COMP90024

Cluster and Cloud Computing

Team 8 / Melbourne

Australian Social Media Analytics

Team 8

Ang	Li	(631317)
Yiwen	Zeng	(715874)
Xuelan	Zhou	(806252)
Fangfang	Huang	(851654)
Tiancheng	Zhu	(894481)

May 10, 2018

Contents

1 Introduction

1. Demo https://www.team8.xyz/
2. Git Repo https://bitbucket.org/anglee/cloud/
3. Video https://youtu.be/Joum198mFOw

The focus of this project is to harvest tweets from Greater Melbourne and undertake tweet analysis before persisting into CouchDB database. Afterwards, features are extracted from the big data sets by using MapReduce model and other data processing algorithms required by specific analysis scenarios. Eventually, scenarios are visualized on a web application.

In terms of devops, the entire system has the capabilities of dynamic deployment. Virtual machines can be automatically created and then all necessary software dependencies and applications will be subsequently installed.

1.1 Scenarios

Four scenarios were designed and implemented.

1. how sentiment varies with time and the periodic characteristic of human emotion
2. how rent affordability influences the overall sentiment of suburbs
3. the correlation between the number of tweets and the population demographic of different areas
4. whether a suburb with more sports facilities generates more tweets related to sports

1.2 Features

1. high fault tolerance
    - no single point of failure
2. two layers of load balancing with health check
    (a) round-robin DNS

   (b) reverse proxy

3. controlled access to CouchDB and in-memory caching
4. implementation on Ubuntu 18.04 LTS (released on 26 April 2018)
   (a) HTTPS and HTTP/2 enabled
   (b) BBR congestion control algorithm enabled

2 Architecture

2.1 Software Environment

The system incorporates the following off-the-shelf software components.

Ubuntu 18.04

Ubuntu is a popular Linux distribution in cloud computing ecosystems. As the latest long-term support edition, Ubuntu 18.04 lays a solid foundation for the project with both brand-new features and stability.

Compared to CentOS and other mainstream distributions, Ubuntu is more friendly for development. Our team benefited from the new features brought by the updated modules.

CouchDB 2.1

One powerful built-in function of CouchDB is **MapReduce**, an efficient technique to process Big Data in distributed system parallelly. In CouchDB, Map function is to filter relevant data, process and present them in indexed keys and output values in CouchDB views. The keys and values are kept in leaf nodes for CouchDB Reduce function to perform aggregation calculations on the output (emitted data) of Map function. For example, the built-in Reduce function `_count` gives the count of items, `_sum` can sum numeric values up, and `_stats` provides statistical metrics like `count`, `sum` and `sumsqr` (sum of squared elements).

CouchDB provides a **RESTful** interface for document manipulations including creation, retrieve, update and delete. These operations are done by POST, GET, PUT and DELETE

methods with URI. Unlike MySQL or MongoDB, no drivers are required.

Compared to MongoDB, CouchDB supports both master-slave and **master-master replication** while MongoDB offers only master-slave replication. Also, CouchDB uses **Multi-Version** Concurrency Control, which simplifies the synchronization between nodes.

Docker 17.12

Docker is employed during prototyping, to install CouchDB with higher flexibility. Such an approach makes it possible to launch multiple CouchDB instances with manipulated port mappings on the same physical server, and even attach different data directories. Besides, packaging and delivery could also be simplified. Some aspects of flexibility are crucial for diagnoses mentioned later.

Python 3.6

Python claims to be explicit and simple while Java is more verbose. On the other hand, Java has higher execution efficiency than Python. Our Twitter harvester does not consume many computational resources. When there is a trade-off between execution efficiency and development efficiency, we prioritize development one. Additionally, Python also has a mature **ecosystem** like packages and community support for data analysis. Off-the-shelf solutions like Tweepy, Shapely, CouchDB are at our fingertips.

Nginx 1.14

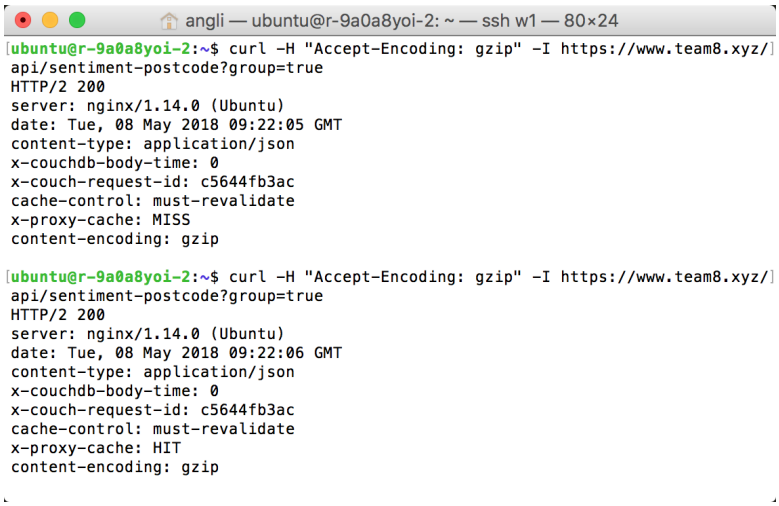


Figure 1: cache, HTTP/2 and gzip enabled by nginx

Nginx is an HTTP and reverse proxy server. We use nginx to serve HTML files and as an load balancer for CouchDB. HTTPS and HTTP/2 are enabled here and JSON responses from the CouchDB are also cached here.

2.2 Architecture

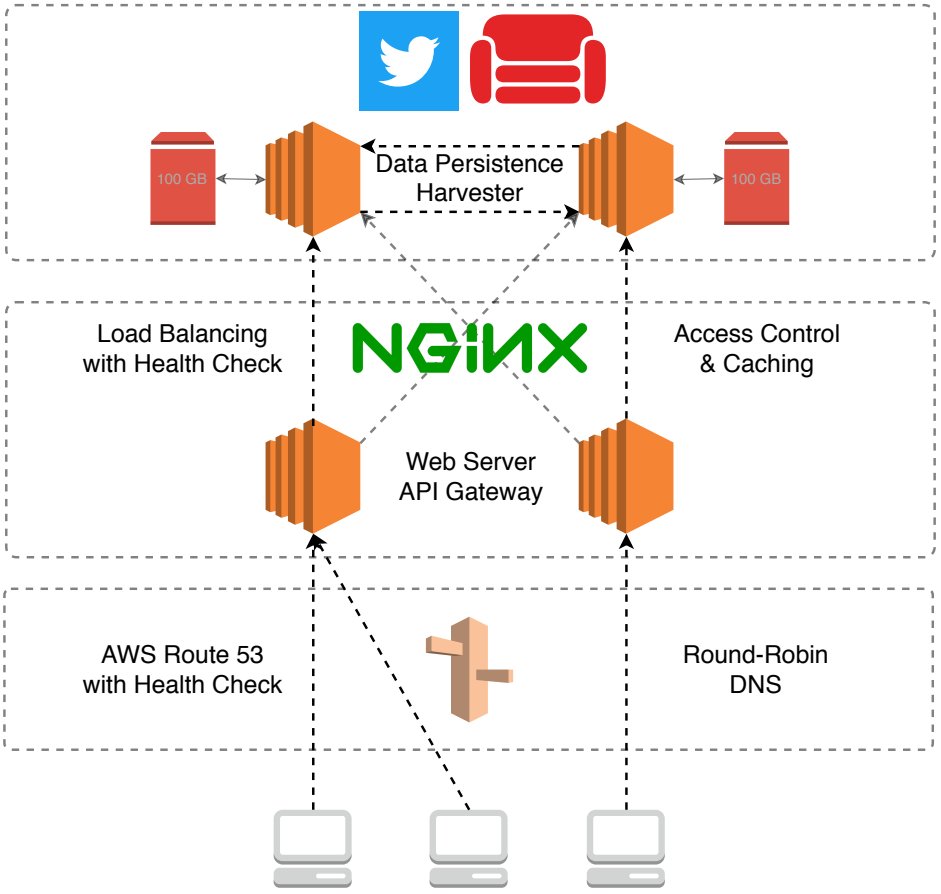


Figure 2: System Architecture

The system design adopts a three-tier architecture. The first tier is the front-end web app in charge of presentation, which will be introduced in Chapter 5. The second layer contains two physical servers. Both servers act as web server and CouchDB API gateway. The third layer has two servers as well as two instances of block storage. Both server host instances of CouchDB and run the Twitter harvesters. Two databases are automatically synchronized by CouchDB.

Resource Allocation

We equally allocated 32 GB of RAM and 8 cores to four identical virtual machines, i.e. 2 cores with 8 GB of RAM per server. For the two servers in the data persistence layer, each of them is attached with a volume of 100 GB.

Round Robin DNS with Health Check

In order to enable HTTPS and for convenient access, we spent $ 1 on registering a domain name `team8.xyz`. Amazon Web Services Route 53 was selected as the Domain Name System. We routed `www.team8.xyz` to the IP addresses of two web servers. Both servers are associated with Health Check. An IP address is returned record only when the endpoint is healthy. If both servers are healthy, two IP addresses will be returned in a random order.

As a result, traffic are balanced to both web servers.

Web Servers and API Gateways

Web servers host the HTML files of the front-end. Moreover, they also act as API Gateways of the CouchDB API.

Access Control

Only `GET` and `HEAD` methods are allowed, otherwise `405 Method Not Allowed` will be returned. This mechanism will ensure CouchDB will be safe from accidental "create", "update" and "delete". Also, `team8.xyz/api/` directory is forwarded to `/tweets/_design/designDoc/_view/` on the CouchDB. Only aggregated views are accessible while details about each tweet are protected.

Load Balancing with Health Check

Request to CouchDB views will be forwarded to two CouchDB servers on a round-robin basis. Once a server is not responding, traffic will not be forwarded to this server for a certain period. Information will be retrieved from the other server.

This mechanism ensures that load is balanced and as long as one server is operating, CouchDB views can be served.

Caching

Responses with `200 OK` status from the CouchDB will be cached in the shared memory (`/dev/shm`) for a certain period. This masochism not only reduces the load on CouchDB but also shortens

the response time to the browsers. If both CouchDB servers are down, a stale cached response will be returned to the client.

Database Servers

Each database server is associated with a block storage volume of 100 GB. A Twitter harvester is running on each of the database servers. A CouchDB cluster will automatically synchronize documents on both nodes.

2.3 Fault Tolerance & Scalability

As discussed above, both web servers and database servers have redundancy and health check. If one web server is down, traffic will be routed to the healthy endpoint by DNS. Similarly, if a CouchDB server is down, requests will be forwarded to the healthy node. In the worst case, if two CouchDB servers and one web server is down, the only healthy web server will still be able to return stale cached information to the clients.

The system also has scalability. If volume space is being run out, we can increase the capacity of volume storage. If web servers are overloaded, we can increase the number of web servers and migrate static files to object storage and distribute them over Content Delivery Network (CloudFront, Akamai, etc.). If CouchDB servers are struggling, we can improve the cache mechanism and increase the scale of the CouchDB cluster.

3 Dynamic Deployment

3.1 Infrastructure

We prepared the virtual machines according to the following process.

Configure Security

The firewall was configured before system deployment. Port 5984 is used by CouchDB. Erlang uses TCP port 4369 to find other nodes. Ports 9100-9200 are required by communication between CouchDB nodes. Also, port 22 is essential for SSH connection. Port 80 is open for

HTTP and port 443 is open for HTTPS.

Also, SSH key was generated and imported. This SSH key will be used by Ansible to connect to the servers.

Create Instances

`boto.ec2.connection.run_instances()` method will create instances based on the parameters given by us.

1. we requested at least and at most four servers (i.e. exactly four servers)
2. we requested the instances to be created in 'melbourne' availability zone
3. the type of instances is 'm1.medium'
4. we chose the image with id `ami-bf99c82f` (Ubuntu 18.04 LTS)
5. we specified the ssh key and the list of security groups

The power status of the instances will be iteratively inquired, instances will be considered as ready when all of them return the power status code of "16". A list of so-called `private_ip_address` can be obtained after instances are successfully created. These IP addresses will be used by Ansible.

Create Volumes

We allocated two instances as web servers and two instances as CouchDB servers. Only two volumes will be created in the "melbourne-np" availability zone.

We got the ids of volumes from the objects returned by the `create_volume()` method and kept querying the status of created volumes. Once the status of each volume is "available", volumes are successfully created.

Attach Volumes to Instances

We start to attach volumes to instances after all of them are ready. The `attach_volume()` method is used to attach volumes to instances. We attach the volumes at `/dev/vdc`. We

kept retrieving the `attach_data.status` property of `Volume()` object. Once all of them equal `attached`, the infrastructures are ready.

3.2 Software Deployment

After virtual machines are configured, we start to set up the software environment on each server with the help of Ansible. As per the system architecture, servers are divided into two groups: web servers and database servers.

For the purpose of dynamic deployment, variables like IP address and credentials are filled by Python script and Ansible playbook. All the credentials are centrally placed in the `/credentials/` directory.

Package Upgrade

We firstly update and upgrade APT packages on all server. APT is a package management system on Ubuntu.

Block Storage

We partition, format and mount attached volume to the system on CouchDB server with the help of `parted`, `filesystem` and `mount` Ansible modules.

Network Optimization

Bottleneck Bandwidth and Round-trip propagation time (BBR) is available from Ubuntu 18.04. We take advantage of this model-based TCP congestion control algorithm to provide a smoother transmission. fq_codel (fair queuing controlled delay) discipline is adapted to reduce bufferbloat and TCP Fast Open is enabled in both directions to speed up the opening of successive TCP connections between two endpoints.

`modprobe`, `lineinfile` and `sysctl` modules are used.

Nginx

We install and configure nginx on web servers.

1. install nginx (`shell`)
2. upload SSL certificate (`copy`)
3. upload static files (`copy`)
4. upload nginx configuration file (`copy`)
5. reload nginx with the new configuration file (`systemd`)

Docker

We install Docker on database servers for deployment of CouchDB.

CouchDB

We run a Docker container of CouchDB on database servers with the following configuration.

1. map 5984, 5986, 4369 and 9100-9200 from the container to the server
2. specify the node name
3. specify the username, password and secret cookies
4. map the data directory to the mounted volume

Afterwards, the Ansible playbook runs curl command to

1. add nodes to the cluster
2. finish cluster
3. create database
4. create views

pip packages

We firstly install the package management system `pip` and the following packages on database servers with the aid of Ansible module `pip`.

1. `pytz` (convert UTC to local time with daylight saving offset)

2. `CouchDB` (encapsulation of CouchDB REST API)
3. `tweepy` (encapsulation of Twitter API)
4. `Shapely` (classify coordinates into postcodes / local government areas)
5. `textblob` (sentiment analysis)

Harvester and System Service

As described in the system architecture, harvesters are running on database servers.

1. clone git repo from Bitbucket (`git` module)
2. upload credential of Twitter and CouchDB (`copy` module)
3. upload system service configuration (`copy` module)
4. register as a system service and start the job (`service` module)

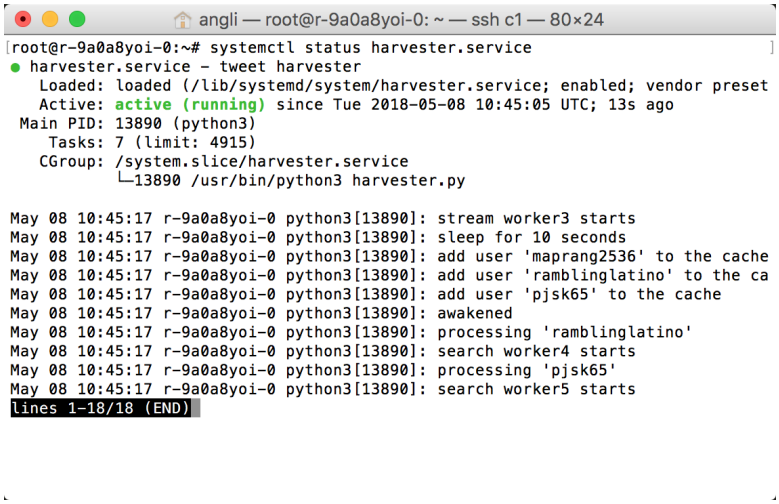


Figure 3: Harvester Running as a System Service

3.3 User Guide

change to working directory

```
cd deploy/
```

run the Python script

```
python3 nectar.py
```

This script will create instances an volumes on NeCTAR and generate configuration files with IP addresses and credentials ready to be used by Ansible playbook. Number of instances for web and database can be separately specified in the script.

run the Ansible playbook

```
ansible-playbook -i credentials/hosts nectar.yaml
```

This script will install required software environment and automatically deploy and run the applications (e.g. web server / CouchDB / harvester).

4 Twitter Harvesting and Analysis

4.1 Harvester

There are two types of Twitter APIs available for data collection from Twitter, which are Streaming API and Search API. Streaming API is mainly used to pull tweets created in real time. Developers are able to set criteria and tweets satisfying the criteria will be returned. Search API, unlike the Streaming API, can query and return tweets in the past that satisfy a certain search criteria. Developers could search tweets by a username, keywords, locations, and time etc. The twitter API library used in this project is called 'Tweepy', which is an open source Python Twitter API library supporting both Streaming and Search API. Both Streaming API and Search API are used in the harvester. In order to improve system overall efficiency, the tweet will be analyzed before stored to the database. Because we are only interested with the tweets posted in Melbourne, the filter used in Streaming API is set to the geolocation of Melbourne. Additionally, tweets without coordinates will be directly discarded.

Rate Limiting

One challenge in collecting tweets is the rate limiting. For the Search API, the limit window duration per authentication key is 15 minutes and there are 180 maximum calls per window.

Both types of APIs require a set of API key, API secret, Access token and Access token secret to be authenticated. Twitter restricts the number of connections can be made to using the same authorization keys. In order to retrieve more Tweets, 6 sets of authentication keys were created for the harvester in this project. Moreover, the newest version of Tweepy has additional parameters to handle the rate limiting: `wait_on_rate_limit` is used to check whether or not to automatically wait for rate limits to replenish; `wait_on_rate_limit_notify` is used to print a notification when Tweepy is waiting for rate limits to replenish.

History Tweets

The other challenge in harvesting tweets is to collect old tweets. The Streaming API only returns real time tweets and search keyword in the Search API will only return tweets posted within 7 days. However, our dataset will be biased if we only consider the tweets posted recently. Hence, we combined the Streaming API with the Search API. The streaming API is used to collect stream tweets, for each tweet collected by Streaming API, the harvester will get the username of the tweet, then call the Search API to get the latest 50 posts from the user's timeline. However, in this approach the Streaming API needs to wait until the Search API finishes its job, which results in enormous pending overhead and unsatisfactory efficiency. The solution we found is to use multiple threads. In our harvester, the Search API and the Streaming API are working in parallel. Both the Streaming API and the Search API will be initialized. The Streaming API will start collecting data once it is ready. Once some tweets are returned, the username will be stored inside a cache of usernames. Duplicate usernames will be ignored. As long as the cache has items, the Search API will start to get the first username from the cache set and start to download latest 50 tweets of that user. This approach avoided the blocking waiting time.

Truncated Tweets

Although Twitter allows user to post tweets no more than 280 words, When tweets are longer than 140 characters, the text returned by the API will be truncated. This is a serious problem when analyzing the text of tweets. Our approach to solve this issue is to set the `tweet_mode` of API to `extended`. There is an attribute in the returned Twitter Object named `full_text`. When truncated is equal to true, the harvester will save `full_text` as the text of the tweets,

otherwise it will save the text attributes of the Twitter Object.

Duplicate Tweets

The tweets returned by the Twitter API contains a lot of duplicates data, therefore it is essential to discard them before putting into CouchDB. Inherently, CouchDB allows user to set the document id (`_id`). The main approach we have adopted is to use the id of the tweet as a unique identifier of the CouchDB document. Before we conduct any analysis, we try to fetch the document from the database. If the document already exists, this tweet will be directly ignored. Also, when saving to CouchDB, if the document id already exists in the CouchDB database, an error of `ResourcesConflict` will be raised. We try to catch the exception if the document already exists.

Retweets

Twitter users not only post their own tweets but also share others tweet, that is retweet. For this project, we simply ignored all retweets. Therefore, any tweet that has the attribute `retweeted` equal to `true` will be discarded.

4.2 Tweets Analysis

Some basic information of the tweets, including the sentiment score, vulgar words and tweet postcode / local government areas were analysed. The analysis was packed into tweets harvesting process, so that the updated tweets data with new fields can be provided as shown in Figure 4.

```
"bad_words": 0,
"sentiment": 0.7000000000000001,
"hour": "19",
"text": "@Uyghurkizi your welcome :) yahxi turwatamsiz?  We chat din amdi twitter ga quxtuh a lol",
"created_at": "Wed Aug 15 09:18:08 +0000 2012",
"local_time": "2012-08-15T19:18:08+10:00",
"coordinates": {
  "type": "Point",
  "coordinates": [
    144.9177444,
    -37.634445
  ]
},
"day": "Wed",
"postcode": "3064",
```


Figure 4: Part of an Updated Tweet Document in CouchDB

Sentiment Analysis

Considering of analysis speed, operation complexity and the ability to analyze text emotions, TextBlob is made use of to calculate the sentiment polarity score of tweet texts. TextBlob can be treat as string in Python, and it is one of the most simple-manipulated Natural Language Processing (NLP) Python libraries offering consistent API based on NLTK and Pattern library. The methods of TextBlob includes tokenization, POS-tagging, sentiment analysis, etc. In this project, Tweet texts were converted into `textblob` objects and the `sentiment` method was called to calculate sentiment polarity score, which is a float in the range [-1.0, 1.0]. The higher the score, the happier the text analyzed to be.

Emoticon analysis

TextBlob, the NLP package used in this project, supports emoticon sentiment analysis well. positive emoticons like :) and :D gain obvious positive numbers, and vice versa. The tweet texts those emoticons inserted in would be practically affected by them.

Limitations

In fact, human languages are more complex than algorithm can understand. In many cases, people express in a sarcastic way but the words are sounds positive, which puzzled the machine.

Geolocation Classification

Because the scenarios are discussed between areas distinguished by postcodes or local government areas (LGAs), but only the coordinates of tweets are known, a tweet locater is needed to locate the tweets. AURIN provides the area bounding points of both postcodes and LGA standard of division. In this case, Shapely package, a Python package focuses on planar geometric objects analysis and manipulation is the best tool to do it. With Shapely, the coordinates group of an area form a polygon object, and search in which polygon the tweet location point is within. If any tweet location is out of the bounds that incorporate all the areas of interest, the output returned `-1` for easy filter later.

5 User Interface

5.1 AngularJS Web App Architecture

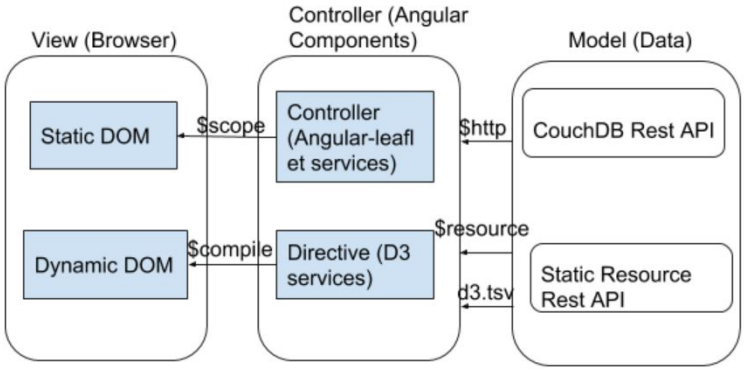


Figure 5: Front-end Architecture

We implement a 2-tier web architecture using Angular framework for front-end components to interact with CouchDB as well as static resources by REST API calls. Aggregated data generated from CouchDB MapReduce functionality can be accessed via RESTful calls and this feature eliminates the needs for a backend implementation. Therefore a Model-View-Controller pattern is adopted in our design where angular controllers interact with static resources and CouchDB database via `$resource`, `$http`, d3.tsv service objects and enable data-binding with HTML views via `$scope` object. These creates a feasible solution for asynchronous data invocation with d3 service rendering the template only after `$resource` or `$http` API calls succeed.

The templating part facilitated by d3 and data content part from REST calls are separated and this is inherent with the idea of separation of concern.

5.2 D3 Data Visualization

Data Driven Document Js library is wrapped in Angular directive to render SVG components in HTML pages. Angular directive provides DOM manipulation api for D3 service to add or subtract HTML elements while restricting all access. D3 library provides easy manipulation of DOM elements and plotting feasibility via svg graph. Integrating d3 library with Angular is done via Angular directive that provides functionality of a link function that binds a specific DOM elements to a certain behavior and data driven template facilitated by D3 services.

5.3 MVC Pattern

Our design of MVC pattern incorporates the idea of separation of concern. The advantage of such design is that if there are multiple views displaying same set of data, once the data changes very views will be notified and the pages will change as well. The two heatmap charts all utilize the same data set from couchDB view and if data changes the views get updates as notified.

6 Twitter Activity & Sentimental Trends

In this scenario, we are seeking the connection between the sentiment change and the post time.

6.1 CouchDB View

As mentioned in previous section, the MapReduce function provided by CouchDB played an important role in data analysis.

One condition needs to be met before a tweet is further processed, i.e. the postcode of the tweet should be equal to or larger than 3000. This assured the tweet is posted within greater Melbourne area. The key emitted by the map function is a pair constituted by ‘day of week (from Mon to Sun) and ‘hour of the day(from 00 to 23), and the value is the sentiment score

of the tweet. An example of the output is {"key": ["Fri", "00"], "value": 0.4}.

In order to obtain a better efficiency, the build-in `_stats` reduce function is applied, which is implemented in Erlang instead of JavaScript. This function returns the sum of sentiment scores, the count of tweets and the sum square of sentiment scores. When querying this view with parameter `group=true`, CouchDB will aggregate all the documents by the key pair.

We use average sentiment score as an indicator of the overall sentiment of an hour h of a day d.

$$\mathrm{E}\left[sentiment_{d,h}\right] = \frac{sum_{d,h}}{count_{d,h}}$$

Also, we derive the standard deviation to describe how far a set of sentiment scores are spread out from their average value.

$$\sigma\left[sentiment_{d,h}\right] = sumsqr_{d,h} - \left(\frac{sum_{d,h}}{count_{d,h}}\right)^2$$

6.2 Heatmap

We employ a heatmap to visualize the difference of average sentiment scores in different time slots since the layout of color spectrum easily captures the discrepancy of different sentimental behavior.

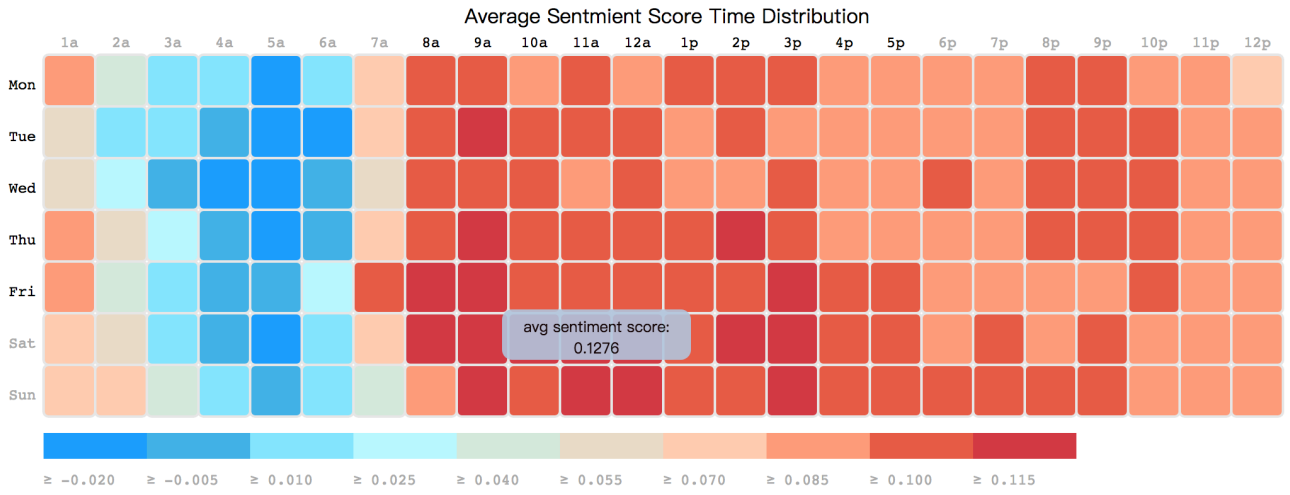


Figure 6: Average Sentmient Score Time Distribution

From Figure 6, we can roughly conclude that tweets seem happier on Saturdays. If we group by day and evaluate the mean, we can clearly see from Figure 8 that Saturday, together with

Sunday and Friday, constitutes a sentiment peak. This result is reasonable because people are more relaxing during the weekend. On Monday, the beginning of five working days, people are more hopeless.

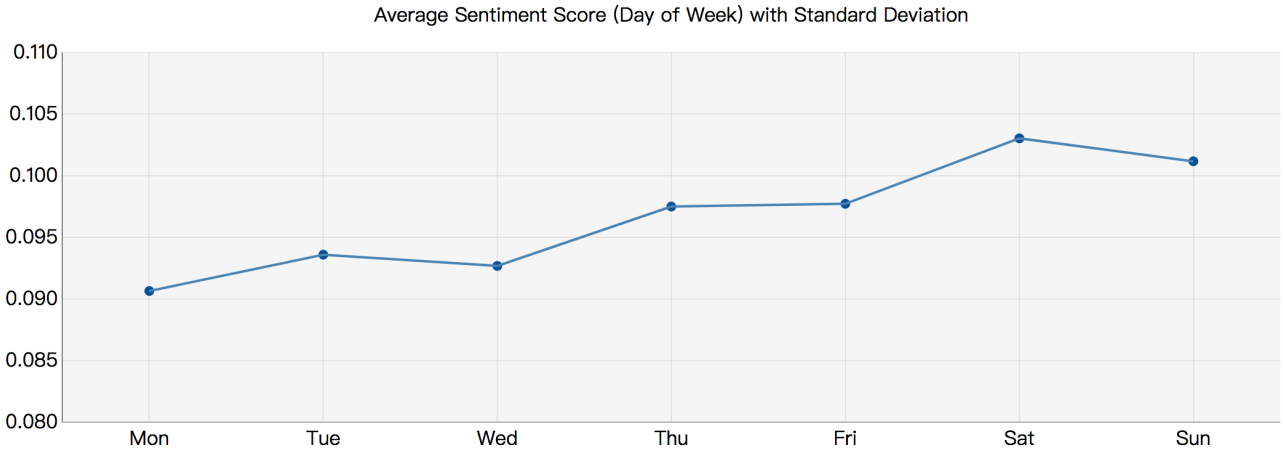


Figure 7: Average Sentiment Score (Day of Week) with Standard Deviation

Similarly, from the heatmap, we find people are down late at night. If we group by hour and calculate the average, from Figure 8 it can be clearly seen that tweets posted at 5am are the most negative ones and people are happiest around 9 am. The result is reasonable. For example, people suffering from insomnia / on the night shift / who have to get up very early tend to express negatively.

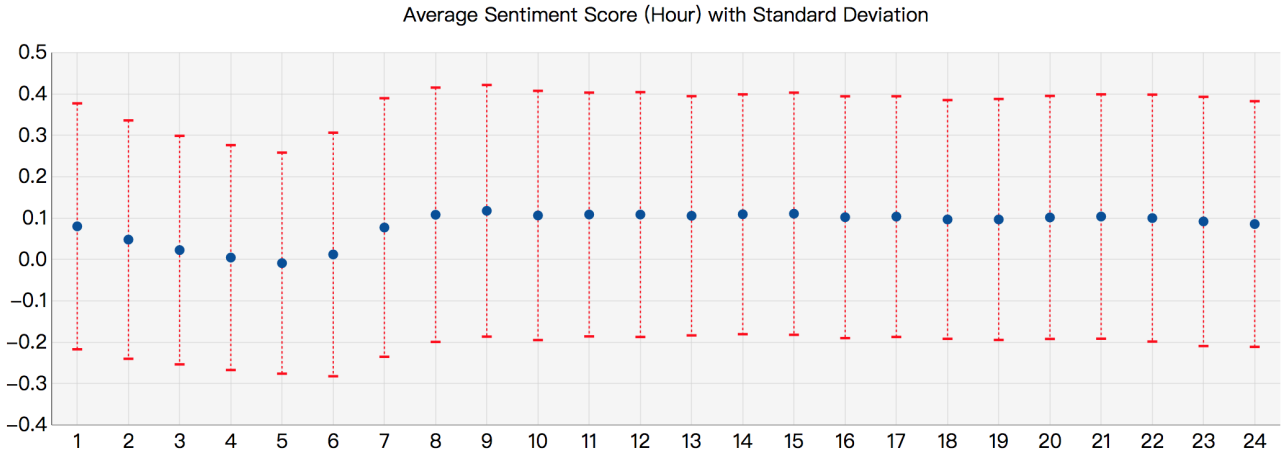


Figure 8: Average Sentiment Score (Hour) with Standard Deviation

6.3 Limitations and Potential Improvements

One limitation of our findings is that in Figure 8 the difference between the peak and the valley (≈ 0.1) is much smaller than the standard deviation (≈ 0.3), although the large standard deviation can be explained by considering people might be experiencing different situations. In addition, the standard deviation at the lowest point (5 am) is exactly the smallest, which may indicates we do not have sufficient samples.

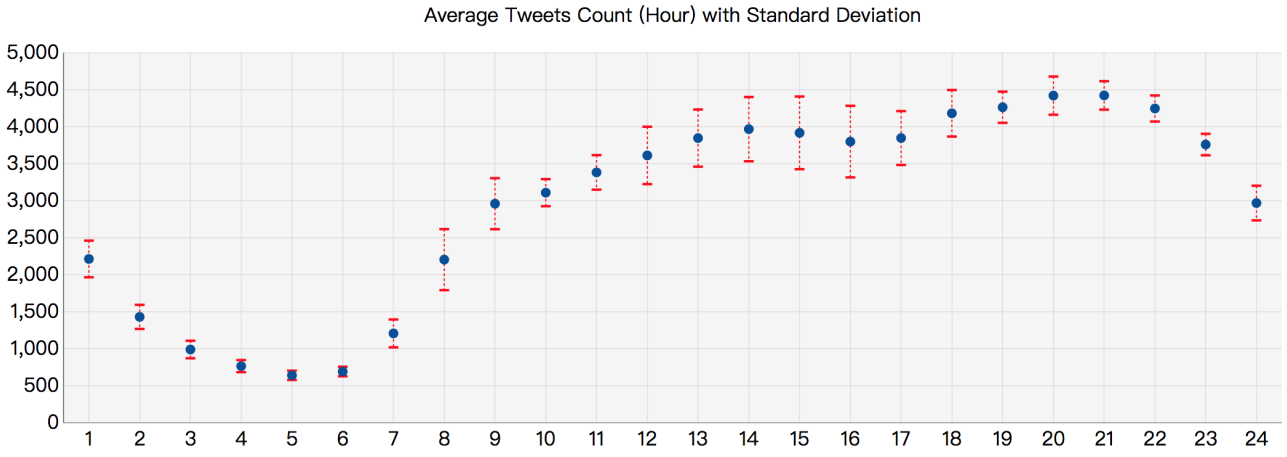


Figure 9: Average Tweets Count (Hour) with Standard Deviation

If we plot the amount of tweets from Monday to Sunday (Figure 9), we will find the sentiment is correlated with the amount of Tweets. For example, Saturday has the largest amount and Monday has the least. However, there are some discrepancies. Specifically, Wednesday is a valley of sentiment but a peak of amount. Also, Sunday has fewer tweets than Friday but is happier than Friday. The relationship between the amount of tweets and the overall sentiment can be investigated by mathematical methods like cross-correlation and least mean square algorithm.

6.4 Spectral Analysis

As it is shown in Figure 6, the sentiment trend repeats the same pattern every day. From the dawn to the morning, the sentiment score raises from the bottom to the peak and remains happy for several hours. In the afternoon, the sentiment score drops slightly and reaches a smaller climax during the TV prime time. Afterwards, the sentiment score drops again until reaching the valley and another cycle begins.

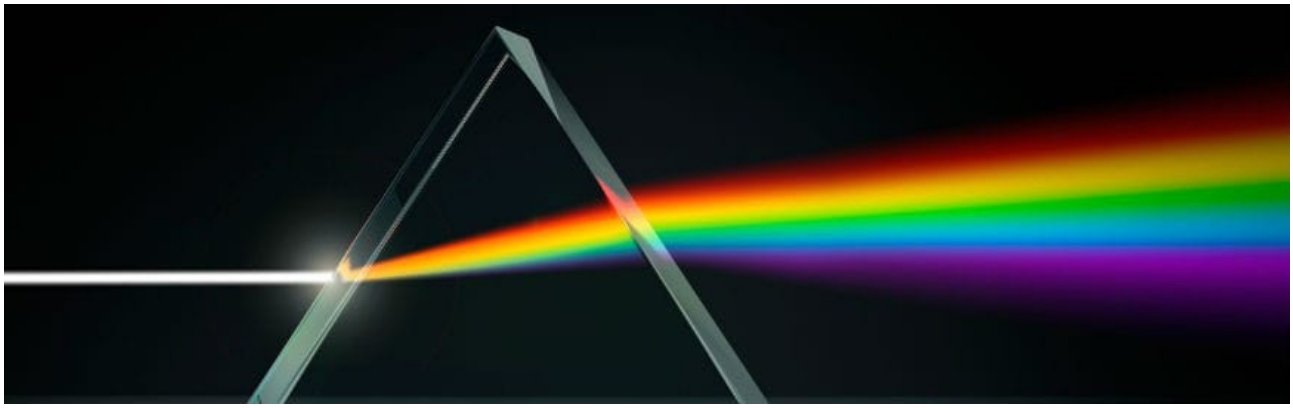


Figure 10: Dispersion of Light by Prisms (from theeyepractice.com.au)

We can simply conclude that the sentiment trend has a period of 24 hours. However, more details need to be discovered. Like a prism dispersing the white light into ROYGBIV, spectral analysis can be conducted on the Twitter sentiment trend.

To construct a time sequence $s[n]$, we firstly concatenate the sentiment scores of 24 hours from 00 to 23 then from Mon to Sun, i.e. Mon 00 $\rightarrow$ Mon 01 $\rightarrow$ $\rightarrow$ Mon 23 $\rightarrow$ Tue 00 $\rightarrow$ Tue 01 $\rightarrow$ $\rightarrow$ Sun 22 $\rightarrow$ Sun 23. As a result, a time series of 168 points is obtained. Then, we conduct the Discrete Fourier Transform on this signal and obtain $\hat{S}[k]$ the magnitude information of the spectrum. Specifically, $\hat{S}[0]$ stands for the Direct Current component and $\hat{S}[84]$ corresponds to the Nyquist frequency (double of the sampling frequency, 2 hours).

$$s[n] = \frac{sum_{d,h}}{count_{d,h}} \qquad n = 24d + h = 0, 1, \ldots, 167$$

$$\hat{S}[k] = \sum_{n=0}^{167} |s[n]| \sqrt{\left(\cos\frac{2\pi kn}{168}\right)^2 + \left(\sin\frac{2\pi kn}{168}\right)^2}$$

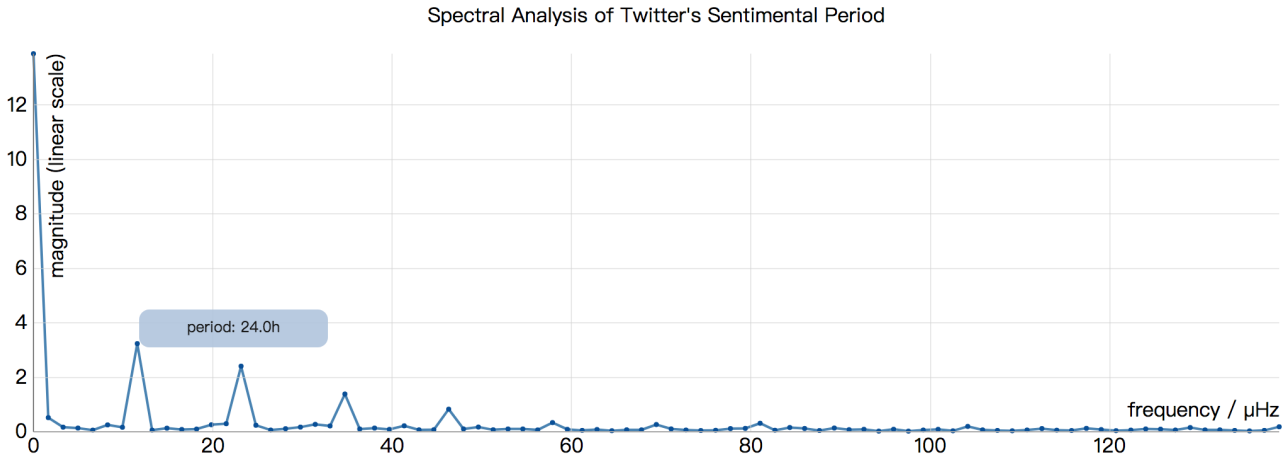


Figure 11: Spectral Analysis of Twitter's Sentimental Period

In Figure 11, a dominant peak can be clearly seen at 11.57 µHz which corresponds to 24 hours of time. Afterwards, a peak happens at 13.15 µHz representing 12 hours followed by 34.72 µHz (8 hours). The 24h peak stands for the variations associated to the **daily cycle** as what we expected. The 12h peak is due to **the division of day and night** and the 8h period is associated with **the shift between work, life and sleep** [1].

7 Twitter Sentiment & Rental Affordability Index

7.1 CouchDB View

The view created for this scenario is called `sentiment-postcode`, it is used to provide information to compare the rental affordability with the sentiment score of the tweet. The sentiment score is aggregated according to the postcode of the tweet. The key is the postcode and the value is the sentiment score of the tweet. The reduce function used is build-in function `_stats`. When query from this view, the post within melbourne area will aggregate according to the postcode and return the sum of the sentiment score, count of tweets and the sum square.

7.2 Choropleth Graph

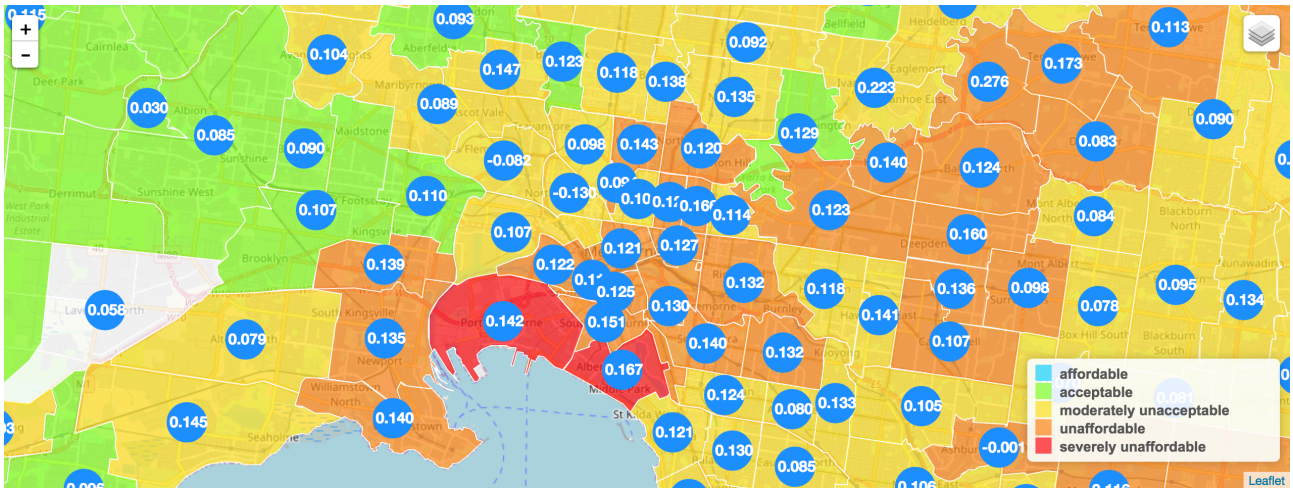


Figure 12: Rental Affordability Index (2017 Q2) vs Average Sentiment Score per Suburb

Choropleth graph is rendered as an overlay layer on top of the street map layer and this is achieved by angular-leaflet directive. Aurin data is fetched via `$http` asynchronous call before the overlay layer is rendered, and the sentimental score for each suburbs is separated generated by an extra layer where data is accessed from CouchDB view function. The Choropleth visualize the correlation between the render of one suburb versus the larger its number is. This provides an interactive experience to explore the map in Melbourne region with feasibility to look into each suburb for individual comparison between its rental affordability and its average sentiment score. But the drawback is also clear in this form of visualization that it is less visually intuitive to find out the correlation between lower rental affordability and higher sentiment score.

7.3 Sentiment

The calculation is based on suburb with two phases, the first is to attached each tweets with properly calculated sentiment score a suburb postcode and the second is to calculate the centroid point of each suburb for the average sentimental score to be rendered onto the map. Both two were data preprocessing phases happening before the data asynchronous accessed and template rendering.

8 Twitter & Population

8.1 CouchDB View

The view used for this scenario is named `population-lga`, it is used to return the total count of tweets within the area of Melbourne. Unlike the first two scenario, the key of the map function is the LGA code (Local Government Area). Because the data of the population of Melbourne is come from AURIN and it used LGA to divide different areas. In the database, if according to the coordinate, the location of tweet was not posted in Melbourne, then the `lga_code` is equal to -1. The map function will first check if the document's `lga_code` is greater than -1. The value of the map function is the sentiment score. The reduce function is the build-in function `_stats` as well. The total amount of tweets within a particular LGA will returned when query the view with group=True.

8.2 Visualization & Analysis

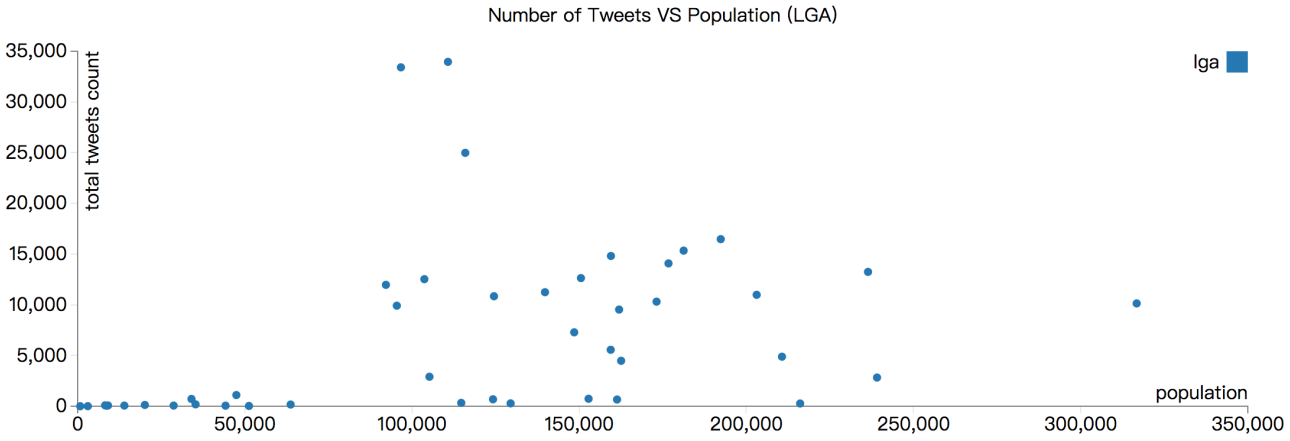


Figure 13: Number of Tweets vs Population (LGA)

This chart excludes one point for City of Melbourne since this particular LGA area has exceedingly more tweets posts than any other area, therefore to better visualize the correlation between these two variants we leave out this point for the y scale to fall within a reasonable regions that most of the points fall into. Correlation between the number of tweets posts in a LGA area and its population can be inspected from the plot with a general tendency that the more population is accompanied by more number of tweets posts in a LGA area. Some outliers exist in the plot with some special cases such as Port Phillip and Yarra with significantly

larger number of tweets counts. Possible reason behind this is that these inner metro areas may attract more tourists and more Internet activities than the rest of the areas.

9 Twitter & Sports

According to a precious review, there is a positive association between the availability of sports and recreation facilities and people's sports participation [2]. Other researches also show a positive correlation between the presence of sport facilities and people's sports participation in Europe [3] and Hong Kong [4]. Thus, there is a motivation to research whether this association also show up in Melbourne.

To combined with Melbourne twitter analysis, it is guessed that the people live in areas with more facilities would be more interested in sports. The supporting dataset is Victoria Sport and Recreation Facility Locations from AURIN. It covers about 80 types of sport and recreation facilities including public park, golf club, tennis court, swimming pool, recreation center and private fitness centers and gyms, etc. with the postcodes of areas they belong to.

9.1 CouchDB View

The Map function of CouchDB was used to query tweets whose text is related to sports that possibly played in these facilities, including running, frisbee game, skate, golf, tennis, etc. as well as the correlated special characters in Unicode.

A view named `sports` is built, whose key is the area postcode. As for value, if a tweet text matches any of the pattern above, the document value is set as `1`, otherwise it is a `0`. This is for making better use of the Reduce function. The `_stats` Reduce function provides the total count of areas and their corresponding value sum.So, the percentage of tweets related to sports in an area can be easily calculated by sum/count.

9.2 Scatter Plot

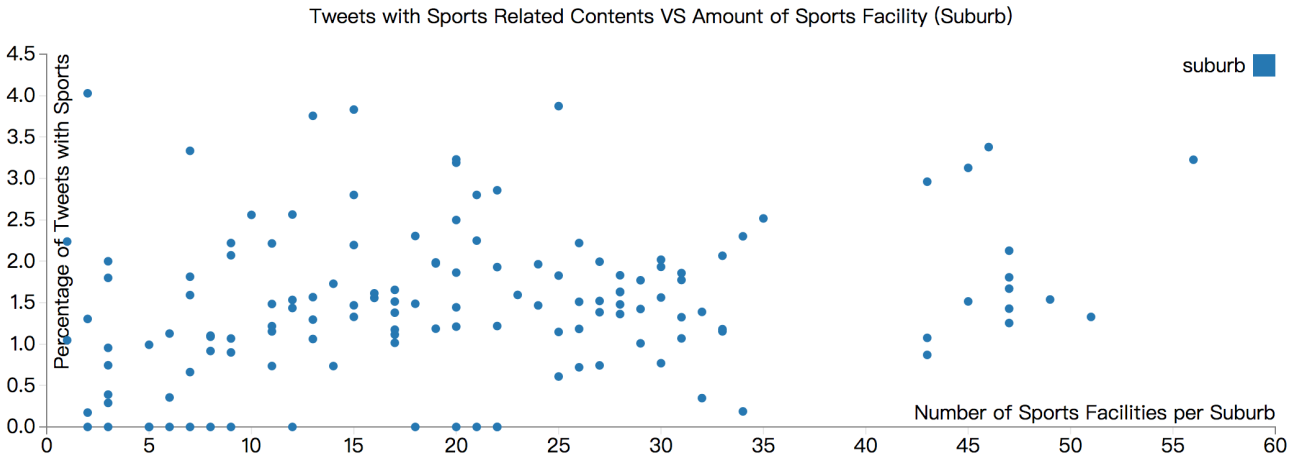


Figure 14: Tweets with Sports Related Contents vs Amount of Sports Facility (Suburb)

Among the total of 170 suburbs where number of sports facility is calculated from AURIN data, we calculate the percentage of tweets that mention sports with coordinate located at each suburbs. We sort all the percentages and exclude the least 10% of percentage of the data as well as the most 10% of the data, leaving only 80% in the middle of the percentage range and visualize them in scatter plot. We also excludes suburbs with facility counts larger than 60 to restrict the x axis range to be smaller where majority of points fall within. Slight correlation can be found by visual inspection of the graph that more facilities in a suburb is accompanied by more tweets mentioning sports.

9.3 Analysis

Sports and recreation facilities truly affect people's participation and interest toward sports, although it is not distinct. As is shown in the scatter plot above, apart from a few outliers which are or obviously much larger than most of the percentages, there is a slight positive relationship between the percentage of tweets of sports and the number of sports facility in a suburb. Most of the percentages are between 0 and 3%. With the facility number increasing, ignoring the zero percentage points, an empty area in the lower part near the x axis of the plot appears, which implies a growing trend. The number of tweets gathered from a suburb introduces the bias. For example, the possibility of zero sports-relative tweets is great if only few tweets are gathered in this suburb.

10 NeCTAR

10.1 Pros

Choices of Availability Zones

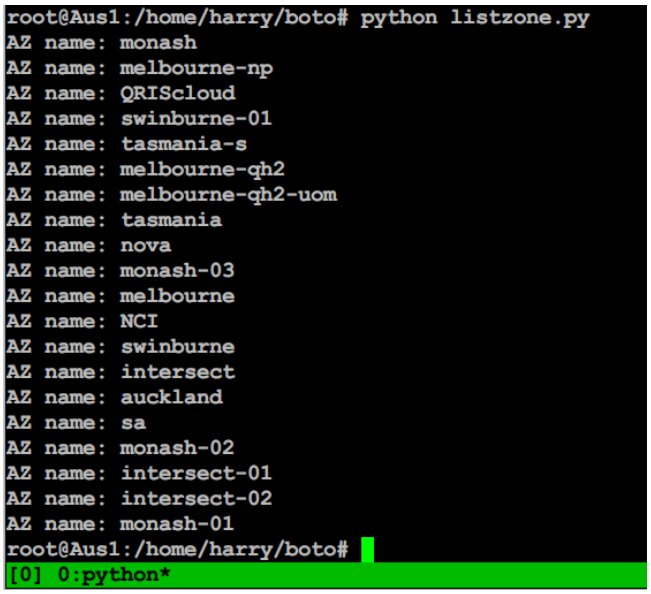


Figure 15: Availability Zones

NeCTAR provides over a dozen of availability zones nationwide (and in Auckland), many among which are in Victoria. That ensures the closest cyber endpoints possible and thus generates great efficiency for deployment. Multiple choices of AZs also has the potential to build geographically-isolated redundancies.

Images, Snapshots and Storage

NeCTAR official had an instant reaction to the final release of Ubuntu 18.04 during the project, which was even swifter than some commercial cloud providers. Customization of system images through snapshot function is also helpful. Volume storage function shows its flexibility by expanding an existing disk and mounting/dismounting a disk on different instances.

OpenStack APIs and Open-source Ecosystem

The NeCTAR Research Cloud uses OpenStack cloud computing platform. OpenStack includes compatibility with Amazon EC2 and Amazon S3 APIs and thus client applications written for

Amazon Web Services (AWS) can be used with OpenStack and NeCTAR Services with minimal porting effort. (http://training.nectar.org.au/package03/sections/all.html)

Among the compatible Amazon APIs, boto stands out as the foundation for automated deployment of this project. For more advanced projects leveraging more various cloud resources, OpenStack APIs imported in Python scripts will burst out greater extensibility and efficiency.

More importantly, OpenStack itself, as an open-source IaaS project, is a neutral and reliable platform that avoids potential lock-in or implicit security threats. Thanks to a flexible Apache 2.0 License, there are countless cases of deployment and secondary development among enterprises, which enhances the OpenStack ecosystem. Undoubtedly, for a long run, NeCTAR will enjoy more benefits from the ecosystem.

10.2 Issues and Challenges

Port Access and SSH Tunneling

For early-stage exploration and testing of CouchDB (without reverse proxies like Nginx), default port 5984 had to be visited frequently. However, sometimes in the 'melbourne-np availability zone, without sufficient official notice, the port was usually not accessible. To rapidly confirm the source of the problem, different port mappings for setting up a CouchDB Docker container were tried (like -p 5985:5984). This issue was initially stated in Thread 165 of discussion board. And, eventually for security reasons, ssh tunneling was suggested in Thread 168, but actually that still seemed to be not convenient for Windows users.

If we were timely informed that port 5984 was frequently attacked externally or blocked in some AZs by NeCTAR, hours might be saved from thoroughly tackling the issue.

Confusing API Variable Names

During the development of automation using python-boto. Some "abnormal" variables were brought to attention. As for the ip addresses, `Instance().private_ip_address` stores the actual public ip address of a NeCTAR instance referenced by `Instance()` object, while `Instance().public_ip_address` is always empty. Meanwhile, "Instance names" displayed on the dashboard

is equivalent to `Instance().private_dns_name`, and that name cannot be assigned by any parameters in the initial `run_instance()` function despite a subsequent chance of modification in the system. Besides, tagging system (`boto.ec2.connection.create_tags()`) is not displayed on the dashboard.

10.3 Potential Enhancement

Floating IP and Private Subnet

Most IaaS service providers have virtual gateways and floating IP sections, like Amazon VPC and EIP. Unfortunately, no floating IP quota was given to our project based on NeCTAR, which blew our idea of prototyping the overall system architecture in private IP addresses using Amazon EC2 and VPC. So, for this project. the flexibility of network on a IaaS platform did not seem to exist.

Without a virtual gateway, security standard might be brought down by increasing chances of exposing ports to the public, and efficiency of transmission between subnets from different masks might also be compromised. Without floating IPs, frequent launching and termination of instances turns out to be inconvenient for recording the IP information.

Object Storage and Volumes

Unlike Amazon S3, current Swift module and its dashboard on NeCTAR does not support access links generation in the frontend. This is a significant part that has the potential to be more user-friendly.

As for the volumes, most instances types have their storage capacity divided into two parts: root disk and temporary disk. But that generates extra effort in partitioning, formatting and changing mount points. By default this looks like a really strange scenario ,as the “temporary disk” looks less flexible than a block storage but acted as a data disk without formatting. (`/dev/vdb`, mounted on `/mnt`). NeCTAR should consider adding proper partition and formatting process of temporary disk, only partitioned and formatted `/dev/vdb1` instead of `/dev/vdb` should be considered as valid and persistent, which Thread 220 did not notice.

11 Appendix

11.1 Network Optimization Test

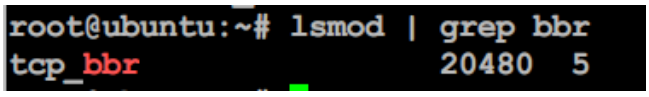
Disclaimer

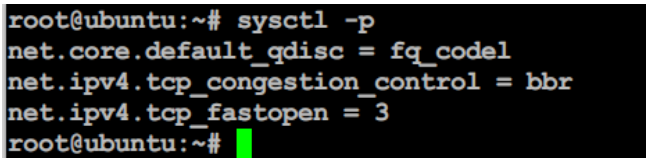
> This section is Tiancheng Zhu's (894481) personal behavior. Other group members think this specific detail is out of the scope of the project. Tiancheng, himself, will be responsible for any of consequences caused by this section.

Bottleneck Bandwidth and Round-trip propagation time (BBR), provides higher throughput and lower latency as network interface controllers evolves to Gbps.

Fq_codel is also set (or overridden to the system default) to reduce bufferbloat. It did not implement TCP pacing until Linux 4.13 (https://github.com/systemd/systemd/issues/5090), which limited BBR functionality.

In `/etc/sysctl.conf`, `net.ipv4.tcp_fastopen = 3` to enable TCP Fast Open (TFO) for both server and client connections.





Regarding the three features mentioned above, with a kernel version of 4.15 on Ubuntu 18.04, the best effort of TCP transmission will be promised. To validate the benefits of the optimizations, by conducting three times each, both directions of iperf3 tests between optimized and non-optimized Nectar(Ubuntu 18.04, m1.medium) environments shows that TCP contention window retransmissions have been significantly reduced when the optimized instance sends packets to the other, compared to that in a reverse direction (with `-R` parameter).

Test	Optimized	Retransmissions	Bandwidth (From Receiver)	Contention Window at 8s-9s
1	No	7063	6.78 Gbps	1.30 MB
2	No	21921	7.31 Gbps	1.35 MB
3	No	8449	8.28 Gbps	1.27 MB
4	Yes	2	6.99 Gbps	532 KB
5	Yes	0	6.30 Gbps	532 KB
6	Yes	1	7.46 Gbps	512 KB

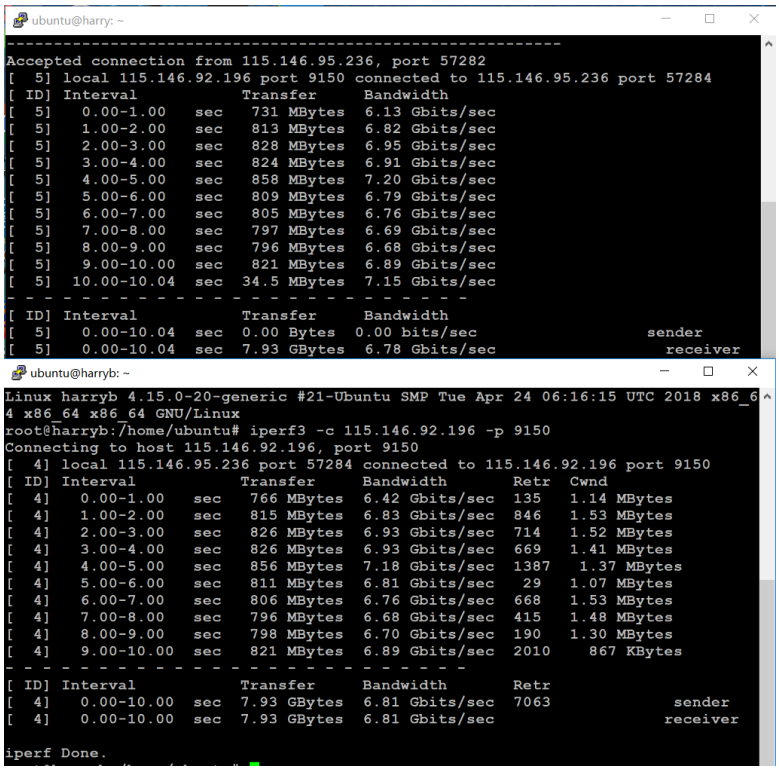


Figure 16: Test 1

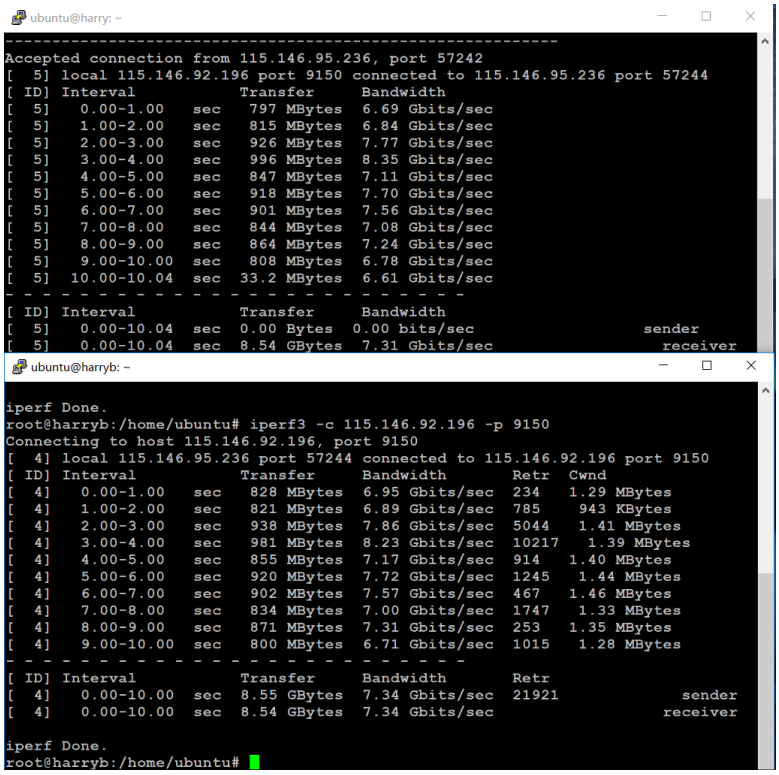


Figure 17: Test 2

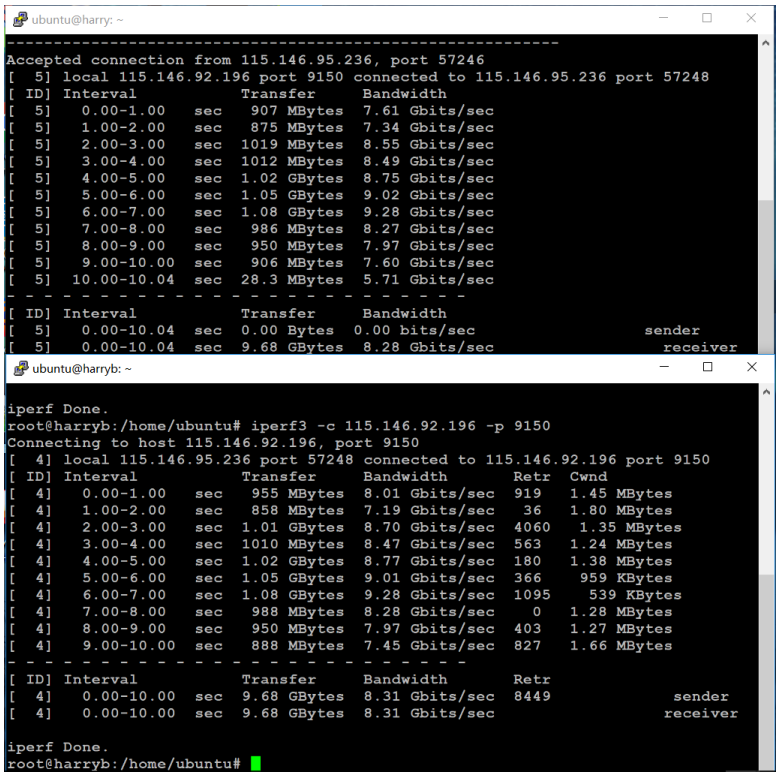


Figure 18: Test 3

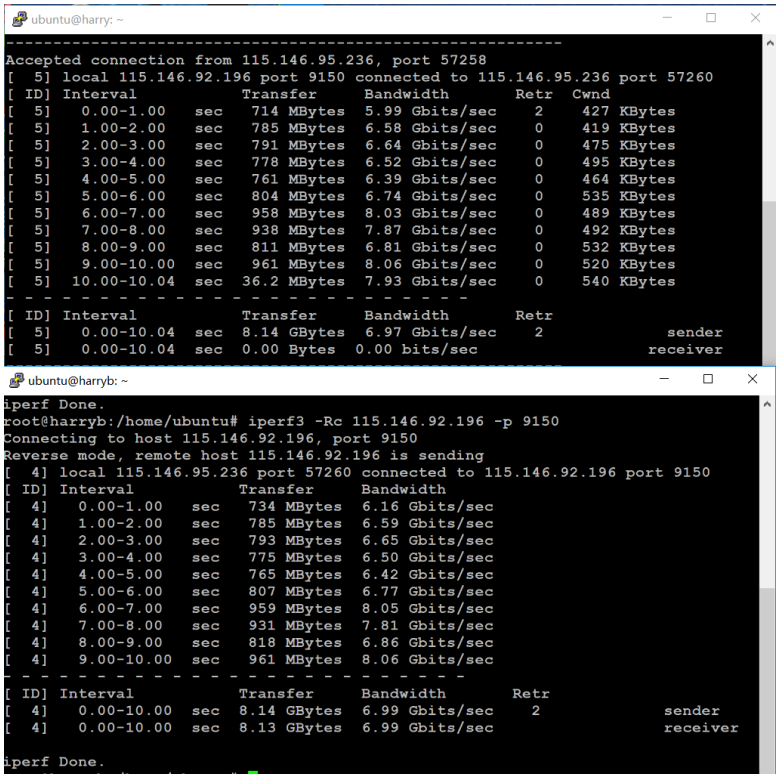


Figure 19: Test 4

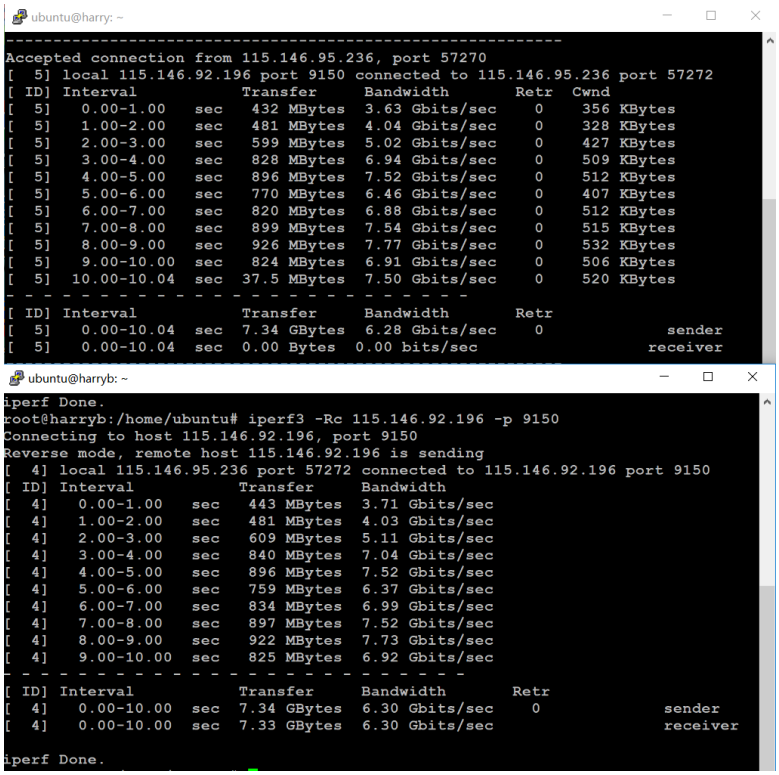


Figure 20: Test 5

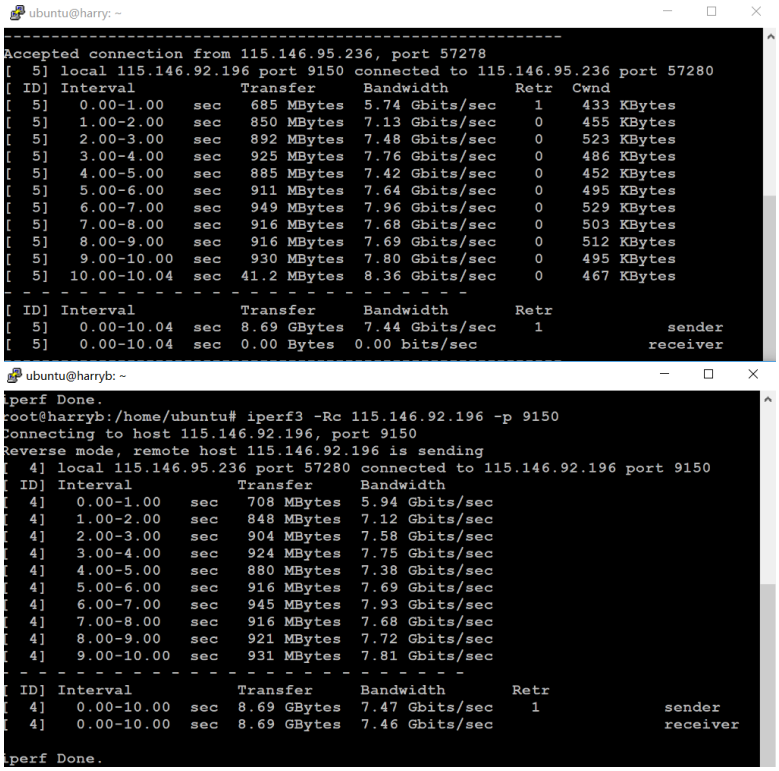


Figure 21: Test 6

References

[1] A. J. Morales, V. Vavilala, R. M. Benito, and Y. Bar-Yam, "Global patterns of synchronization in human communications," *Journal of The Royal Society Interface*, vol. 14, no. 128, p. 20161048, 2017.

[2] M. J. Koohsari, T. Sugiyama, S. Mavoa, K. Villanueva, H. Badland, B. Giles-Corti, and N. Owen, "Street network measures and adults' walking for transport: Application of space syntax," *Health & place*, vol. 38, pp. 89–95, 2016.

[3] R. G. Prins, A. Oenema, K. van der Horst, and J. Brug, "Objective and perceived availability of physical activity opportunities: differences in associations with physical activity behavior among urban adolescents," *International Journal of Behavioral Nutrition and Physical Activity*, vol. 6, no. 1, p. 70, 2009.

[4] B. Y.-M. Wong, S.-Y. Ho, W.-S. Lo, E. Cerin, K.-K. Mak, and T.-H. Lam, "Longitudinal relations of perceived availability of neighborhood sport facilities with physical activity in adolescents: an analysis of potential moderators," *Journal of Physical Activity and Health*, vol. 11, no. 3, pp. 581–587, 2014.