer East"],"value":{"sum":3.1001666666666,"count":9}},
{"key":["Melbourne - South East"],"value":{"sum":0.8501666666666,"count":6}},
{"key":["Melbourne - West"],"value":{"sum":-0.03966666666666668,"count":13}}
]}
```

All JavaScript map/reduce functions definition can be found in *./harvester/view.py*.

[5] Key array is [doc.sa4, doc.sa3, doc.sa2, doc._id].

5 System Functionalities and Analysis

In this section, we demonstrate web app's visualisation, and attempt to connect it in some way to AURIN [9] data, identify some insights.

5.1 Sentiment Score Geo-wise Comparison

5.1.1 Heat Map Visualization of Sentiment Score

As indicated in the graph, the sentiment scores are represented by a heat map on Google Map, with the scores aggregated by statistical area. We can select different statistical areas, years[6], and sentiment analysis types for this heat map (e.g., polarity). The colour green is associated with a higher score, whereas red is associated with a lower score. When the mouse is over the areas, tooltips will appear with the area name and score.

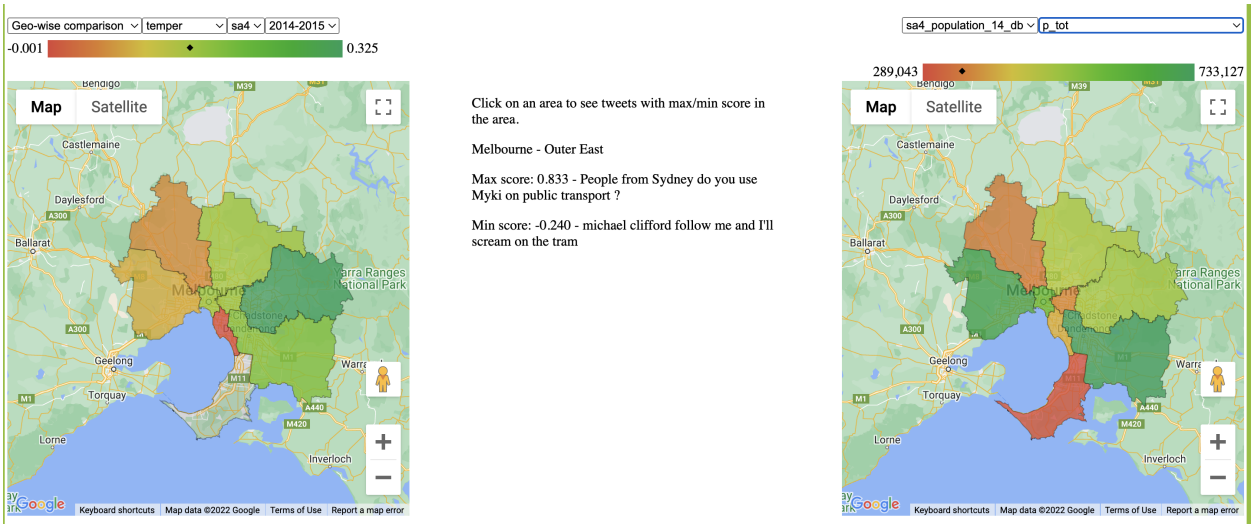

Figure 4: The sentiment score geo-wise comparison. Temper vs Total Population. Apart from the tram missing data in Mornington Peninsula (there is no tram), a positive correlation between population and temper is observed. One hypothesis could be that an increased population makes the government put more efforts in improving tram services. Therefore, people are less likely to be annoyed due to trams. Another finding is that on this graph, positive score has a larger magnitude, as well as the median seems to be positive.

[6]Due to the fact we only have geo data for 2014-15 tweets, 2014-15 is currently the only option for the year selection

5.1.2 List Tweet with Highest and Lowest Score

After you click on one of the areas, you will see tweets with the highest and lowest sentiment scores on the side. This can assist in comprehending the extreme viewpoints in any sector.

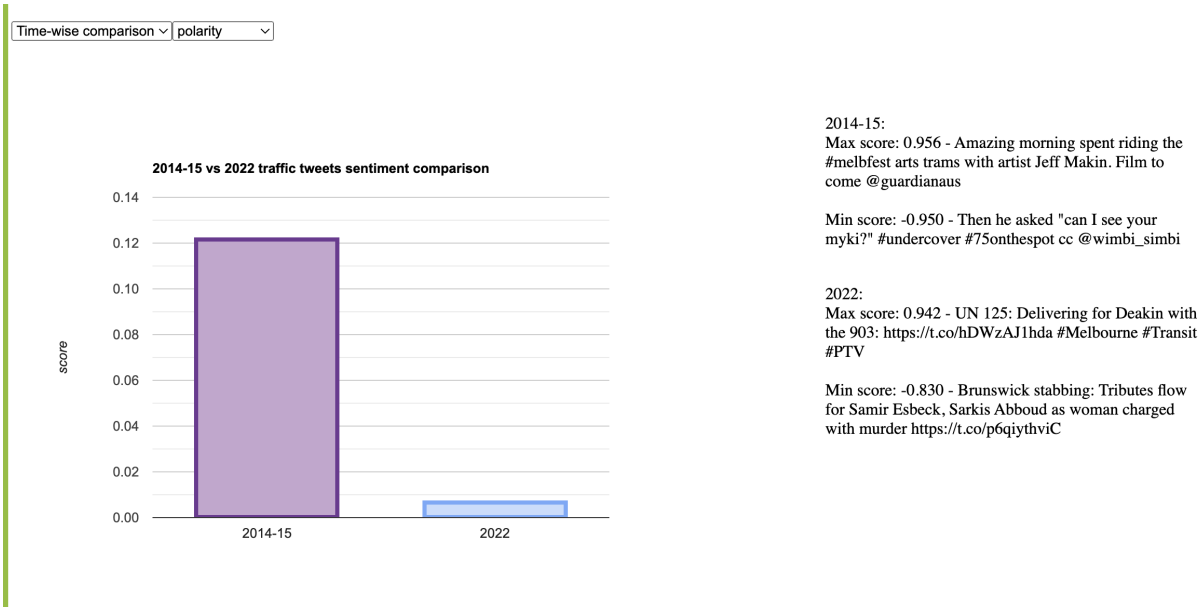

Figure 5: The AURIN data sentiment score comparison by geography. In 2022, the polarity score is lower. The data were taken in recent weeks, and frequent rallies and strikes these days may cause tram service disruptions.

5.1.3 AURIN Data by Statistical Area

Corresponding to the map for tweet sentiment score on the left, heat map for AURIN data is on the right. AURIN data is likewise kept in couchDB, with each data set on AURIN having its own database. The user can choose from a variety of data sets and features to visualise on a map. This is compared to the tweet scores on the left to look for data correlations.

5.2 Sentiment Score Time-wise Comparison

5.2.1 Bar Graph Compare Past and Current Sentiment Scores

In addition to the heat map, the history (2014-15) and current (2022) sentiment scores in the selected facet are displayed in a bar graph for comparison between historical twitter data and current tweet data.

5.2.2 List Tweet with Highest and Lowest Score

Tweets with the highest and lowest sentiment scores in the past and present will be displayed for each facet of sentiment score.

6 Limitations

There are certain challenges and limitations we faced during this project. By referring to "challenges", we mean they are solvable and have been discussed at some previous length. Next, we will present a list of main limitations encountered during fetching, processing, and using Twitter data.

Firstly, we do not have access to some advanced Twitter interfaces since those interfaces require academic track level access. One consequence of that is we do not have the option to search tweets by their geographic information. This means that we may be unable to perform analysis by regions in Melbourne for the recent data, although we can still do that for the provided historic data. Another implication is that we need to take alternative measures to locate tweets in Melbourne, such as keyword search.

There are also some other constraints when gathering tweets. The function to search recent tweets gives the tweets from the last seven days, but searching older tweets is not available since it requires a higher level of access. There is also a rate limit when fetching tweets through search or filtered stream, which may limit the number of tweets we can obtain in the end. Streaming rules are used to filter the stream. However, each rule is associated with the account it is using. This implies that we need different accounts to set up different rules if we want to filter with different keywords.

Thirdly, several difficulties and limitations were also encountered during data analysis. We noticed some features of these harvested tweets that may negatively impact our analysis. First of all, some tweets are merely Yarra Trams's notices about service changes or disruptions. These tweets may hardly reflect any sentiment of the users who posted them, although someone may argue that these tweets still reflect the quality of trams. Furthermore, there are some tweets with identical content, but they are @ (mentioning) different users. These identical tweets, if in large quantities, has the potential to overshadow other data points in our statistics. Some tweets about trams also appear to be replies to other tweets. This poses challenges to sentiment analysis due to the increased lack of context. The occasional sarcasm in tweets and the censorship of sensitive words (replaced by #) further increase the difficulty of correctly analyzing the sentiments.

The last constraint on our sentiment analysis is that more computationally heavy models like BERT [10] were not used due to the lack of resources (i.e., no GPUs allocation on MRC). An improvement in the accuracy of sentiment analysis may be possible if those models were to be used.

7 Individual Contributions

Name of the person	Contributions
Bingzhe Jin	Twitter harvesters Sentiment analysis Web frontend AURIN Video editing
Hongwei Chen	Web frontend & backend
Tian Hui	Docker CouchDB cluster setup Twitter harvesters Demonstration
Zhen Cai	System architecture design MRC instance creation Identification of tweet statistical areas CouchDB views Demonstration
Ziqi Zhang	AURIN

8 Conclusion

In conclusion, this project builds a system to harvest tweets from Melbourne and analyse public opinion on the Yarra Trams. The most recent results suggest a positive correlation between population and the temper score. Though a time-wise saw a recent decline in the polarity score, it seems that most Melburnians are satisfied with Yarra Trams. Therefore, we believe Melbourne is a livable city.

Due to the limited time available for gathering tweets, the data utilized for analysis is limited, and future results may differ from current ones.

References

[1] (n.d.). *Melbourne Research Cloud Documentation.* https://docs.cloud.unimelb.edu.au/ [Accessed 3 May 2022]

[2] Apache CouchDB® 3.2 documentation. (2021). *2.2. Cluster set up.* https://docs.couchdb.org/en/3.2.0/setup/cluster.html [Accessed 30 April 2022]

[3] Use Cases, Tutorials, & Documentation — Twitter Developer Platform. (2022, February 3). *Search tweets - How to build a query.* https://developer.twitter.com/en/docs/twitter-api/tweets/search/integrate/build-a-query [Accessed 8 May 2022]

[4] Apache CouchDB® 3.2 documentation. (2021). *3.5. CouchDB HTTP server.* https://docs.couchdb.org/en/3.2.0/config/http.html#cross-origin-resource-sharing [Accessed 13 May 2022]

[5] Public Transport Victoria. (2021). *PTVH5514 Yarra Tram Network Maps 2021.* https://www.ptv.vic.gov.au/assets/PTV-default-site/Maps-and-Timetables-PDFs/Maps/Network-maps/PTVH5514-Yarra-Tram-Network-Maps-2021.pdf [Accessed 13 May 2022]

[6] SenticNet. (n.d.). *Sentic APIs.* https://sentic.net/api/ [Accessed 9 May 2022]

[7] Department: Affective Computing and Sentiment Analysis. (n.d.). *The Hourglass Model Revisited.* https://sentic.net/hourglass-model-revisited.pdf [Accessed 9 May 2022]

[8] Australian Bureau of Statistics. (2021, July 20). *Digital boundary files.* https://www.abs.gov.au/statistics/standards/australian-statistical-geography-standard-asgs-edition-3/jul2021-jun2026/access-and-downloads/digital-boundary-files#downloads-for-gda2020-digital-boundary-files [Accessed 12 May 2022]

[9] Sinnott, R. O., Bayliss, C., Bromage, A., Galang, G., Grazioli, G., Greenwood, P., Macaulay, A., Morandini, L., Nogoorani, G., Nino-Ruiz, M., Tomko, M., Pettit, C., Sarwar, M., Stimson, R., Voorsluys, W., & Widjaja, I. (2014). The Australian Urban Research Gateway. *Concurrency and Computation: Practice and Experience, 27(2)*, 358-375. https://doi.org/10.1002/cpe.3282

[10] Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2018). Google AI Language. *Bert: Pre-training of deep bidirectional transformers for language understanding.* https://arxiv.org/abs/1810.04805 [Accessed 15 May 2022]