Cluster and Cloud Computing Assignment 2
Distributed Computing System for Tweets Analysis

Group 7

Hanxun Huang - 975781
Haonan Chen - 930614
Lihuan Zhang - 945003
Xu Wang - 979895
Yang Xu - 961717

Lecturer: Prof Richard Sinnott

School of Computing and Information Systems
University of Melbourne
Australia
May 2019

Contents

1 Acknowledgments

Thanks for every team member worked on this project and NeCTAR Cloud for providing the computational resources. We are grateful to participate in this interesting project. For a detailed Userguide, please refer to README.md files in each folders. Our GitHub private repository link: `https://github.com/HanxunHuangLemonBear/COMP90024-2019S1-Team7`

2 Introduction

TrackHub is a real-time platform that could track the twitter user and their post content. Our system analyzes the text and image posted by the user. The results will be grouped by geolocation bounding box for further analysis. The frontend will also able to load AURIN data to build the data visualizations. Currently, our system focuses on three of the seven deadly sins, the Lust, the Wrath, and the Gluttony.

3 Project Background

- Build a cloud based scalable system that can track Twitter user movement. The software system should be easy be maintain and fault-tolerant.
- Ability to analysis the text and image content of the Tweets based on the Seven Deadly Sins. The data analysis section should also be easy to scale or add additional models to analyze the different content.

4 System Implementation

4.1 Automation Deployment

The technique we use for web servers' creation and application continuous delivery is Ansible[1] and Docker[2]. By using OpenStack SDK with Ansible, we can create instances on NeCTAR[3] with preferred flavor and customized configurations. The procedure for deploying applications are threefold:

1. Create instances of 2CPU 9G RAM flavor with Ubuntu 16.04 LTS image, see video: `https://youtu.be/2shxCohju8Q`.
2. Add proxy settings to server environment and docker environment for network penetration on private servers; Install dependencies including vim, git, curl, docker, docker compose etc.; Configure git in order to ssh GitHub through public/private key pair; Create file systems and mount volumes for mounted disks, see video: `https://youtu.be/JHfxmtzU_g4`.

[1] https://www.ansible.com
[2] https://www.docker.com
[3] https://nectar.org.au/

3. Pull the latest code from our private group GitHub and deploy applications for each servers, including CouchDB cluster, Django backend service, web crawler service, natural language processing service, VUE.js frontend and Grafana monitor, which architecture are introduced in the next section. See video: `https://youtu.be/pWH7eOh666s`

Ansible is a lightweight management software that does not need to install clients on managed hosts and operates multiple target hosts at the same time based on SSH. With the help of various internal functional modules, it realizes the functions of batch system configuration, batch program deployment, batch operation commands and so on.

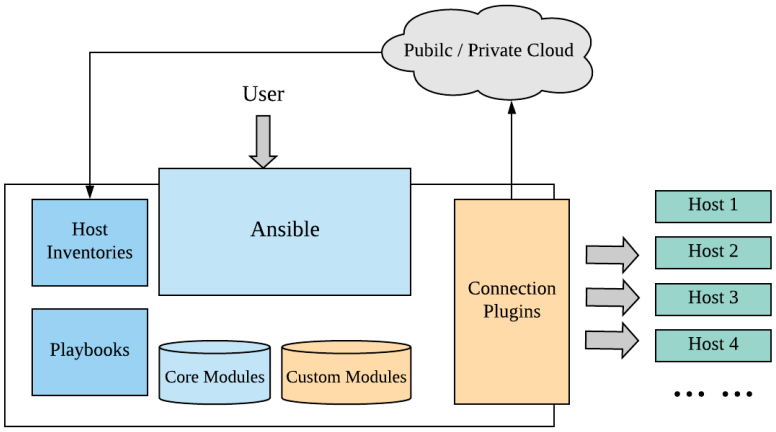

Figure 1: Ansible Introduction

Host Inventory: The list of hosts, that is, the list of managed hosts. Playbooks: An ansible script, imagine putting multiple tasks together and executing them together. Core Modules: The Core Module of Ansible. Custom Modules: Custom Modules. Connection Plugins: Connection plugins used to establish connection relationships with managed hosts based on SSH.

4.2 Cloud System Architecture Design

4.2.1 Microservices Arrangement

Figure 2 generally shows how we allocate the services in order to balance the workload across all instance we have. From our experiments, CouchDB and Machine Learning Services would cost most computation resources, thus we decided to use 3 instances to construct CouchDB cluster, each deployed 80GB volumes, and put machine learning at the 4th instance.

In order to avoid exposing all the instance IP addresses and ports to users, we also construct an Nginx service to reverse proxy all the services, with this

design, user can only access the port 80 of instance 1. The frontend container is located at the same instance with Nginx to lower the latency to load webpages.

The backend service is located at instance 2 as well as crawlers, then the crawler could upload tweets to backend directly instead of through gateway. The other backend was placed at instance 3. Natural Language Process and Machine Learning was separated on different instance because both of them consume many resources. The cloud monitor services were placed at instance 4 because they are light level so would not compete for resources with machine learning. The final workload analysis will be presented later.

The first step of our development is to design an appropriate cloud architecture to achieve the purposes. Considering the limited resources (8 cores, 36 GB RAM) arranged to us, the basic system architecture must be designed carefully, or it would be a mess. According to the functions we want to provide, the ideas of microservices are suitable for this project, we should split the function into the smaller granularity, and distribute running them on different instances.

After abstracting the specifications, we found that the necessary components should including the following parts: database, web backend, web frontend, machine learning model, natural language process model, and twitter crawler. Because of the machine learning component is base on computer vision and pattern recognization, a picture storage service was also required. What's more, referent to the other cloud company, it might be better to construct a monitor service to display the latency of service and the CPU and RAM usage rate of instances.

4.2.2 Database: CouchDB Cluster

Firstly, we considered the choice of the most appropriate database, there are several options: MongoDB, CouchDB, and HBase. MangoDB software routers can only embedded in application servers, while any HTTP client can access to CouchDB. CouchDB should be much easier to implement in this project since we may have thousands of RESTful requests to CouchDB per hour. HBase is the best choice when using Hadoop framework for map/reduce tasks. HBase provides high reliability, high performance, column storage, and supports massive real-time read and write operations. However, The inherent limitations of RowKey determines that it is impossible to effectively support multi-condition queries. It is not suitable for large-scale scan queries, neither. Our application includes functionalities, such as summary the tweet data in a particular region, that requires multi-condition queries on the tweet data in the database. Thus, HBase is not an ideal choice for our application. Finally, we decided to choose CouchDB because it would be easier to build a cluster and to provide high availability to other services (especially when the stability of nectar was unknown at the beginning). According to our experience, both the database and machine learning model would consume much computational resource, so we choose to construct a CouchDB cluster on 3 of our instances.

4.2.3 Frontend Overview and Functions

Design Overview
Front-End consists of 2 web pages with one for project and team member overview and one for data visualization. The data visualization page has three main parts which are 1 full screen google map view for presenting the statistics of tweets that contains geo/coordinate information and 1 bar chart below to summarize the statistics and 4 pie chart at the bottom for showing the total result from machine learning and text analyzing. The map view is also used to display Aurin data across regions.

Implementation Overview

1. **Vue:**
   We use Vue.js for as our frontend development which is one of the most popular framework for web development and it provides a very easy way to build powerful single page applications that is suitable for our project. One significant advantage of using Vue is that we it allows us to bind data from backend easily into various interactive components or even nested components for a rich presentation of data visualization. Most of our data visualization functions are implemented in a component named as 'Map.vue'. Moreover, Vue has the advantage of multi platform by using Vue native which can transfer our project into a native mobile application easily based on the current implementation.

2. **Google Map JavaScript API:**
   Our user interface uses the google map API to develop our map component which is considered to be the one of the best libraries for map based web application. Typically, we mainly uses google map's data layer to load the Geojson file come with data fetched from the backend and display the data with Vue Chart in google map's infowindow. We also maximize the use of google map's markers and ploylinepaths to display the tracking information of twitter users in a colorful and detailed way.

   However, there is a limitation while using google map API that is the performance limited by browser's caching ability. Considering the fact that the size of VIC Geojson could be more than 15MB which is quite heavy weight for frontend to load it several times with different datasets. The same performance issue also happens while we are trying to tracking more than 20 users simultaneously on the map which may lead the frontend to crash.

3. **Axios:**
   Axios is the proposed API consumer for Vue application which is used to fetch data from our backend through RESTful APIs. The API calls from frontend are consistent with the specification from the backend service.

   In order to make a good user experience, we handle errors carefully while communicating with the backend. For example, the frontend application

will only allow users to use tracking button when a twitter user's id is provided or number of tracking is provided.

Moreover, in order to prevent unauthorized access to the system we use API key generated from the backend for access control which means most HTTP requests to the backend must includes a API key in the header for authentication.

4. **Vue Chart:**
   This is the library we used for data visualization which provides our user interface the ability to show statistics in various kinds of interactive charts such as pie chart and bar chart. It is also embed into google maps's infowindow where we used it to display pie charts for showing the statistics for certain area on the map.

General Search

The general search function allows users to show the overall statistics (that has geo information) from machine learning and text analyzing and then displays the result in terms of regions as below which is an example in the scope of Melbourne. The other optional scope is for the statistics from the whole VIC. The detail of each region contained in the scope could also be shown by clicking that region and the result will be displayed in a infowindow.

General Statistic

It displays the overall statistics (with or without geo) from machine learning and text analyzing.

Aurin Statistic

It displays the aurin statistic of each region contained in the scope of whole VIC to compare with the result from General Statistic function above. It supports the following scenarios:

1. The unemployment people.
2. The total number of hospital admissions.
3. The total number of male hospital admissions.
4. The total number of female hospital admissions.
5. Weapon related offense.
6. Assault-related offense.
7. Sexually related offense.
8. Robbery related offense.
9. Harassment and Threatening.
10. Total offense.

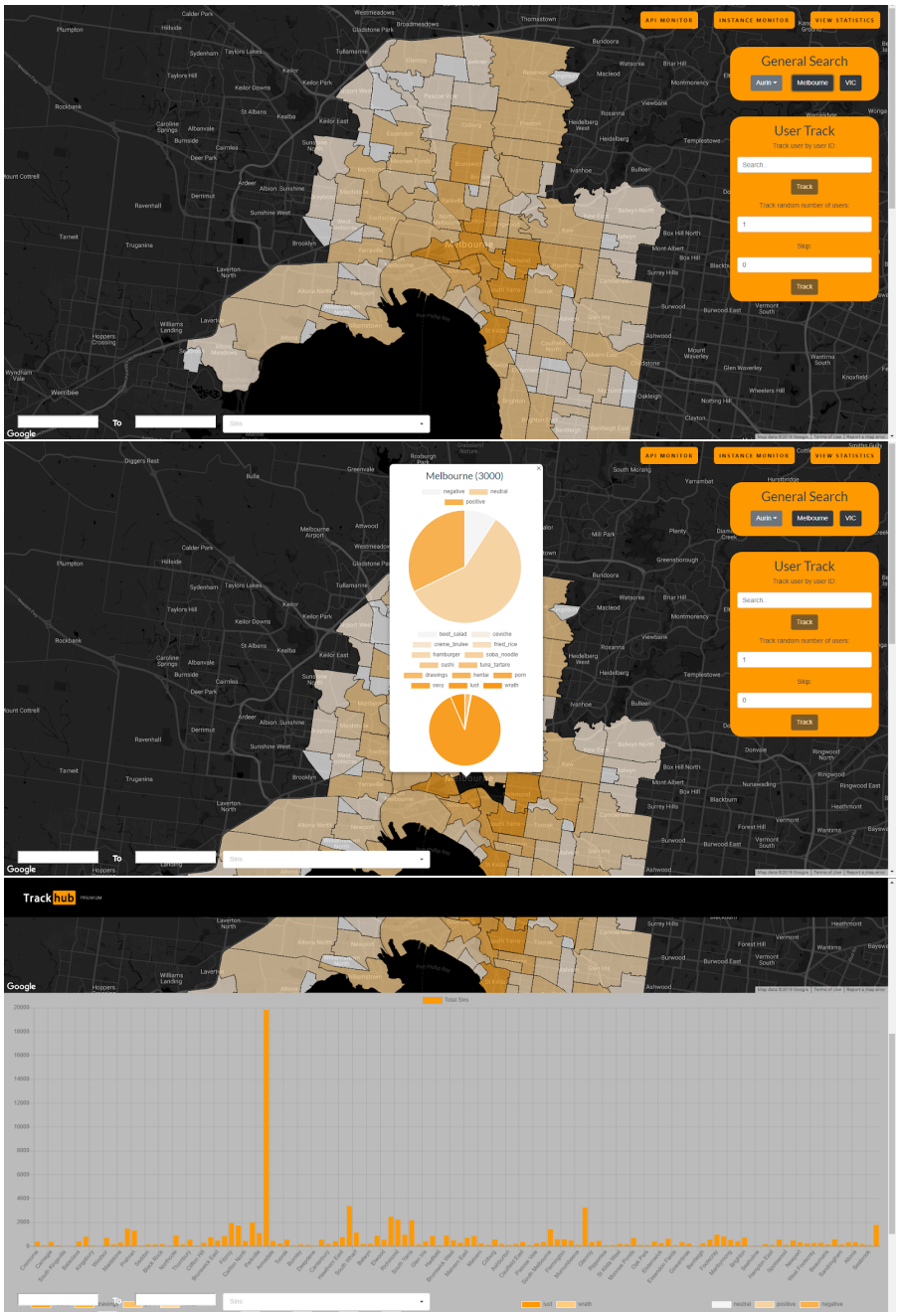

Figure 2: General Search

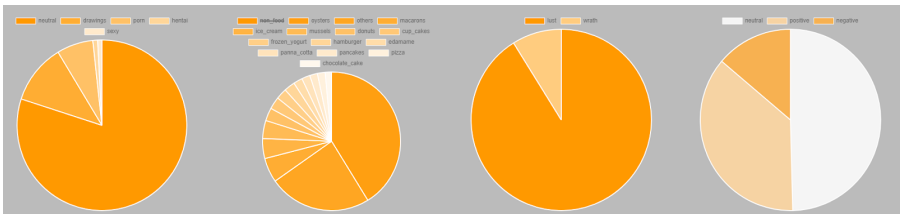

Figure 3: General Statistic

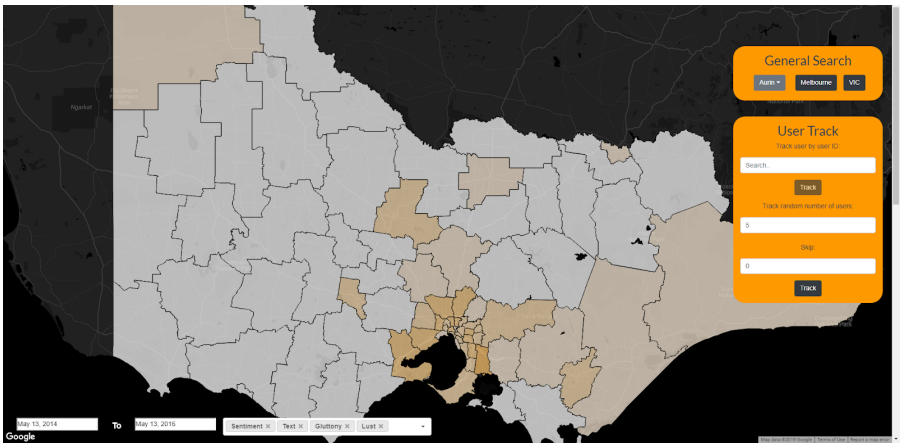

Figure 4: Aurin Statistic

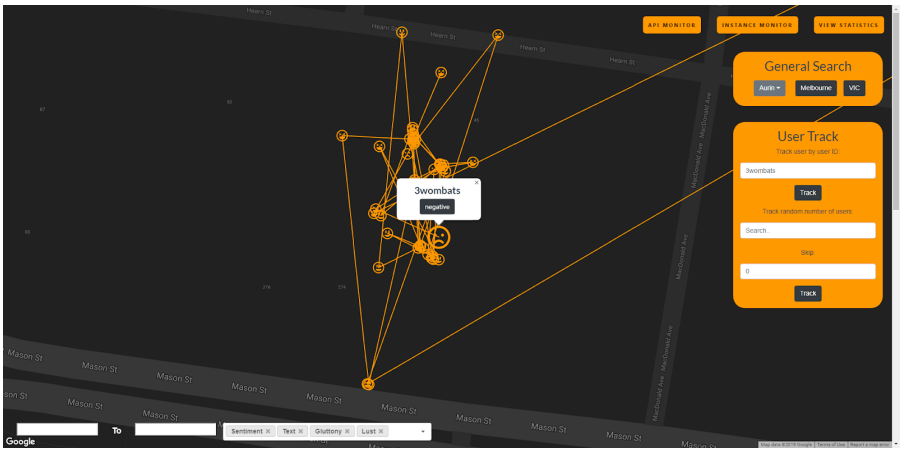

Figure 5: Track 1 user

Track 1 user by ID for certain scope within certain time period
By using this function, the path for each user will be shown on the map as above and alone with users' sentiment being marked on the map as a happy face for positive emotions, smiling face for neutral and sad face for negative emotions, see Figure 5 . Moreover, the time, text, tags and image each mark within the path could be shown in the following way:

1. Time and Text will be shown when mouseover the mark.
2. Tags and Image(link button) will be shown when mouse click the mark.

Track random n number of users within a certain time period
This function can track random number of users on the map similar as the one above but it will not include text information. Each users' path is assigned to a random color and it could be highlighted by the mouseover event, see Figure 6.

Interesting Findings
By examining our tracking function we find that Twitter users who are identified with lust tags are more likely to have either negative or positive sentiments which means those who are identified as lust tend to have certain emotional bias from neutral.

4.2.4 Backend: Django

Spring Boot, Flask or Django are all could be used for this project, but due to the limited development time, Django was the most appropriate one because it is easier to build and debug. The abundant Python package would greatly speed up the development. What's more, Python naturally supports dictionary data

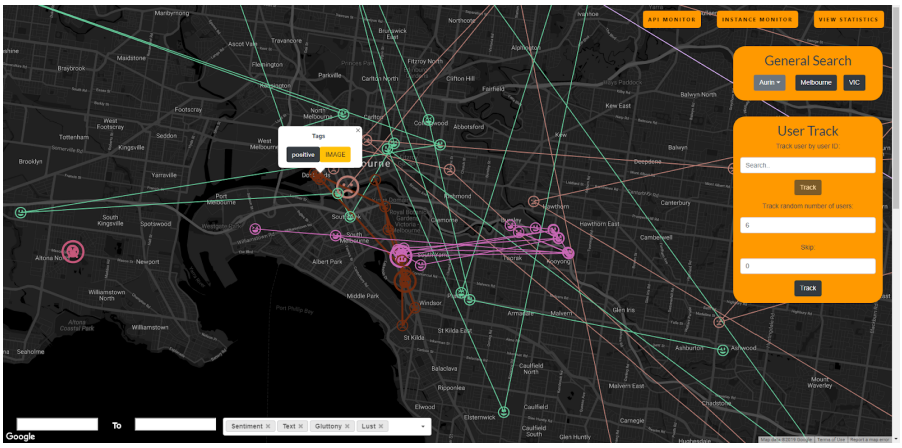

Figure 6: Track N users

type, it was more convenient to operate JSON documents than Java. The concurrent performance of Django is worse than Sprint Boot, however, is definitely enough for this project.

4.2.5 The Implementation of Tweet Crawlers

Overall Architecture

In our application, we have adopted several versions of tweet crawlers to satisfy our different demand of tweet data. Since the overhead of our instance 2 is the lowest among all four instances, we deploy all three crawlers on instance 2. Two of three tweet crawlers retrieve historical tweet data, and the other one gets the real-time tweet stream. After all historical tweet data, we need (tweets from 2014 to 2018, in Melbourne and Sydney) have been scraped, we stop two historical tweet crawlers (getByUser.py, getResearchData.py). The tweet stream crawler (tweetStreamer.py) will keep running as long as our application is not terminated. The tweet crawlers utilize twitter APIs and Restful API, and we also use MPI to speed up the data processing. The tweet data scraped by crawlers will be uploaded to the backend through Restful "post" call and stored in CouchDB.

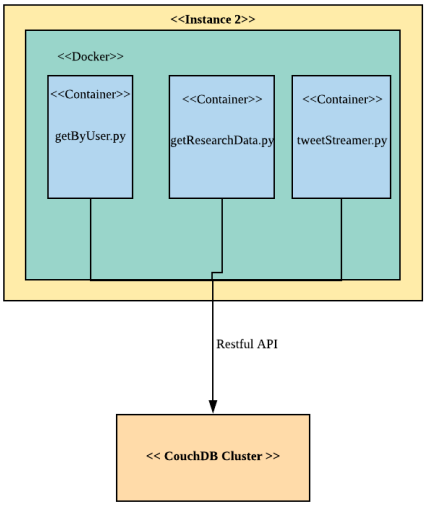

Figure 7: Tweet Crawlers Architecture

Design of Tweet Crawlers

All tweet crawlers communicate with the backend through Restful API. For each tweet, the crawler will firstly check whether there is any image in the tweet. If there is an image (e.g.: an image URL contained in tweet data), the crawler will retrieve it and upload the image to the backend, and then deal with the tweet text content. The images retrieved from tweet don't include profile image. Before uploaded, all images are reformatted to a size 256*256. All images will be used in the testing set of Image Recognition task. The text-based content of twitter is converted into the following format and uploaded to backend by crawlers:

```
{"id": <str>,
"user": <str>,
"text": <str>,
"date": <str>,
 "hashtags": [<str>],
 "geo": [<str>],
 "img_id": [<str>]}
```

Figure 8: Tweet Crawlers Format

Converting tweet into a specific format allows us to implement sentiment analysis and map/reduce process easily, with maintaining the consistency of data.

All crawlers implement "try" and "except" clauses to deal with error handling. When there is any error in the process of reformatting image, uploading image or uploading tweet data, a specific message will be printed out the wrote into the log file. Using the message, we can easily to understand what kind of error happens and find out where in codes the error happens. Also, since we need to fill out a query to specify what kind of tweets we need to download, all crawler use "argparse" package to parse command line to get the information of query.

The three types of tweet crawlers are:

1. **getByUser.py:**
   This crawler takes advantage of GetOldTweets3 package for python (Henrique 2016). This package allows the program to retrieve historical tweet data by defining the username, geolocation and publishing date of tweets. The functionality of GetOldTweets3 package may seem similar to Twitter Search API. However, Search API can only provide old tweets posted in the last seven days. The GetOldTweets3 package allows the program to retrieve old tweets during any arbitrary period. We use GetOldTweets3 package in getByUser.py to explore the historical tweets from a specific username.

   The disadvantage of this approach is that a lot of tweets scraped by GetOldTweets3 package don't contain geo-coordinates, because the PyQuery method used to scrape geolocation of the tweet doesn't match with the new structure of tweet HTML data. This disadvantage also appears in Twitter Search API. Most tweets acquired by using search API do not have geo-coordinates. Another disadvantage of GetOldTweets3 package is that, over a period of time, the tweet obtained through this approach is fixed. Therefore, if there is some connection problem on the network and we restart this crawler, then there are many duplicate tweets scraped because this API has to scrape tweets from the start date defined in the query. We solve this problem in our application. The duplicate tweets uploaded are handled by the backend to remove the redundancy.

   Moreover, in this crawler, we can use MPI to parallels the process of uploading tweet data. The tweets are scraped in specific batch size. For each batch of tweets, we use MPI to speed up the processing (convert format, upload...) of tweet data.

2. **getResearchData.py:**
   This crawler scrapes tweets data from Unimelb Research Cloud by using the provided API. The format of tweet data on Unimelb Research Cloud is slightly different from the real-time tweet data obtained using streaming API. This difference is handled in the pre-processing stage in this crawler. Then, the tweet data is uploaded to the backend using restful

API. The advantage of tweet data from Unimelb Research Cloud is that most tweets have geolocation information, which is useful to the representation of frontend and tasks of data analytics since we need to use a map to show statistics for each Local Government Area (LGA). When we scrape tweet data from the cloud, we specify the city name and time in the query.

3. **tweetStreamer.py**
   This crawler utilizes Tweepy package to download real-time tweet data. The Tweepy package provides functionalities similar to Twitter's streaming API. Moreover, Tweepy provides an encapsulated class named StreamListener to download twitter message in real time. Using Tweepy packages reduces the overhead of operations in scraping real-time tweets using RESTful Post requests. To add more functionalities, such as uploading real-time tweet to the backend, we define a class TweetListener, which inherits from StreamListener and overrides the on_data() function. on_data() function is used to process the real-time tweet data, convert tweet data into the standard format and upload the reformatted tweet data to backend. The on_error() function prints out the system message, which helps us to track the error when there is any.

   Besides, StreamListener supports downloading tweets in accordance to regional information, so we define geo bounding box ("[141, -38, 150, -34]") to retrieve all real-time tweet data in this area. This geo bounding box has contained Victoria State and most parts of New South Wales State. We assume that there are more tweet users and real-time tweets in large cities such as Melbourne, Sydney, and Canberra. This geo bounding box can be changed using the command line argument.

   This crawler keeps running when there is no error in the application. When there are some errors, such as server problems, the crawler will be paused for 30 seconds automatically and then try to restart again. This approach of error handling can avoid the shutdown of stream tweet crawler due to the problem of our application.

4.2.6 Image Recognition Machine Learning Model

The image recognition will focus on two of the seven deadly sins, Lust and Gluttony. For Lust, we classified the image into five categories, Drawing, Natural, Sexy, Hentai, and Porn (NSFW Dataset). For Gluttony, we analyzed the image content and classify into a specific dish name, a total of 178 categories and a non-food category (Food179 Dataset). The ResNet and MobileNet which are both states of arts Convolutional Neural Networks in Computer Vision field. The standard RestNet is particular good for the task that need accuracy. The MobileNet on the other hand, has better inference time while maintain acceptable accuracy.

Dataset

For NSFW (Not Safe For Work) dataset, we used an image crawler that crawling

the Reddit base on the class label of the image. The first version of the NSFW dataset consisted of all the unmodified imaged crawled image, however, after training, we discovered that the outcome is not perfect, there is a lot of false positive prediction when we deploy the model in production. We suspect the dataset is noisy and contains misclassified images, subsequently, we manually examined the dataset and created the second version of the NSFW dataset. We also added a test set of ILSVRC2012 created by Russakovsky et al. (2015) into natural class to reduce the false positive rate.

For Food179, we manually combined Food Images (Food-101) from Kaggle and UEC FOOD 256 (Kawano and Yanai 2014) into a single dataset as our first version. We also randomly selected some natural class image from our NSFW into the non-food category as version 3. Since the NSFW image dataset is crawled from the Reddit, it is more suitable to analysis the Tweet images because they have a similar user base, the social network users. Version 3 should dramatically reduce false positive predictions.

Neural Network Architecture

The Resnet Architecture invented by He et al. (2016) is one of the states of the art in computer vision field and has shown top-level performance in various benchmark competitions. We adopted the resnet and as for both datasets classification machine learning model. We also used the He initialization method(He et al. 2015), which is also a state of art initialization method especially useful for Relu activations functions which are used in resnet. Since the Twitter dataset is large that could contain a large number of images, we also used MobileNet (Howard et al. 2017) to improve the efficiency in analyzing the images.

Architecture	Average Inference Time (on CPU)
MobileNet	0.217s
ResNet50	0.358s
ResNet101	0.535s

Table 1: Inference Time Comparison

Training

We used the Stochastic Gradient Descent (SGD) with scheduled learning rate decay instead of Adam optimizer since Adam cannot guarantee the find the optimal solutions. We also experimented the AdaBound that have the training speed comparable with Adam and also guarantees the global optimal (Luo et al. 2019). We used the hold-out to split the dataset into a training set and evaluation set, evaluation contains 20% of the entire dataset.

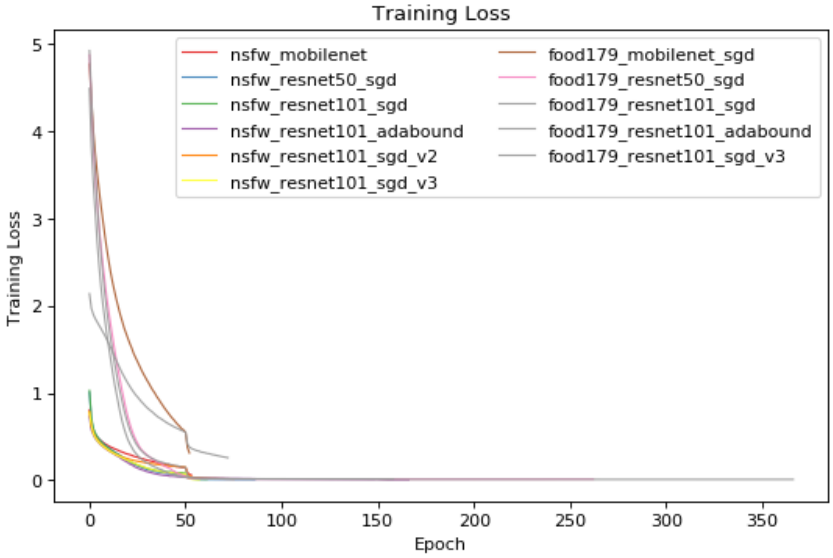

Figure 9: Training Epoch vs Training loss

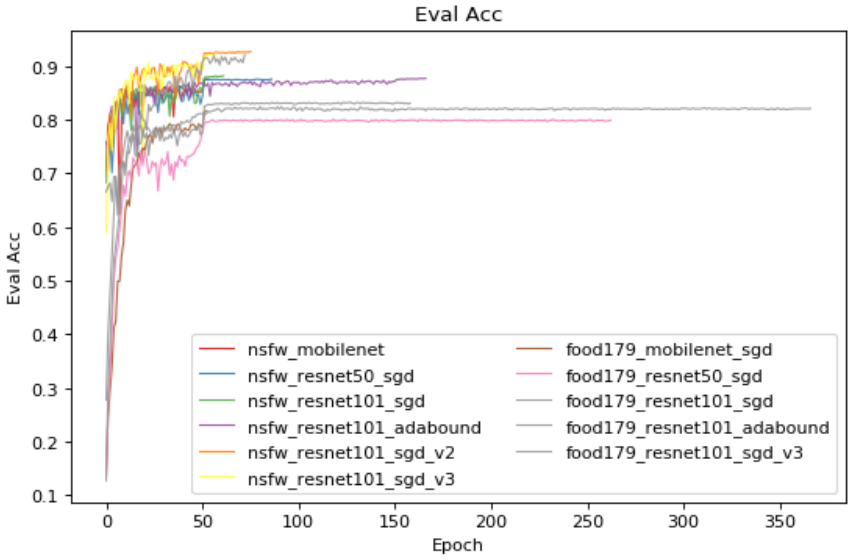

Figure 10: Training Epoch vs Evaluation Accuracy

Deployment

Before deployment, we gathered a 2 set of actual tweets from our crawler program. The first set is consistent with normal image tweet and the second set

Architecture	Optimizer	Dataset Version	Eval Accuracy
MobileNet	SGD	NSFW-v1	0.8760
ResNet50	SGD	NSFW-v1	0.8759
ResNet101	SGD	NSFW-v1	0.8823
ResNet101	AdaBound	NSFW-v1	0.8774
ResNet101	SGD	NSFW-v2	0.9272
ResNet101	SGD	NSFW-v3	0.9227

Table 2: NSFW Model Evaluation Accuracy Comparison

Architecture	Optimizer	Dataset Version	Eval Accuracy (Top5)
MobileNet	SGD	Food179-v1	0.8611
ResNet50	SGD	Food179-v1	0.7994
ResNet101	SGD	Food179-v1	0.8311
ResNet101	AdaBound	Food179-v1	0.8210
ResNet101	SGD	Food179-v3	0.9210

Table 3: Food179 Model Evaluation Accuracy Comparison

is consistent with the NSFW images. Our initial model does not perform well in these actual tweets, the main reason is the false positive predictions. As discussed in the dataset sections, we have developed several different version of the dataset. As the table is shown below, the model trained on version 3 is the better-performed model in terms of balancing the false positive and true positive predictions. Therefore, we used both version 3 for both of our image recognition tasks.

Model Type	Normal Tweets	NSFW Tweets
nsfw-resnet101-sgd-v1	0.1944	0.7458
nsfw-resnet101-sgd-v2	0.0724	0.6392
nsfw-resnet101-sgd-v3	0.0631	0.8523

Table 4: NSFW Class Prediction Ratio on Tweets data

We used PyTorch frameworks for our image recognition system, it is known for its more suitable for research instead of the production environment. Therefore, we need to develop our own inference program for deployment. We added additional fields that needed for inference when we save our neural network model checkpoints. Such as index to class label map and image normalizing mean and standard deviations. In this way, we do not need to have an original training environment except the model architecture implementations, we could simply replace the model checkpoint easily to update the system.

Our custom inference program is connected with our backend system. It will regularly check with backend to get a batch of images that have not been classified yet and upload the result back to the backend system. The backend will subsequently update the database. The batch size and time interval are

customizable by a config file can by adjust base on the load of the entire system. We will also upload the version of the model into the result dictionary to the database so that we could easily re-run on the data that have older version inference result. The inference program also implemented with MPI capabilities since the PyTorch does not automatically utilize multiple CPU cores. We adopted the Single-Program Multiple-Data architecture, the root process will handle the data distribution, gathering, and communications with our backend system. According to Amdahl's law (Hill and Marty 2008), the theoretical maximum speedup is 1 divided by the number of cores. Our MPI version inference is limited by the communication overhead with the backend system and the distribution as well as the gathering of the data between each process.

We implemented our own image recognition system and it connected with our backend to retrieve images that need to be classified. We used Docker to deploy such system. Our intention is to make it more robust , easy to maintain and scalable. Docker is most suitable choice. Dockerfile was created for the image recognition system that mounted the config file and model checkpoints on the host computer filesystem. The Dockerfile specifies the packages from PIP that are needed for the system as well as how to run the script. The we can replace the model or update the checkpoint without rebuild docker images or restart the services, it can be easily upgraded since it is mounted on the host filesystem. It is also easy to adjust the work load though the config files.

4.2.7 Text Analysis

In the Text Analysis process, we use RESTful GET requests to retrieve tweet data from the backend. The interface where we retrieve tweet data is different from where we upload the raw tweet data. After implementing text analysis, we generate three tags ("lust", "wrath", "sentiment") for each tweet text and upload the tag results to the backend. For the tag results that are sent to the backend successfully, the backend will mark the corresponding tweets as "processed". For tweets analyzed whose tags fail to be uploaded to the backend due to some errors, the tweets are still marked as "unprocessed". The tweets retrieved from the backend in Text Analysis process are only "unprocessed" ones.

Text pre-processing

In this procedure, we remove all meaningless information from original tweet text. We use Regular Expression to eliminate symbols such as hashtag (#) and "at" (@) to keep only English text. We use

```
text = re.sub( '([#])|([^a-zA-Z])', ' ', text )
```

to remove all symbols. This line of code is powerful because it doesn't only remove non-English character in a sentence, but also those in the content of hashtags. An example of text pre-processing:

1. **Original text:** Prof. Li talks about the exam today #ohmygod...
2. **Pre-processed text:** Prof. Li talks about the exam today ohmygod

The disadvantage is that, if a hashtag contains non-English characters in the middle of it, such as "#pray4me", the hashtag will split into multiple words after pre-processing ("#pray4me" "pray" "me"). However, the number of such hashtags are small, considering we have collected over Two Million tweets in the database. Moreover, when implementing text analysis, our approach will focus on the meaning of tokens instead of the combination of them. After removing all non-English characters, the text will be split into a list of tokens, which are analyzed by text analysis algorithms.

Text Analysis Approach

1. **Check "Lust" Meaning (WordNet + Wu-Palmer Similarity):** WordNet is a large lexical database of English. In this database, Nouns, verbs, adjectives, and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Synsets are interlinked through conceptual-semantic and lexical relations(Miller 1995). WordNet groups words together based on the meaning of words. The interlinks between words are not only determined by the string of letters, but also the actual meaning of words. In this way, words in WordNet are semantically disambiguated. The meaning of the word will determine the proximity of words in WordNet.

   Wu-Palmer Similarity is used to measure the semantic similarity between words with the same "part of speech"(Wu and Palmer 1994). Each word can have multiple "part of speech". For example, "study" can be used as a verb or a noun. When using Wu-Palmer Similarity, we only need to consider a specific "part of speech" of the word. We define a conceptual domain for the specific "part of speech" of the word. When computing the similarity of two words, we put them into the same conceptual domain, which requires they have the same "part of speech". For instance, we don't compute the similarity between "excellence" (noun) and "wonderful" (adjective), even if we know they are similar in the meaning. The conceptual domain is a hierarchical structure for words. Each node on this hierarchical structure is a word with a unique meaning. To compute the similarity between two words, we need to compute the Conceptual Similarity between the corresponding nodes in the conceptual domain (ibid.).

   Wu-Palmer Similarity and WordNet are applied to identify the words with "lust" meaning in the tweet text. We define three lists of keywords with "lust" meaning. These lists contain none, verb and adjective. If the token in tweet text is similar to any of these words, we conclude that the corresponding tweet text contains "lust" message. For each token in the list, we firstly use lemmatizer to remove inflectional endings only and to return the base or dictionary form of a word. Then we find the cognitive synonyms (synsets) of the word. For each word in synsets, we compute the Wu-Palmer Similarity between the word and our keywords. Finally, we only need the maximum value of Wu-Palmer Similarity for all three kinds of "part of speech" (noun, verb, adjective). Since the synsets of each token contain a lot of words, we set the threshold to 0.8. The upper

bound Wu-Palmer Similarity is 1.0, for the same words. This threshold is chosen heuristically, and we find it to work well to identify the "lust" tweet text.

2. **Check "Wrath" Meaning (Profanity Package):**
   Python Profanity Package is used to identify whether the tokens of tweet text contain some profanity words[4]. The word list defined in this package are words that indicate "violence" meaning, which corresponds to "Wrath" in Seven Sins.

3. **Sentiment Analysis (TextBlob Package):**
   We implement sentiment analysis using the features of the TextBlob package(Loria 2012). The sentiment analysis in TextBlob package is based on Naïve Bayes Classifier(S.-C. Huang et al. 2011). By computing the conditional probabilities of each token in the text, the analysis will produce a polarity score, which indicates the sentiment of the text. If the polarity score is larger than 0, then the sentiment of the text is "positive". If the polarity score is less than 0, then the sentiment of the text is "negative". Value of "0" indicates the tweet text is "neutral".

4. **Parallelization of process via Spark:**
   Due to the overhead of text analysis and RESTful requests to the backend, the speed of text analysis for a single tweet is slow. When using a single thread to deal with the text analysis, it takes three seconds to process one tweet. To speed up the text analysis process for over Two Million tweets, we need to parallelize the process. In this situation, we use Spark as the framework of parallelization. We choose to run Spark locally using PySpark package instead of constructing a Spark Cluster.

   There are two reasons for this choice: Firstly, the overhead of communication between Master node and Slave node may worse the performance of Spark. Also, you have to deal with the data storage when considering a cluster or distributed system. Secondly, although we have over Two Million tweets, the size of data is relatively small (around 1.5 GB). These data can fit in the memory of the local machine. However, for a larger size of data, such as 100 GB, which cannot fit in the local memory, the Spark Cluster must be constructed to distribute the data into multiple nodes. Therefore, we run Spark locally with four work threads, since the instance we use only has two cores.

   Resilient Distributed Datasets (RDDs) are the way data are stored in Spark during computation. We retrieve tweet text and corresponding tweet id, then create a list of (text, id) pairs. Then, we create RDDs using the list of (text, id) pairs (via parallelize() method) and map each pair to "updateTweet()" function via "map()" method. In this function, we implement text analysis and finish the upload of tags to the backend. A message pair (flag, id) is returned by "updateTweet()" function. We

[4]https://pypi.python.org/pypi/profanity

use "collect()" method on RDD to collect all returned message pairs. The error handling process will check the message pair, if there is a "0" flag, it means some error happens in the process of uploading tag results. In this situation, the process in Spark will be stopped for 30 seconds, then the program will try to rerun text analysis again. Where there is no tweet in the database that is not processed, the program will be paused for 1800 seconds, waiting for new real-time tweets to be uploaded to the database.

The performance of parallelization via Spark is excellent in practice. Before using Spark, it takes three seconds to finish the whole text analysis process of a single tweet. After parallelizing the process with Spark, the average volume of tweets processed by the program is 45000 per hour. Therefore, the program using Spark run 37.5 times as fast as the program without any parallelization! Obviously, Apache Spark is suitable for tasks of Big Data Analytics.

5. **Suggested Improvements:**
   First, the sentiment analysis applies TextBlob package, whose sentiment features are based on Naïve Bayes Classifier. The recall of Naïve Bayes Classifier on sentiment analysis may not be high enough. Also, we implement a sentiment analysis on the tokens of tweet text. We identify the sentiment of tweet text based on the score of the single token. This approach ignores the context of the text, which may affect the meaning of the token. For example, this approach may not be able to capture the sentiment of some sarcasm. For this situation, using RNN with LSTM cells to implement sentiment analysis seems a better approach, because it can take into account the context of the sentence. Firstly, we can use thousands of tweet texts to construct a dictionary mapping word to integers (starting from 1). Then, we can construct an embedding layer to map each word to a vector. The outputs of the embedding layer will be the inputs of LSTM cells. By training this RNN with thousands of tweet texts, we can construct a neural network for sentiment analysis. The input of RNN is a list of tweet text. RNN learns the position of each word and their neighbors. Therefore, the context of the text is learned by RNN and the recall of sentiment analysis should be higher.

   Second, in our current implementation, we use RESTful GET requests to retrieve unprocessed data from the backend, which are not efficient. A more delicate way is to construct connections between Spark and CouchDB. We can use the official Couchbase Spark Connector, where we use our CouchDB as a data source to implement Spark tasks. More info on `https://github.com/couchbase/couchbase-spark-connector`

4.2.8 Picture Storage Sevice: NeCTAR Object Storage

There are two possible options for us to store the pictures downloaded from twitter:

1. Use Nectar Object Storage Service. We can utilize OpenStack Object Storage Service Swift to realize it.

2. Build a Hadoop cluster and use HDFS to manage the images.

We make several comparisons between Swift and HDFS:

1. HDFS uses a central system to maintain file metadata, while in Swift, metadata is distributed and replicated across clusters. Using a central metadata system is no different from a single point of failure for HDFS, making it more difficult to scale to very large environments.

2. Swift was designed with a multi-tenant architecture in mind, and HDFS does not have the concept of a multi-tenant architecture. Our application should be designed as scalable, delivering the service for a large number of users. Multi-tenant architecture allows a single instance of the software and its supporting infrastructure to serve multiple users. Swift is much better in this scenario of our application.

3. HDFS is optimized for larger files (this is what happens when working with data), and Swift is designed to store files of any size. We use the storage service to store pictures, whose sizes are not fixed. Swift allows us to store different sizes of pictures.

4. In HDFS, files are written once and only one file can be written at a time. In Swift, files can be written multiple times; in concurrent operating environments, the service subjects to the last operation. In our application, there are multiple crawlers harvesting tweet data at the same time. When crawlers upload tweet data to the backend, there may be concurrent read and write. Swift can handle the issue of concurrency automatically.

In conclusion, Object Storage Service is more suitable for our application. So, we choose to use Nectar's service directly instead of constructing our own service. Because all of our API are following Restful rules, and the results are stored as JSON file, the Nectar Object Storage Storage also be used as a cache for our statistics results.

4.2.9 Cloud Monitor Service: Grafana + InfluxDB + cAdvisor

The most popular solution for cloud monitor service is Grafana + InfluxDB + cAdvisor, this fully functional solution is convenient to construct and use, also very easy to be customized, we can use this service to monitor the status of our instances, API latency and counting the number that our APIs were called.

4.3 Development Stages and Schedule

1. Write ansible script to construct the instance: 20th April - 27th April

2. Develop Machine Learning Model: 22nd April - 1st May

3. Develop Natural Language Process Model: 22nd April - 1st May
4. Develop twitter crawlers: 24th April - 27th April
5. Build CouchDB cluster with Docker: 26th April
6. Design API document: 27th April - 11th May
7. Develop backend and frontend: 28th April - 12th May
8. Run twitter crawlers: 28th April - now
9. Run Machine Learning Model: 1st May - now
10. Run Natural Language Process Model: 1st May - now
11. Statistic Results: 11th May - 13th May
12. Fetch Aurin Results: 11th May - 14th May

4.4 Cloud System Architecture Implementation

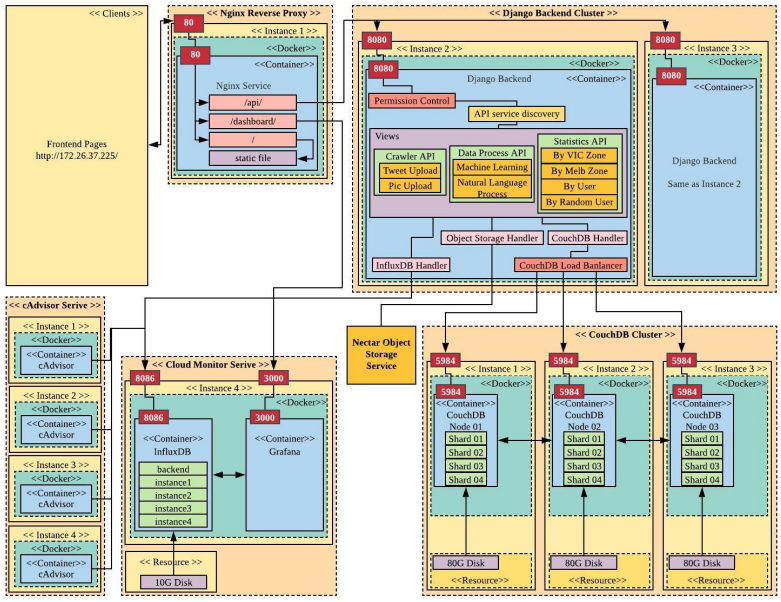

Figure 11: Cloud Architecture

4.4.1 Microservices Arrangement

Figure 12 generally shows how we allocate the services in order to balance the workload across all instance we have. From our experiments, CouchDB and

Machine Learning Services would cost most computation resources, thus we decided to use 3 instances to construct CouchDB cluster, each deployed 60GB volumes, and put machine learning at the 4th instance.

In order to avoid exposing all the instance IP addresses and ports to users, we also construct an Nginx service to reverse proxy all the services, with this design, user can only access the port 80 of instance 1. The frontend container is located at the same instance with Nginx to lower the latency to load webpages.

The backend service is located at instance 2 as well as crawlers, then the crawler could upload tweets to backend directly instead of through gateway. The other backend was placed at instance 3. Natural Language Process and Machine Learning was separated on different instance because both of them consume many resources. The cloud monitor services were placed at instance 4 because they are light level so would not compete for resources with machine learning. The system performance will be presented later.

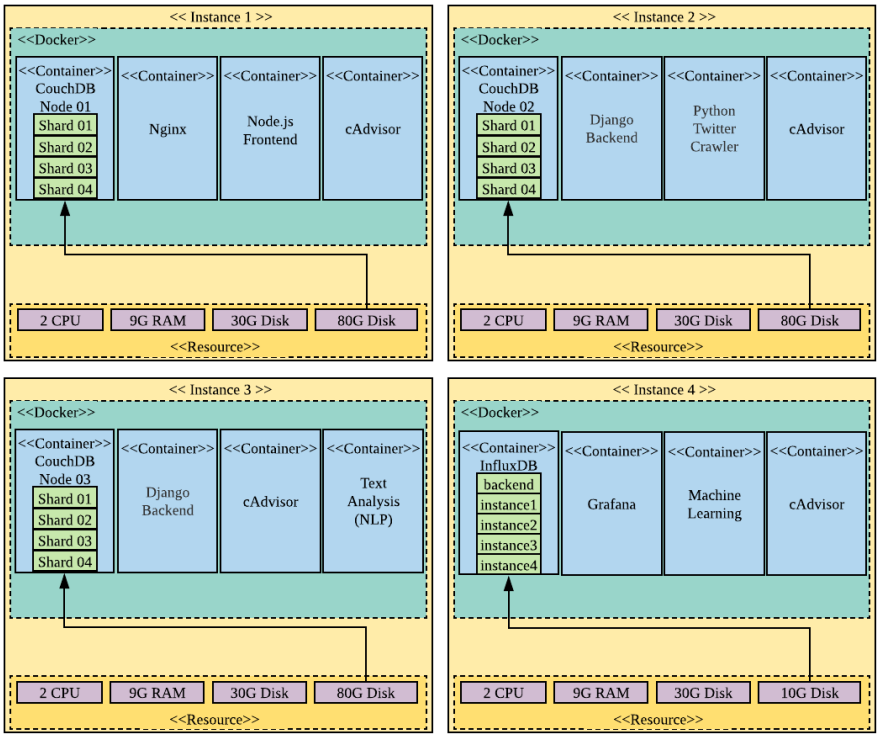

Figure 12: Service Arrangement

4.4.2 CouchDB Cluster

Figure 13 represents the architecture of CouchDB cluster. Actually, a single node database was also acceptable but sometimes it may crash and results in all services shut down, so, in order to provide high availability database service, we construct a cluster with 3 nodes and each of these containers maintain a completed copy of the dataset, each dataset have 4 shares, thus, the database service is still available even 2 nodes in the cluster shut down, and still can provide fully read-write function.

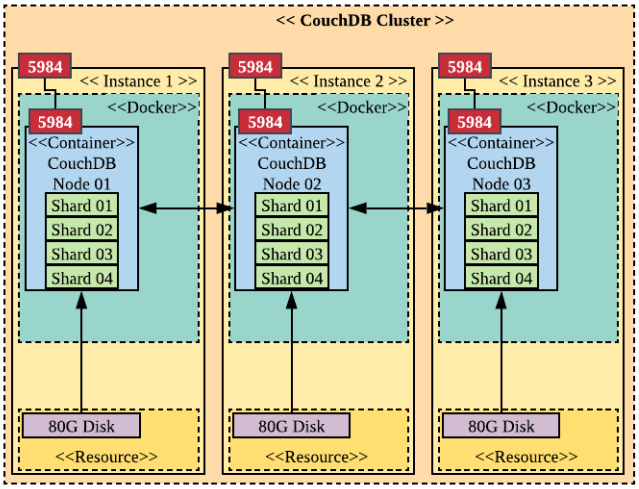

Figure 13: CouchDB Cluster

4.4.3 CouchDB Design Doc and Map Reduce Functions

The design document:
`https://www.yuque.com/docs/share/f1603623-ad6f-44c7-a40c-09867a0487df`
At the beginning we tried to use the mango query feature which provided by CouchDB after version 2.0, this feature was used to replace the temporary views, it works well at first because of the small amount of data, but soonly after achieving 2 million tweets, one statistics query may take half an hour! It's unbearable, thus we decided to rebuild an appropriate MapReduce design doc. After this modification, the longest query time was faster to 15 seconds and the simple queries are accelerated to less than 1 second. Greatly improve the performance of the whole system.

There are two documents, the first one is used to query the statistics results of our machine learning and natural language processing:

1. **time_geo_all_tags:** this view uses time as key and all necessary segments as value for statistics according to time, then the website can use this view

to display random users' moving path, and can also display the machine learning results tags of their tweets.

2. **user_geo:** this view uses username as key, thus it provides the website the ability to track a user and display what he said in a tweet and his sentiment or even the photos.

3. **zone_tags:** this view uses Melbourne districts and tags as key and the reduce function is _sum, thus it can statistics the number of tweets that match our expectation in each Melbourne districts.

4. **vic_zone_tags:** similar to zone_tags, this view uses Victoria State districts and tags as key, thus website can statistic the results according to VIC.

5. **machine_result:** this view uses machine learning tags as key, and use _sum as reduce function to statistic the results of machine learning.

6. **text_result:** similar to machine_result.

The second one is used to query the tweets have not been processed:

1. **machine:** this view used to query the tweets haven't been processed by machine learning.

2. **text:** to machine view, used to query tweets haven't been Natural Language Process.

3. **zone:** query unlocated tweets by Melbourne districts.

4. **vic_zone:** query unlocated tweets by VIC districts.

4.4.4 Backend Service Subsystem

Backend APIs document:
`https://www.yuque.com/docs/share/0394e4d9-965c-4ddb-837f-7a04d8098e28`
The most important part of the whole architecture is the backend and how backend connects with other components to construct a subsystem to support the whole cloud application. The detailed design of the backend was shown in Figure 14.

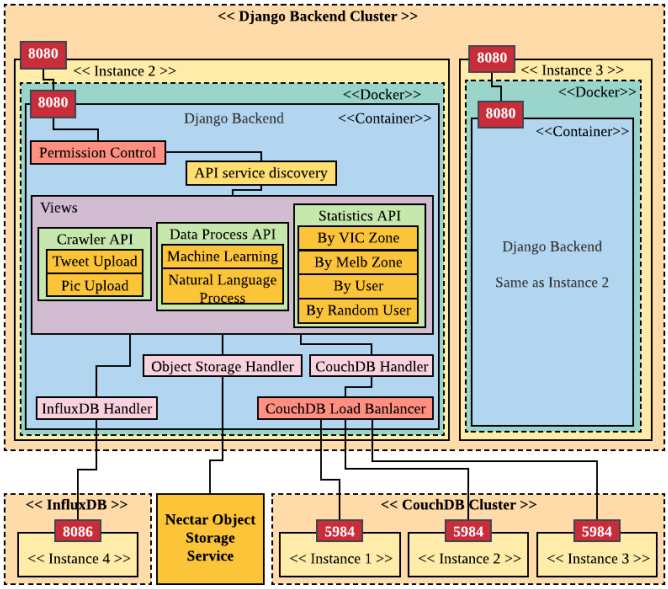

Figure 14: Backend Subsystem

Firstly, in order to avoid malicious access to our APIs, we built a simple permission control component to check if request brings a certain api-key.

Then the API service discovery component would route the request to corresponding views, for crawlers, there are APIs used to upload tweet and picture. The Machine Learning Script and Natural Language Process Script were able to grep unprocessed tweets from backend and upload them after processing. Finally, the frontend could send certain query parameter to backend and receive the statistics results in a short time.

The InfluxDB handler is used to operate the influxdb that we build in instance 4, the main function is used to display the time that our APIs called and the latency to return value, then we are able to analyze the performance of our whole system.

Object storage handler developed with a python-swift package, used to communicate with Nectar Object Storage Service and upload or download pictures, we also use this service to store the results of statistics as a cache, when user query with the same parameters, the backend could download the cache results as give it to users directly, which can lower the query time from 15 seconds to less than 1 second. However, there is a problem that the service is not stable enough, sometimes the object container would be inaccessible, this is one of the problems of the research cloud.

The CouchDB handler is a component used to operate the database and because we construct a cluster and we used python-couchdb package, it was

not convenient to use Nginx to proxy the request. Thus we integrate a Load Balancer to distribute the request to different nodes, and with this design, the total throughput per hour of CouchDB cluster reach 804k per hour and the potential throughput should over 1 million per hour.

Because the backend needs to support all the other service, we run two docker containers of the backend to provide a high availability service. The Nginx on instance 1 would automatically balance the workload between these two containers.

4.4.5 Handle Duplicate Tweets and Error Handling

We play a trick to solve the problem of duplicate tweets, after we analysis the official document from Twitter, we ensure that the id of the tweet are universal unique id, which means tweet id would never be duplicated in Twitter's database, thus, we can use the tweets' id as the document id in our CouchDB database directly. When our crawlers send a tweet with duplicate id to the backend, it would not be recorded in the database because the id already exists in it. This trick helps us avoid thinking about complex data cleaning algorithms. What's more, from the cloud monitor we found that our crawlers get more than 1 million duplicate tweets, and all of them were rejected by the backend.

All of our key services have redundancy backup. For example, if one of our backend services fails, the docker daemon process will restart it immediately, even when one of backend restart fails, we still have another working backend and the Nginx will proxy all the request to the running backend. The only possible unavailable situation is both backends fails and could not restart, it is unusual. The CouchDB cluster also provides high availability services because each node maintains a copy of the dataset, which means even 2 nodes fail, the database service is still accessible. The other services are more stable and seldom fails, thus it is acceptable to run only one container.

The details of server error handling like how the backend holds the error including duplicate tweets or query parameter wrong and then return an appropriate message to clients are too lengthy, it would better to check our code directly. The basic idea is to use try..except to catch all the error and process to possible reason then give it to clients, and send message to InfluxDB to record the error.

4.4.6 Cloud Monitor Service Subsystem

API monitor:
`http://172.26.37.225/dashboard/d/ji261NiWz/api-monitor?orgId=1&refresh=%2010s&from=now-1h&to=now&kiosk`
Resource usage monitor:
`http://172.26.37.225/dashboard/d/fZ6poDmZk/instance-resource?orgId=1&refresh=%202s&from=now-1h&to=now&kiosk`
During the development, we found that was really necessary to build some monitoring service to display our dockers' status and the resource usage of our

instances, thus we soonly construct a cloud monitor service with cAdvisor + InfluxDB + Grafana. InfluxDB is a kind of time sequence database which can store key-value pairs with timestamps and build index according to timestamps. Then the users or applications could query the statistics results in a very short time.

The cAdvisor are able to collect all the hardware usage information from instances and send them to InfluxDB.

Then the Grafana could connect to InfluxDB and provide a beautiful frontend interface and very easy to customize for display what we need.

With this monitor service, we can check the status of all of the other services like how many tweets the crawlers received or how many pictures had been processed by machine learning script. What's more, if the error number reach limit value, Grafana would send notice to us, then we can fix bugs earlier. This service greatly improves the efficiency of development, we never need to spend time to SSH the instances to check the dockers.

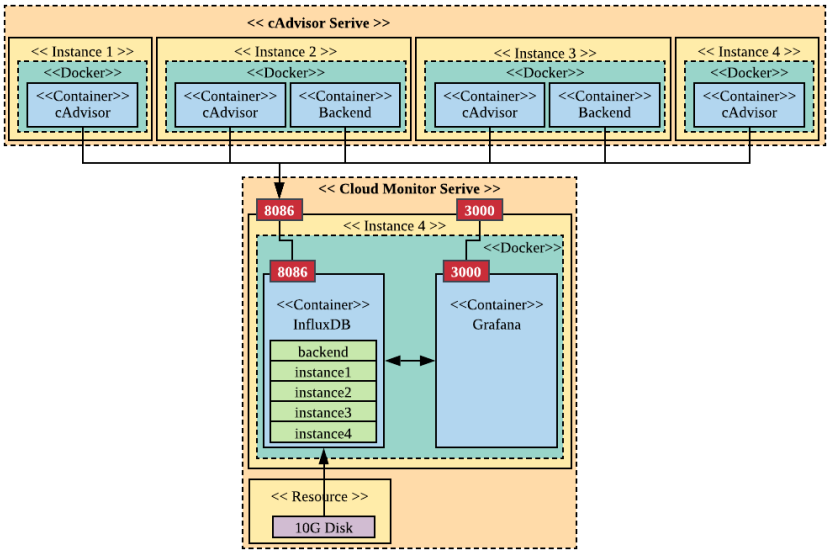

Figure 15: Cloud Monitor Subsystem

4.4.7 Cloud Monitor and Performance Analysis

Figure 16: CPU and Memory Resource Usage

Firstly, the CPU resource arranged for each group was insufficient, from the figure 16 we could found that when all of our services were fully running, the CPU usages of all instances are over 90%. However, it also represents that the workload is balanced between instances, which means our architecture design is generally correct.

Figure 17: CouchDB Load Balancer Monitor

The maximum throughput of CouchDB cluster reaches 804k per hour, which was even beyond our expectation. Thus we believe the potential throughput should reach 1 million.

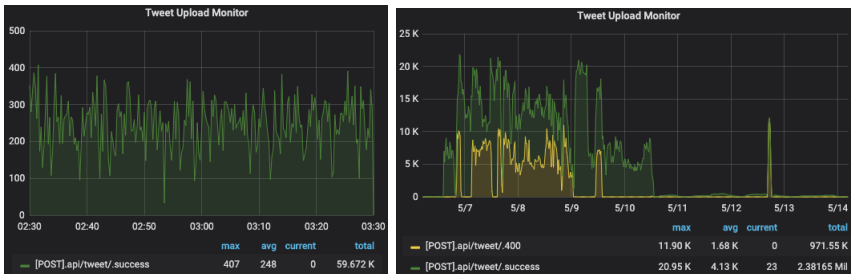

Figure 18: Tweets Upload Monitor

When all of our crawlers working, we can record more than 59k tweets per hours and we had stored more than 2.38 millions of tweets. After we only running the real-time crawler, we found there are only 1k tweets published in Australia per hour. What's more, there were also about 1 million duplicate tweets were rejected by the database.

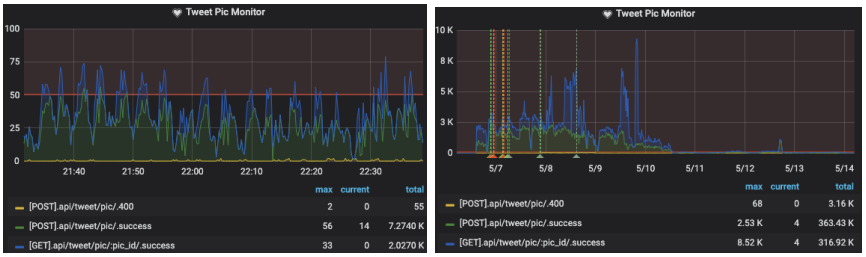

Figure 19: Tweets Picture Upload Monitor

The Object Storage Service saved at most 7k pictures per hours when all crawlers working, and the total number of photo reach 363k until now.

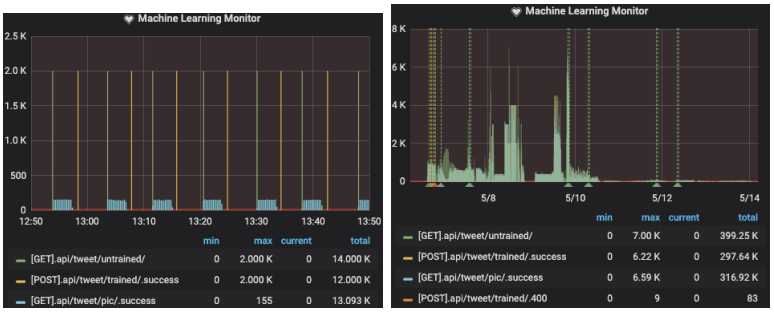

Figure 20: Machine Learning Monitor

The machine learning model can process about 12k of picture per hours, able to match the speed that we store pictures. The machine learning model

processed 297k pictures finally.

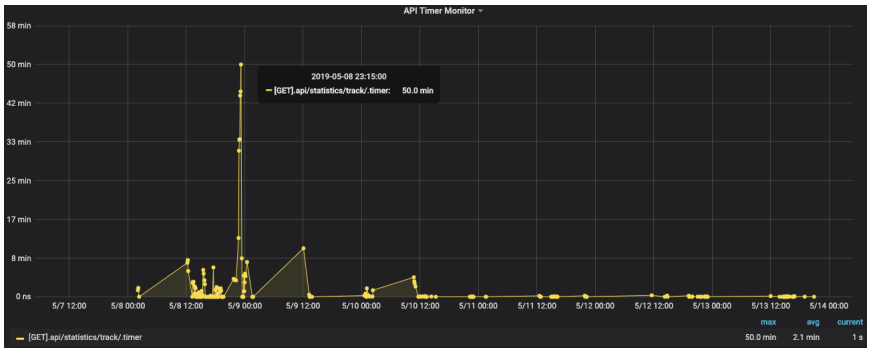

Figure 21: API Timer Monitor

When we used mango to query the statistics results from CouchDB, we found the hardest query spent 50 minutes, which was definitely unacceptable, thus we reconstructed the code fastly and decided to use MapReduce to query the data, then the query time lower to less than 15 seconds.

According to all we discussed above, the total throughput of the backend could reach 900k per hour, which is enough for most applications.

4.5 Data Analysis Results

Our crawler focused on the entire Melbourne area, with all the image collected from our database, the total statics are displayed at tables below. However, our image recognition model is not perfect, although it used the states of the art techniques. We have around 7% false positive predictions that indicate there are 7% possibilities that a certain image is misclassified as a certain class. However, the total distribution of the class should be the same. Unsurprisingly, oysters and mussels are very popular among the food categories. The macarons and ice cream are also the most popular image in the Melbourne Area. However, due to the number of non-food images and the number of tweets that have actual geo-locations, we are unable to produce enough data to compare with AURIN for Gluttony. The majorities of NSFW class label are natural and drawing, the Internet Meme could be classified as natural or drawing based on the proportion of text size of the image. The NSFW images (Porn, Hentai and Sexy class) are actually a small set of images and our system also misclassified some of the images. We combined the text analysis and image recognition result into a single set of tags for frontend to display and compare with AURIN data.

To illustrate the relationship between AURIN data and Twitter data, we make use of three datasets of AURIN. For the data analytics, we determine to analyze two aspects: the economy level of a region and the stability of a region. To illustrate the economy level of a region, we make use of the dataset “ABS - Data by Region - Education & Employment (LGA)” and extract the

Image Class Label (NSFW)	Count
Natural	168,166
Drawings	23,878
Porn	14,560
Sexy	1,916
Hentai	1,688

Table 5: NSFW Image Statistics

Image Class Label (Food179)	Count
Non Food	256,555
Oysters	274
Macarons	38
Ice Cream	31
Mussels	28

Table 6: Food179 Image Statistics

number of unemployment people of each Local Government Area (LGA). For the stability of a region, we use the dataset "PHIDU - Admissions - Hospital Types and Sex (PHN)" and extract three features: "total number of admissions for all hospital", "total number of male admissions for all hospital" and "total number of female admissions for all hospital". We also apply the data set "LGA Number of Offences in Victoria by Offence Type" and extract six features in this dataset: "weapons and explosives offences", "assault and related offences", "sexual offences", "robbery", "stalking harassment and threatening behavior" and "grand total offence count". We notice that datasets of AURIN summarize statistics based on LGA. Therefore, we implement our frontend map to match with such a division of LGA. Moreover, we only perform data analytics for LGAs of Victoria state, because most Twitter data we collected come from Victoria state. The distribution of statistics for each category are visualized as follow: (The area with darker number includes a larger number of cases. However, "Black" area means there is no case in this region or this area doesn't belong to any LGA — "Unincorporated")

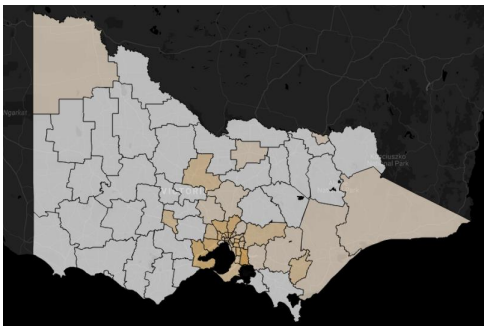

Figure 22: Unemployment

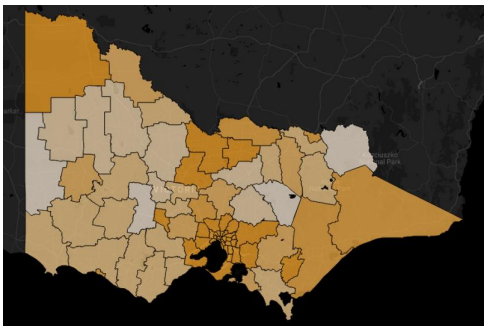

Figure 23: Total hospital admission

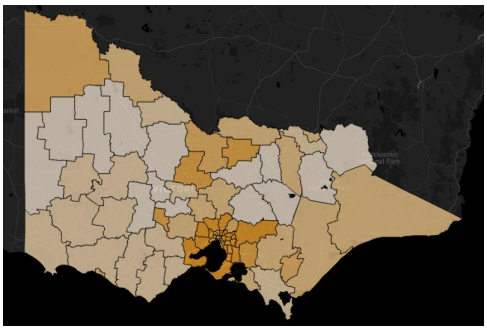

Figure 24: Male hospital admission

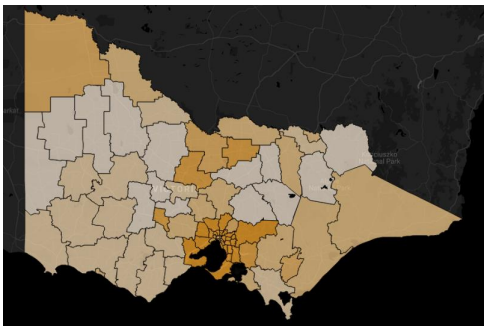

Figure 25: Female hospital admission

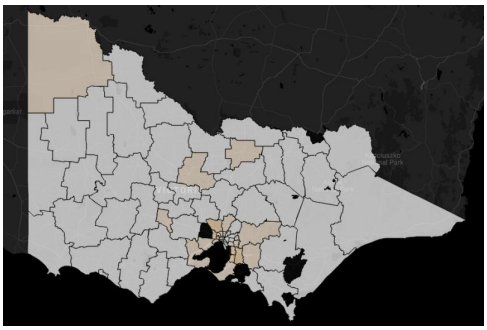

Figure 26: Weapon related offence

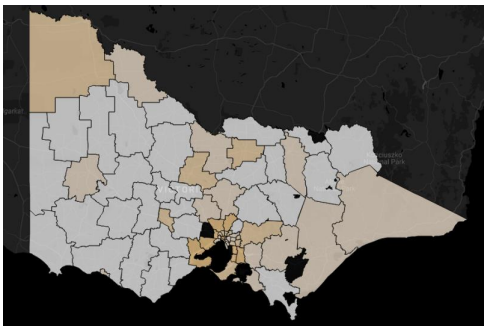

Figure 27: Assaults

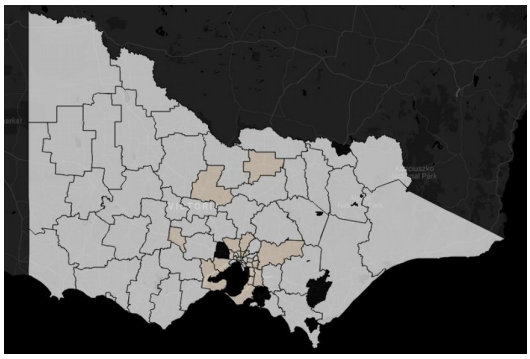

Figure 28: Sexual offence

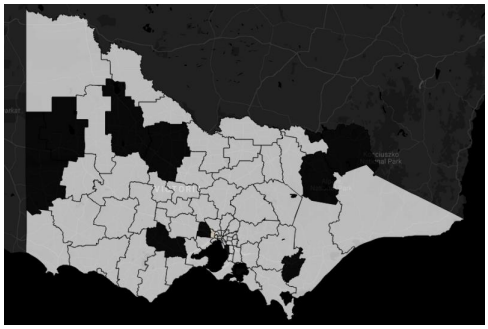

Figure 29: Robbery

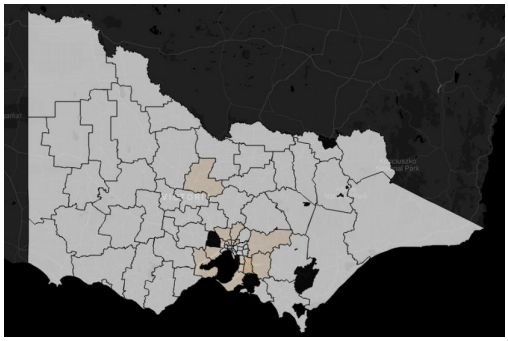

Figure 30: Harassment and threatening

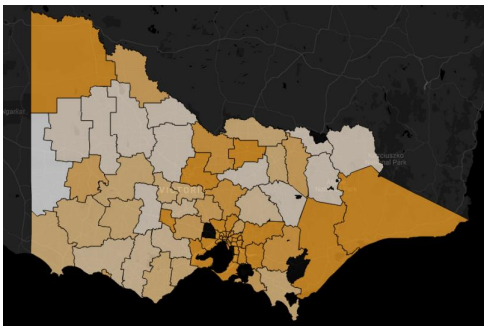

Figure 31: Total offence

Before comparing with Twitter data, we can already find some patterns of relationship between each category AURIN data:

1. The statistics of "Unemployment" by LGA matches with the statistics of "Assaults" by LGA. It indicates that the economy level in that region influences the stability of a local government area.

2. The number of robbery cases is low for all regions. Some regions don't have any robbery cases at all.

3. The statistics of "total hospital admission" matches with the distribution of the number of "total offence". This result is a surprise because the number of "total hospital admission" includes people who get into the hospital due to diseases. However, the match between these two distributions indicates that most people may get into the hospital because they are hurt, either physically or mentally, by one of the above categories of offence. We can conclude that there is a strong positive relationship between "total hospital admission" and "total offence".

For our Twitter data, since most tweets, we harvest come from the Unimelb Research Cloud, whose twitter data are grouped by the names of capital cities. In the process of harvesting tweets, we defined the "location" in the query as "Melbourne". Thus, most tweet we harvest come from the LGAs surrounding Melbourne. For LGAs that are distant to Melbourne, the number of tweets captured is low. The visualization of Twitter data is as follow.

Figure 32: The Visualization of Twitter data

In the data analytics, we only focus on LGAs which have a large number of tweets captured (Areas that are neither white nor black). In the distribution of Twitter data, the areas with "orange" color are the ones that have a large number of tweets containing "lust" or "wrath" messages or have more images identified as "porn". The distribution of "total sins" (tweets that are classified as "lust" or "wrath" by text analysis or tweets carrying images identified by image recognition as "porn") are shown below.

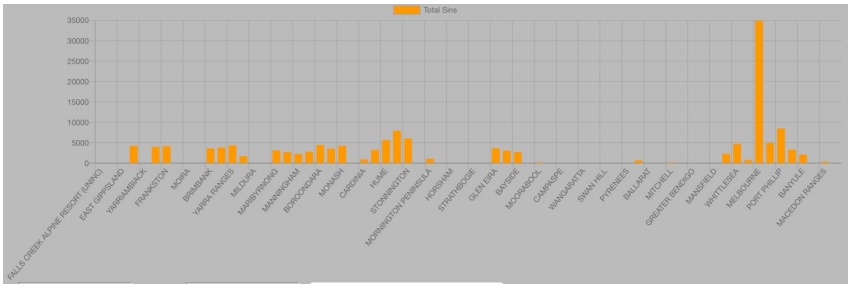

Figure 33: The distribution of "total sins"

We can find that the LGAs with larger population density have a higher number of total sins. Melbourne is such a typical example. However, we should remember that the statistics of the tweet can also be influenced by the popularity of Twitter in that region. For some LGAs, such as Mildura, the number of total sins is meager. However, according to AURIN data, the number of total offences in this area is exceptionally high. The difference between tweet statistics and AURIN data comes from the number of Twitter users in this region. When Twitter is not popular in the region, we cannot make a reasonable summary of the occurrence of sins from tweets.

Moreover, by comparing the distribution of tweet data with that of AURIN data, we can find that, areas with more "lust" or "wrath" tweet messages or more "porn" images tend to be more unstable, and more offence cases of every category tend to happen in this area (except for robbery cases). These areas

also suffer from a high unemployment rate. Notably, we find that there are more offence cases in LGAs with a large population, such as Melbourne and Hume. The sentiment analysis finds that the proportion of non-neutral tweets from these areas is higher than other LGAs. Therefore, the actions of offence may be related to the sentiment bias indicated by tweet text. People who have sentimental polarity may tend to conduct offence actions.

In conclusion, the statistics indicated by tweet texts and images match with the general distribution of economy level and stability for LGAs reflected by AURIN data. However, this indication from tweet content only applies to the region where a large proportion of popularity uses Twitter in daily life. For regions where there are not many Twitter users, we should refer to the official data of AURIN.

5 Nectar Cloud

There are a few pros and cons we discovered when we work on this project with the UniMelb Research Cloud.

5.1 Advantages

There are a number of strengths using the cloud for this project. The backbone network provided high-speed internet access when we are retrieving the images as well as tweets datasets. The internal network between instances is also fast enough that are comparable with local access. For example, when we design our image recognition system communication with our backend, how to retrieve the image is one of difficult design choices we have to make. However, since the internal communications between instances are fast, we simply used RESTful API to retrieve the images from our database through the backend system.

Compared with Spartan HPC system, it is also very easy to set up the environment we need for our software to run. HPC has a fixed set of modules. Cloud system has several newest version OS images as well as their legacy versions to choose from. We could build our environment stack based on these OS images. We can customize the Modules. The UniMelb Research Cloud also provides API that we could utilize tools such as Ansible to automatically create stack of instances based on our configurations. It fits the idea that our system needs to be scalable.

The cloud-based system also provides security features that help to reduce the burden of our system. Since a large part of Network security is already handled by the Cloud, we could focus more on other aspects of our software system. However, we do need to understand how the security groups work in order to get the entire project work flawlessly.

5.2 Disadvantages

There are a few drawbacks that we encountered during this project. The Object Storage system fails periodically so part of our system will not be able to access the data. Although it will not suspend the entire software nor fail the subsystem, it does put some of our data analysis tasks into a suspension. Availability of the cloud resources is another disadvantage. Although the cloud system should, in theory, have an unlimited amount of resource as long as new nodes being added into the cloud, we do experience overload when we work on the project. The resources allocated for this project are not enough. Another disadvantage is that we have to consider DevOps when we design our system. The monitoring system is needed to ensure all systems are running smoothly.

6 Conclusion

In conclusion, we developed a robust software system capable of handling a large amount of datasets. We focused on twitter data in Melbourne and Victoria area with our image recognition and text analysis process. Additional datasets and analyzing models can be added easily since our system design considers the scalability. Our system is also fault-tolerant so that subsystem can fail without interrupting other services. Overall, this is an industrial-level project that can be deployed for real-world scenarios.

References

He, Kaiming et al. (2015). "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034.

— (2016). "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Henrique, J (2016). *GetOldTweets-python [Computer software]*.

Hill, Mark D and Michael R Marty (2008). "Amdahl's law in the multicore era". In: *Computer* 41.7, pp. 33–38.

Howard, Andrew G et al. (2017). "Mobilenets: Efficient convolutional neural networks for mobile vision applications". In: *arXiv preprint arXiv:1704.04861*.

Huang, Shian-Chang et al. (2011). "Using supervised kernel locality preserving projections to improve classifier performance on credit rating forecasting". In: *Journal of Information and Optimization Sciences* 32.1, pp. 189–204.

Kawano, Yoshiyuki and Keiji Yanai (2014). "Automatic expansion of a food image dataset leveraging existing categories with domain adaptation". In: *European Conference on Computer Vision*. Springer, pp. 3–17.

Loria, S (2012). "Textblob python library or sentiment analysis". In: *sloria/-TextBlob on GitHub at commit eb08c12 "Twitter via sms faq," April* 13.

Luo, Liangchen et al. (2019). "Adaptive gradient methods with dynamic bound of learning rate". In: *arXiv preprint arXiv:1902.09843*.

Miller, George A (1995). "WordNet: a lexical database for English". In: *Communications of the ACM* 38.11, pp. 39–41.

Russakovsky, Olga et al. (2015). "Imagenet large scale visual recognition challenge". In: *International journal of computer vision* 115.3, pp. 211–252.

Wu, Zhibiao and Martha Palmer (1994). "Verbs semantics and lexical selection". In: *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*. Association for Computational Linguistics, pp. 133–138.