Unversity of Melbourne

COMP90024 Cloud and Cluster Computing

Assignment 2 - Twitter & Aurin (Team 3)

Ke, Kimple (523692)
jke@student.unimelb.edu.au
Tang, Fei (581810)
feit@student.unimelb.edu.au
Jin, Bofan (583688)
bfjin@student.unimelb.edu.au
Li, Roger (585361)
r.li3@student.unimelb.edu.au
Ye, Zeyu (290159)
zeyuy@student.unimelb.edu.au

SUPERVISED BY
Prof Richard Sinnott — rsinnott@unimelb.edu.au

May 12, 2016

Contents

1 System Design and Architecture

The system was designed to run on multiple machines in parallel to keep harvesting tweets using the Twitter API for the purpose of analysis. The machines used can be physically existing or virtual machines. We used 4 virtual machines with 8 cores (32Gb memory total) and up to 250Gb of volume storage and 250Gb of object storage, because of the restriction. On each of the virtual machines, we have the Python scripts running and harvesting data independently.

Since Twitter has restricted the number of tweets we can retrieve per unit of time, it is difficult to harvest datasets that are large enough to do meaningful analysis. To overcome this issue, we have designed our script so that each of the harvesting of tweets is distributed among the four available virtual machines. The two states we have chosen to harvest tweets from are Victoria and New South Wales, as they have the highest populations and we would expect to be able to gather more tweets from these two states. We also created four different Twitter accounts to further increase the rate at which we can harvest tweets.

Of note is that most of the tweets do not contain geolocation information, and therefore we cannot make any conclusions from those tweets or do any comparison against the datasets we have in Aurin. These tweets will be discarded, and those that contain geolocation will be stored to database.

All tweets harvested were then preprocessed (reverse geocoding and sentiment analysis) and then stored in the database.

The next components of the entire architecture are the data analysis and visualisation web application, for which we will have chosen topics and a range of scenarios that we will analyse. We perform this analysis by retrieving relevant datasets from couchDB and downloading relevant datasets from Aurin for comparison and analysis. The result of this analysis is itself also saved into our databases. We will then use the data we have collected, filtered and aggregated to visualise them in appropriate types of graphs.

2 System Requirements and Usage

2.1 Libraries

Our application are all coded in python and consists of 3 main parts - mining, analyze and visualization. The mining uses twitter library to mine and extract the tweets, these will then be stored into CouchDB using the couchdb library. The analysis part first uses nltk to preprocess data with the sentiment analysis and SLA, then use MapReduce in CouchDB to retrieve filtered data and correlate them with AURIN data, and the results will then be formatted and pumped into our visualization webapp. The webapp uses django framework.

Application Libraries

```
# Install build essentials and common libraries for
   build (use sudo if permission required)
## Mac
```

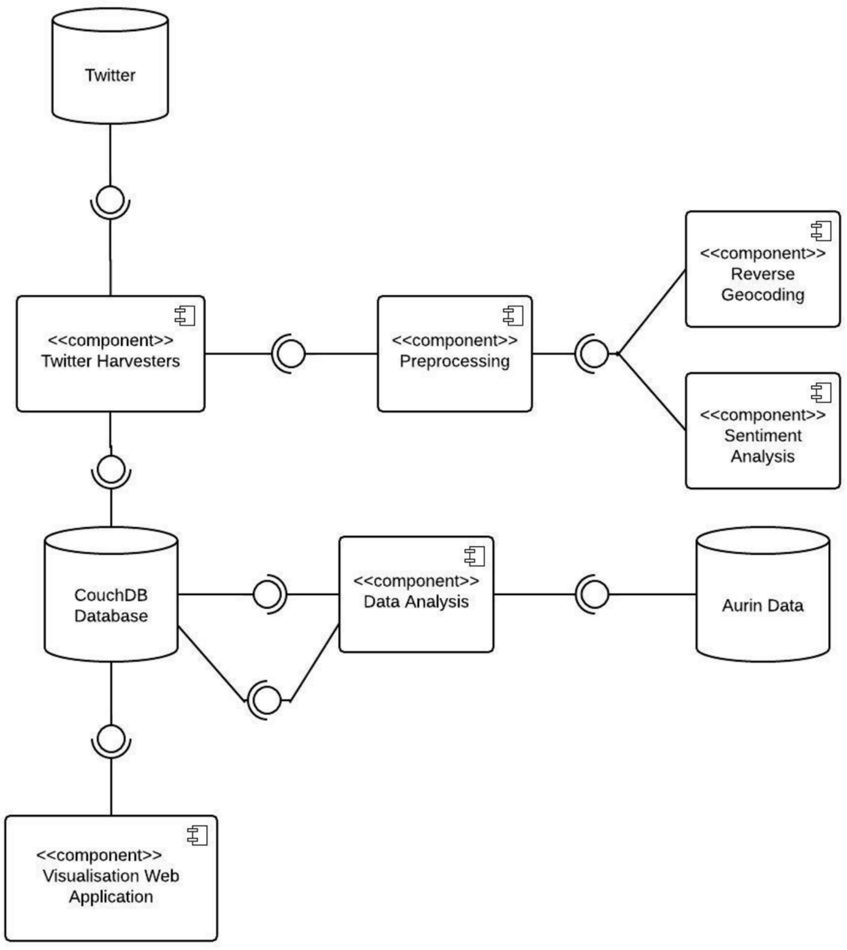

Figure 1: Overview of System Architecture

```
## - just install xcode
## Ubuntu
apt-get install software-properties-common build-
    essentials
#Install pip (use sudo if permission required)
## Mac
easy_install pip
## Ubuntu
apt-get install python-pip python-dev
# for the search and analysis application -
    twitter_miner, install nltk couchdb and twitter
pip install nltk twitter couchdb
# for the web app application -
    cloud_computing_data_visualization
pip install django
```

2.2 Application Usage

To get data for scenario analysis, run the analysis.py file with the following command:

analysis.py usage

```
python analysis.py --topic <topic> --db <db_name> --
    keywords <keywords> --aurin <aurin_data_path>
```

where topic is just an int indicates which scenario number this is, db is the database name for this search, keywords can be a any strings that is going to be searched, multiple keywords are allowed and aurin is the csv data downloaded from the Aurin database which is used for analysis.

A sample command will be

analysis.py example

```
python analysis.py --topic 1 --db coormelbourne --
    keywords afl --aurin ./data/sla_unemployed.csv
```

The income_analysis.py is similar to analysis.py but it is for a specific scenario and since it involves sentiment analysis and it does not need a keyword and therefore it is a stand alone script.

To run this script, type the following command:

analysis.py example

```
python income_analysis.py --topic <topic> --db <
    db_name> --output <output_data_path> --aurin <
    aurin_data_path>
```

where topic is just an int indicates which scenario number this is, db is the database name for this search, output is the string of the output file path and aurin is the csv data downloaded from the Aurin database which is used for analysis.

A sample command will be

analysis.py example

```
python income_analysis.py --topic 1 --db coormelbourne
     --output result.csv --aurin ./data/sla_income.csv
```

The rest of the script will be automatically invoked when the virtual machine is deployed.

3 System Functionalities

3.1 Twitter Harvesting

Twitter harvesting in the system can be achieved by two different methods. One method uses the Twitter streaming API, while the other method uses the Twitter search API. Both methods are capable of utilising several Twitter accounts in order to increase the speed of harvesting tweets.

3.1.1 Streaming API

The streaming API is used by an application which is continuously run in order to collect an ongoing stream of tweets. In our system, we use the four available virtual machines to run four separate instances of this application which uses four different Twitter accounts to access the Twitter streaming API. As shown in Figure 2, the application will first send request to the Twitter streaming API and harvest a stream of real-time tweets. The stream of tweets is filtered by location of a certain area in Australia. This is achieved by specifying the bounding-box coordinates for each area when calling the Twitter streaming API. Each instance is aware of which area to collect tweets from (and which coordinates to use) through a command line parameter. Each tweet that is received from the stream of tweets will be preprocessed (see data preprocessing) and stored into the streaming database.

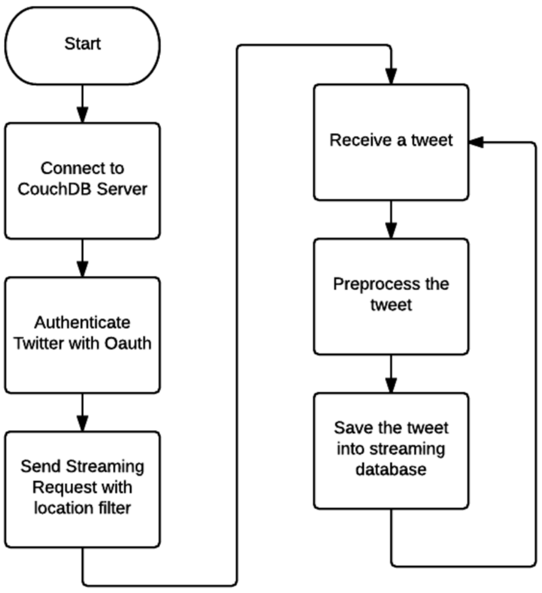
Start
Connect to CouchDB Server
Authenticate Twitter with Oauth
Send Streaming Request with location filter
Receive a tweet
Preprocess the tweet
Save the tweet into streaming database

Figure 2: Flowchart of streaming API harvesting application

3.1.2 Search API

However, Twitter's Streaming API only provides us with about 600 tweets per hour, which is too slow to create a comprehensive database. So Twitter's Search API is also used for the purpose of harvesting more tweets and increasing the data set for further analysis.

The Search API harvesting application retrieves tweets in a different man-

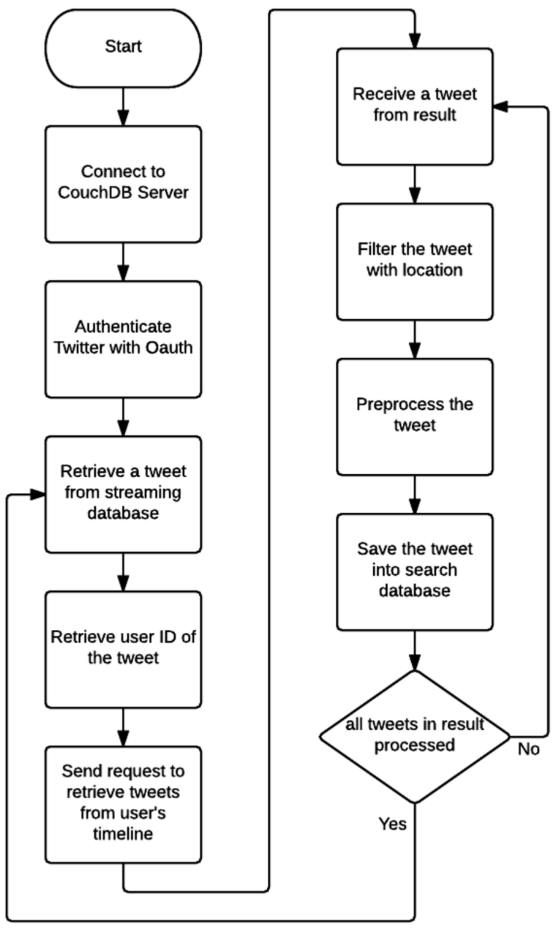
Start
Connect to CouchDB Server
Authenticate Twitter with Oauth
Retrieve a tweet from streaming database
Retrieve user ID of the tweet
Send request to retrieve tweets from user's timeline
Receive a tweet from result
Filter the tweet with location
Preprocess the tweet
Save the tweet into search database
all tweets in result processed
No
Yes

Figure 3: Flowchart of search API harvesting application

ner to the streaming API. As shown in Figure 3, it iterates through each tweet already present in the streaming database, and retrieves more tweets from that Twitter user's timeline. Because we know that the tweets we already have in the databases are from a specific location, it is very likely that location of further tweets from that Twitter account will also be from that location. This allows us to harvest even more tweets in that area. Once that specific Twitter account has been searched, it is marked as 'searched' in the database, such that the account will not be searched again.

All tweets are preprocessed (location and sentiment tag calculated and added), before being inserted into the relevant CouchDB database based on the location of the tweet. There are two separate databases (for tweets retrieved from the streaming and search APIs), one for each state that we are doing analysis on.

3.2 Data Processing

The preprocessing step involves actions on the tweets which occur prior to the tweets being saved into the database. These steps are reverse geocoding to determine the Statistical Local Area (SLA) of a tweet, and sentiment analysis, which classifies each tweet as either ‘positive’, ‘neutral’, or ‘negative’.

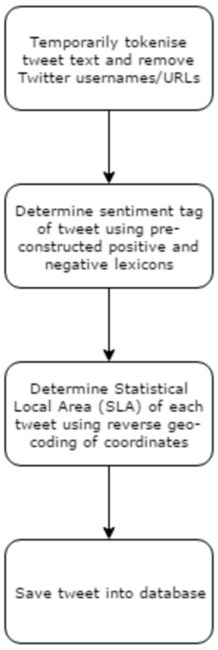
Temporarily tokenise tweet text and remove Twitter usernames/URLs
Determine sentiment tag of tweet using pre-constructed positive and negative lexicons
Determine Statistical Local Area (SLA) of each tweet using reverse geo-coding of coordinates
Save tweet into database

Figure 4: Flowchart of preprocessing

3.2.1 Reverse Geocoding

Reverse geocoding is a preprocessing step which takes the coordinates of a tweet and assigns it to an SLA. This is done by calculating the most likely postcode of the location of a tweet given its longitude and latitude. Longitude and latitude values for Australian postcodes could be found online, as well as a mapping of postcodes to SLAs. Once the postcode is determined, it is mapped to the appropriate SLA, and this SLA code is attached to the tweet prior to insertion into a database.

3.2.2 Sentiment Analysis

The second preprocessing step is sentiment analysis of tweets. This involves identifying tweets which exhibit overall positive sentiment or negative sentiment, and classifying tweets as either ‘positive’, ‘neutral’ or ‘negative’.

The system classifies each tweet into one of these three categories by building a two lexicons: one containing positive words and the other containing negative

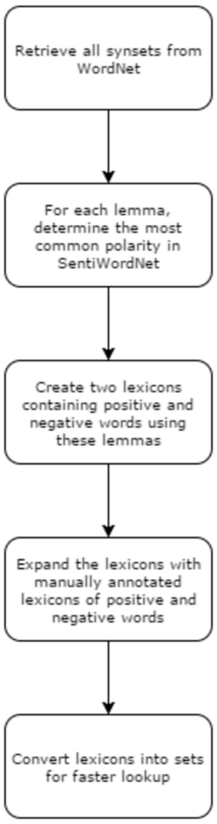
Retrieve all synsets from WordNet
For each lemma, determine the most common polarity in SentiWordNet
Create two lexicons containing positive and negative words using these lemmas
Expand the lexicons with manually annotated lexicons of positive and negative words
Convert lexicons into sets for faster lookup

Figure 5: Flowchart of sentiment lexicon generation

words. For each tweet, if it contains more words from the positive lexicon, it will be classified as a 'positive' tweet and given a sentiment tag of 1. Conversely, if it contains more words from the negative lexicon, it will be classified as a 'negative' tweet and given a sentiment tag of -1. If there are an equal number of words from each lexicon, the tweet will be classified as a 'neutral' tweet and given a sentiment tag of 0. In order to compare the tweets with the two lexicons, the text is also preprocessed to remove URLs and references to Twitter accounts, as well as to tokenize the text into words.

The positive and negative lexicons used for sentiment analysis are generated in two steps.

The first step uses the WordNet and SentiWordNet lexical resources. WordNet and SentiWordNet group words into synsets, which are essentially groups of synonyms. SentiWordNet additionally contains polarity scores for each synset, representing positive, negative and neutral sentiment. For each synset in WordNet, the corresponding polarity value is looked up in SentiWordNet, and that value is remembered for each individual word. It is possible for words to have multiple polarity values in SentiWordNet, each representing a different definition of that word.

For the purposes of sentiment analysis in this system, the most common polarity value across all definitions of a word is taken as the polarity value for that word. Words with a positive polarity value are then added to the positive lexicon, and words with a negative polarity are added to negative lexicon.

The second step uses a manually annotated opinion lexicon accessible through the NLTK (Natural Language Toolkit) package in Python. These manually annotated lists of positive and negative words are combined with the positive and negative lexicons, which were generated in from SentiWordNet, to build the final lexicons.

These two steps need only be run once per invocation of the Twitter harvesting application. Once the lexicons have been built, they are converted into sets for faster lookup, and used to classify each tweet with the appropriate sentiment tag as previously described.

3.3 Data Analytics

3.3.1 Map

Before doing a research on a scenario, we would come up with a list of keywords, which helps to identify tweets the related to this scenario. As shown in Figure 6, one of the functionalities implemented in the system is to search for tweets containing a specified keyword, or one of the keywords in a given list of keywords. This achieved by using regular expressions to search through the text of each tweet. The implementation of such searching process is within the map function which is written in javascript. The regular expression we used to search is a combination of all keywords in the form of:

/\b keyword 1 — keyword 2 — keyword 3... \b/

As a result, for each tweet that contains at least one of the keywords, the map function will emit SLA of that tweet as key and 1 as value for further analysing using reduce.

3.3.2 Reduce

After the map function, reduce function would be used to count the number of tweets in each SLA and grouped them together into different SLAs. In details, one of the default reduce function "_count" would be used to count the number of tweets. Since the value of each row from the output of map function is 1, the result of reduce function "_count" would be exactly the number of tweets after searching. Then by setting group type to "exact", the result of reduce function will be grouped into different SLAs, while each SLA is associated with the number of tweets in that SLA.

3.3.3 Correlation with AURIN data

Once we have obtained the number of tweets from each SLA from map and reduce, we read in the relevant values from Aurin datasets which we have down-

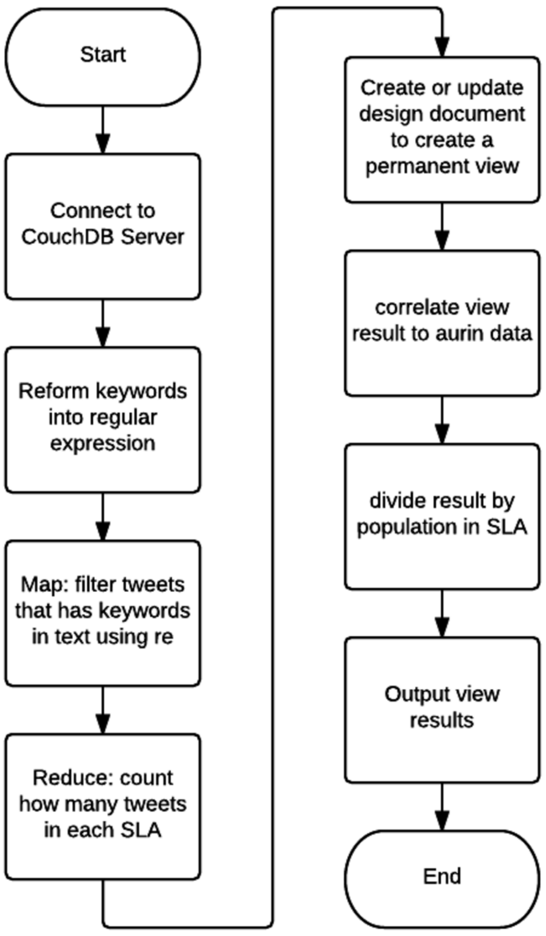
Start
Connect to CouchDB Server
Reform keywords into regular expression
Map: filter tweets that has keywords in text using re
Reduce: count how many tweets in each SLA
Create or update design document to create a permanent view
correlate view result to aurin data
divide result by population in SLA
Output view results
End

Figure 6: Flowchart of analysis application

loaded. The name of the csv file containing the Aurin data is given via a command line parameter.

This data from this file allows us to find, for each SLA where we found at least 1 tweet, the value we are interested in comparing our number of tweets to. However, there are more steps required before the data is able to be analysed or visualised.

Because there are differing populations in each SLA, in order to get meaningful results it is necessarily to normalise all of our numbers such that they can be compared. This is done by multiplying the number of tweets we see in each SLA by 1000, and then dividing by the population of that SLA (for which the data was also obtained from Aurin). This gives us the number of tweets observed per 1000 population.

Similarly, some of the Aurin data also needs to be normalised. For the Aurin data which is already supplied in values per X population, or as a percentage, this step is not required. But for any Aurin data given as a raw count, we multiply the value by 100, and divide by the population of the SLA. This gives us the value for that Aurin dataset per 100 population.

Once these steps are complete, we are able to store this analysis into a database, ready for visualisation.

3.4 Data Visualisation

Once all the data analysis is complete and stored into the relevant databases, we are able to visualise the results of our scenarios in a web application. This web application is implemented in Python with the Django and AngularJS frameworks.

The web application presents the results for our scenarios in the form of graphs which map the Aurin data against the Twitter data.

4 Issues and Challenges

4.1 Error handlings

In terms of parsing input parameters for the Twitter harvesting applications and the data analysis applications, they all use the Python package ArgumentParser. This can be considered some form of error handling, as it allows the applications to detect when incorrect command line parameters are given, while also showing the user an outline of the correct usage. ArgumentParser also allows the applications to dictate which command line parameters are 'required', or essential for the application to run properly. If these parameters are not set or entered in an unexpected format, the application will again alert the user of the issue and outline the correct usage.

5 Deployment Requirements and Usage

5.1 Build Tools - Gulp

We use gulp as the automation tool to auto manage some of the repetitive long command line tasks we do to do a release build for our product. This tools turns out so useful that we eventually ended up using it to automate our docker image building and deployment processes. When we build our application, we will create a dist folder under the root directory which will contain all the compile .pyc file for our application. This allows our code to be protected and encrypted for release.

gulp

```
# To use gulp nodejs have to be installed first (use
   sudo if permission required
## Mac
## - download the pkg installer from official nodejs (
    https://nodejs.org/en/)
## Ubuntu
sudo apt-get install -y build-essentials
curl -sL https://deb.nodesource.com/setup_6.x | sudo -
    E bash -
sudo apt-get install -y nodejs
# Once nodejs is successfully installed, install gulp
    (use sudo permission if required)
npm install -g gulp
```

There're several gulp tasks defined, but the only two that the developers need to use is the default task and the docker-publish task. Below are the usage for these two tasks

The below command runs the default task, it compiles all the python scripts into .pyc file ready to be released and will drop all the .pyc files into a dist folder under root directory, the dist folder will be recreated for every build

default task

```
gulp
```

The below command runs the docker-publish task, it will actually run the default task first to rebuild the .pyc files and packages them into dist directory for our solutions. Then it will build docker image for our application, finally push the built image to our docker repository on docker hub (Twitter Miner: `https://hub.docker.com/r/shuliyey/twitter_miner/`, Cloud Computing Data Visualization: `https://hub.docker.com/r/shuliyey/cloud_computing_data_visualization/`). Note this command requires docker to be installed and started refer to the "Containerization - Docker" section

docker-publish task

```
gulp docker-publish
```

5.2 Containerization - Docker

We use docker containerization technology to containerize our application for both the twitter mining and visual web application. This way it allows our application to be easily shipped, released and managed as docker images (Also docker is just so cool). There're two Dockfile created to automate the docker image creation process. One for Twitter Mining and One for Visual Web Application. Both of these Dockerfiles have been regression tested to containerization the correct environment with all required libraries to run the two web applications.

Docker

```
# Install Docker
## Mac
## - download the pkg from official site (https://docs
    .docker.com/mac/step_one/)
## Ubuntu (16.04)
sudo apt-get install docker.io
sudo usermod -aG docker $USER
# To build docker image for twitter mining application
## make sure you are under the twitter mining
   repository root directory
docker build -t shuliyey/twitter_miner:<tag> . # e.g.
   docker build -t shuliyey/twitter_miner:v1.0 .
## To push/publish the docker image
docker push shuliyey/twitter_miner:<tag> # e.g. docker
    push shuliyey/twitter_miner:v1.0
## To run the docker image
docker run -d -it -e TM_INDEX=<0/1/2/3> shuliyey/
   twitter_miner:<tag> # e.g. docker run -d -it -e
   TM_INDEX=0 shuliyey/twitter_miner:v1.0
# To build docker image for visual web application
## make sure you are under the twitter mining
   repository root directory
docker build -t shuliyey/
   cloud_computing_data_visualization:<tag> . # e.g.
   docker build -t shuliyey/
   cloud_computing_data_visualization:v1.0 .
## To push/publish the docker image
docker push shuliyey/
   cloud_computing_data_visualization:<tag> # e.g.
   docker push shuliyey/
   cloud_computing_data_visualization:v1.0
## To run the docker image
docker run -d -it shuliyey/
   cloud_computing_data_visualization:<tag> # e.g.
   docker run -d -it shuliyey/
   cloud_computing_data_visualization:v1.0
```

5.3 Auto deployments (remote) - Ansible

We use ansible to automate the environment configuration of the 4 VMs we have on NeCTAR. We have 3 ansible playbooks. The first playbook auto configures the environment to auto install/update and start couchdb. The second playbook auto configures the environment to auto pull and run the twitter_miner docker image we have on dockerhub. The last playbook auto configures the environment to auto pull and run the cloud_computing_data_visualization docker image we have on dockerhub.

Ansible

```
# Install ansible
## Mac (use sudo permission if required)
pip install ansible
## Ubuntu
sudo apt-get update
sudo apt-get install software-properties-common
sudo apt-add-repository ppa:ansible/ansible
sudo apt-get update
sudo apt-get install ansible
# To auto configure CouchDB VM (use sudo permission if
    required)
## make sure you are under the root directory of the
   twitter_miner source respository
ansible-playbook playbooks/deploy-couchdb.yml
# To auto configure Twitter Miner VM (use sudo
   permission if required)
## make sure you are under the root directory of the
   twitter_miner source respository
ansible-playbook playbooks/deploy-twitter-miner.yml
# To auto configure Cloud Computing Data Visualization
    VM (use sudo permission if required)
## make sure you are under the root directory of the
   cloud_computing_data_visualization source
   respository
ansible-playbook playbooks/deploy-webapp.yaml
```

To Auto deploy the latest Cloud Computing Data Visualization Application

Auto Deploy

```
git remote add production ubuntu@couchdb-cloud-cluster
   -computing:~/cloud_computing_data_visualization.git
git push -u production master
```

6 Selected Scenarios

For all of the selected scenarios, the data used (both tweets and Aurin data) is at the Statistical Local Area (SLA) level.

6.1 Comparison of trouble with transport and interest in politics

In this scenario, we look for correlations in data related to difficulty with access to services, and the number of tweets related to politics. In particular, we focus on the number of people who claim to have trouble with access to transport. In order to do this, we have extracted from Aurin the relevant data for all SLAs, and used a list of keywords related to politics to search for tweets in our databases.

This scenario can provide some interesting insights into which areas are more interested in politics, and if there are any correlations between their difficulty of access to transport and how interested they are in politics.

It may be the case that those who find it difficult to access transport are more inclined to be interested in and tweet about politics, as they want to be able to have their voice heard about the troubles they are facing. They might hope for political advancements which would help them gain access to better transport. In contrast, for the people who already have good access to transport, this means one less reason for them to be politically motivated, and hence they have less reason to be interested in politics.

Should this hypothesis be correct, we would expect to see in our results a greater number of tweets relating to politics in areas which the Aurin data indicates a higher proportion of people who claim to have troubles accessing transport, and vice versa.

6.2 Comparison of income and happiness

This scenario looks to compare how much money is earned on average per taxpayer in each SLA, and to find correlations between this and tweet sentiment, which is considered a measure of the happiness of the person who made the tweet.

Whether money is able to 'buy' happiness is something which people have always wondered (and had differing opinions on). Our data analysis will allow us to see whether or not there is in fact any correlation between income and happiness in the areas we are analysing.

Certainly, it wouldn't be expected that the correlations we see here necessarily apply in general. For example, what you can buy with money differs not only globally, but even in different areas of Australia. It is certainly possible that money would have more of a direct impact on happiness depending on the area in question.

6.3 Comparison of visits to green spaces (once per week) and interest in barbeques and parties

In this scenario, we are looking at whether there is a correlation between people who visit green spaces once per week, and people who tweet about barbeques and parties. It is interesting because by analysing the correlation between these two events we then can conclude whether people prefer to party and have barbeques at home, or if they prefer to have barbeques outside in a green space. Also it allows us to reveal what people actually do when they visit a green space.

It is also possible that there is no correlation between these two events at all since visiting a green space doesn't necessarily imply having a barbeque. We will find out by finding people that post tweets containing the following keywords: "barbeque, bbq, party", and then we determine the SLA of the location where those people posted the tweets, and compare it against the data we downloaded from Aurin to see if there exists any meaningful correlation.

6.4 Comparison of unemployment rate and interest in AFL

In this scenario, we are looking at the number of people who tweet about the Australian Football League (AFL), and the number of people who are unemployed in each SLA. In order to do this, we have simply searched our tweets for the keyword ‘afl’, and read from the record which SLA that particular tweet was written from.

There are legitimate reasons why unemployed people would, or would not, be interested in AFL. One line of thought goes that unemployed people ‘should’ be trying to look for work, and thus would not have the time to watch or tweet about the AFL. It may also be the case that areas with high employment are generally less wealthy overall, and thus AFL is less accessible to them (television, tickets to game, etc.)

However, you could also argue that looking for work does not take as much time as a full-time job does. In addition, it might be that in areas with high unemployment, there are more people who do not put in any effort into finding a job, and have a lot of free time. This could lead to them being more inclined to be interested in watching the AFL.

7 Results

7.1 Comparison of trouble with transport and interest in politics

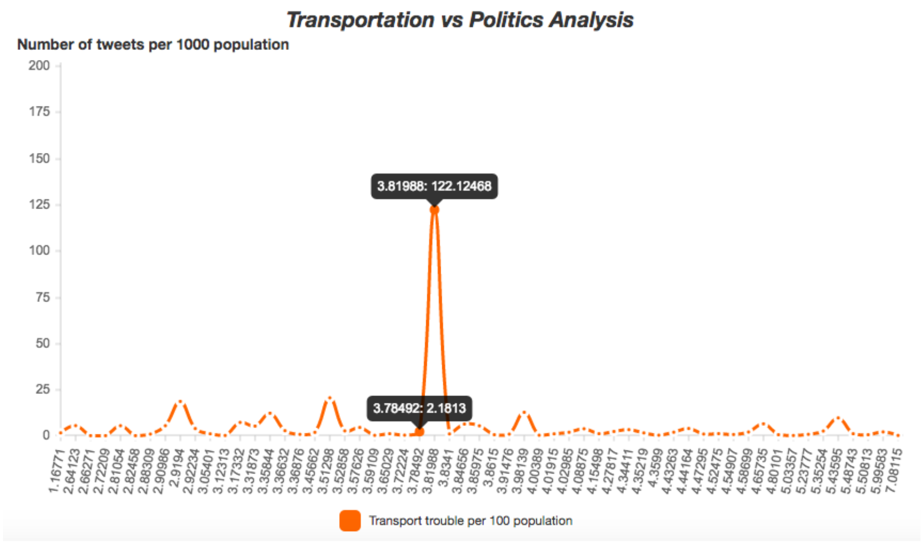

Figure 7: Graph showing trouble with transport against interest in politics in Sydney

We can see from Figure 7 that the number of people who were reported as having trouble with transport in their respective SLAs ranged from about 1 per 100 population to 7 per 100 population. We can observe that, in general, the number of tweets relating to politics at either extreme end of this scale were much lower than the SLAs which fell into the middle range of this scale.

For SLA with very low levels of transport trouble, the result appear to match what we expected, which was that we see a lower number of tweets about politics in these SLAs. Because there is less to complain about in that area, people have less reason to tweet about politics.

The figure also shows a steady increase in the number of tweets as the proportion of people claiming transport trouble increases, also as predicted in the initial scenario analysis. There appears to be a sharp increase at transport trouble per 100 population = approximately 3.8. This could be an outlier in our data, and could be a result of the same person repeatedly tweeting about politics leading to unusually large volume of tweets for that area.

There does, however, seem to be a decrease in tweet volume again, as we approach the higher end of transport trouble values. A possible interpretation of this result could be that, the increased number of people who believe there is a problem with transport in that area would lead us to believe that there really are major transport issues in that area. Perhaps transport is in such a bad state in those areas, or has been in a bad state for so long, that people do not feel that politics is the right avenue to solve the transportation issue (or that there is any avenue at all to solve the problem).

7.2 Comparison of income and happiness

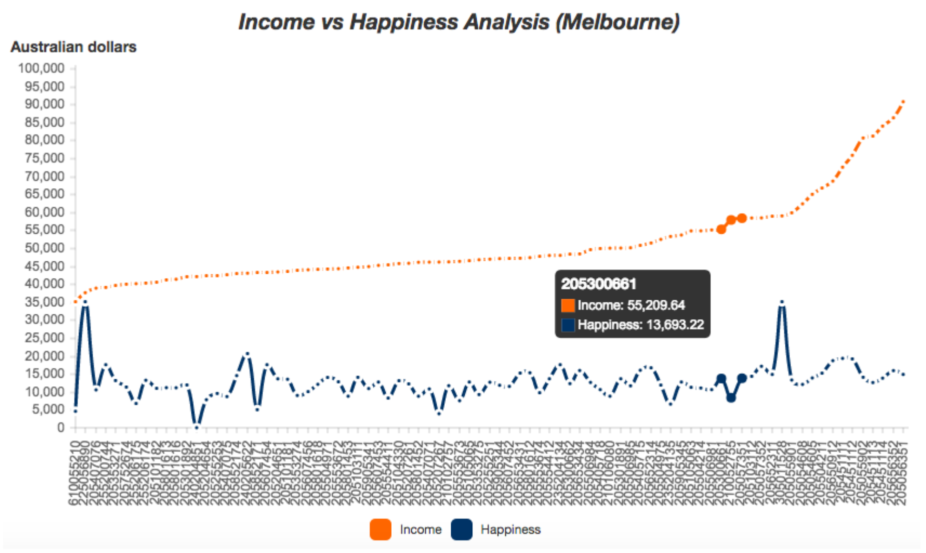

Figure 8: Graph showing income against happiness in Melbourne

As our enduring belief that if we just do the right thing, happiness will follow. Happiness is not a reward, but it is a consequence. Can money buy happiness? This question goes back centuries where many researchers have studied the topic and different researchers might have different say about it.

In the modern and more connected world, people use Twitter to express their feelings everyday, including their opinions on the surroundings, on events, and on other people. These feelings and opinions constitute our resources in our research. Proper sentiment analysis is performed on the tweets data of each SLA or Statistics Local Area to analyse if the tweet have expressed positive or negative or neutral sentiment in the SLA. And then sentiment analysis will help to generate a measurement of level of happiness in each SLA. Level of happiness for each SLA will be a percentage, calculated as the total number of positive sentiment divided by the total number of tweets in each SLA. The level of happiness will be amplified by timing the lowest income value in each SLA to make it comparable on the graphs.

From Figure 8 and Figure 9, as income increase dramatically, however, happiness doesn't follow. There is no clear indication that money can really buy happiness and we see this in two different cities, i.e. both Melbourne and Sydney. Melbourne people might be slightly more sensitive to income change versus Sydney people doesn't fluctuate at all. This shows another aspect that different cities in different countries can have different views on money, although we can still conclude that money is not positively correlated to happiness in our research.

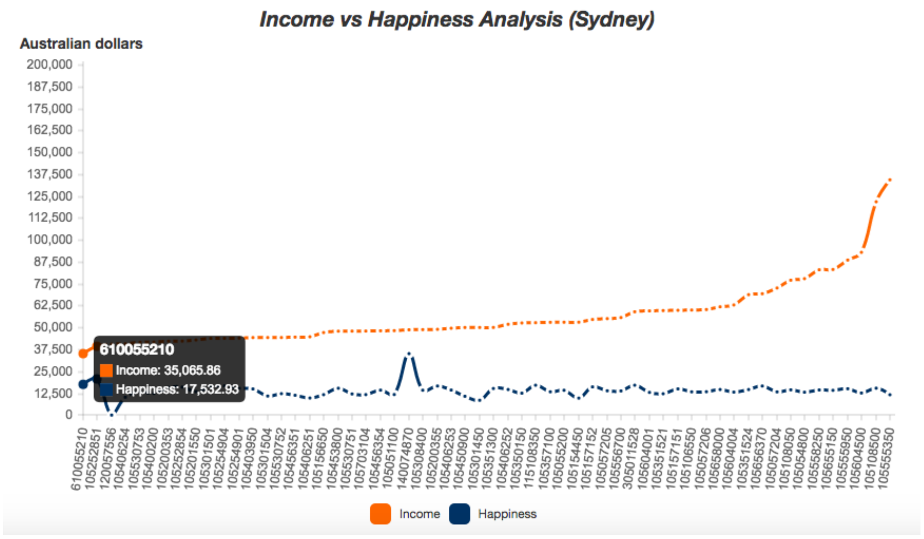

Figure 9: Graph showing income against happiness in Sydney

7.3 Comparison of visits to green spaces (once per week) and interest in barbeques and parties

Below is a plotted graph with x-axis being the estimate value of people visiting the green space(once per week) and y-axis being the number of tweets about barbeque and party within a SLA area. From the graph, we can sort of see the correlation here that it's not very common for people to tweet about having a barbeque and then visit a green place. Although there is one exception in which people tweet a lot and visit green place a lot, this could happen when there is

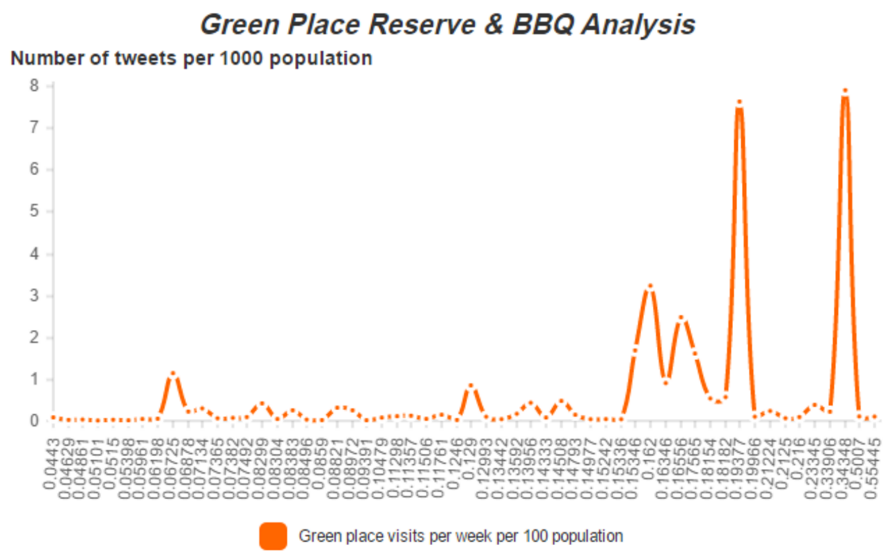

Figure 10: Graph of Green place visits against tweets about barbeques/parties

a nice green place in that area which people talk about a lot and go there a lot for barbeques and parties.

But in general, according to our results, areas where people visit green places a lot do not go there for barbeques, whereas areas where less people visit green places tend to have barbeques when they do visit.

7.4 Comparison of unemployment rate and interest in AFL

From the Figure 11, it is obvious that areas with higher unemployment show more interest in AFL than areas where with lower unemployment. It is not a surprise to see this result because surely people that are unemployed will have more spare time to do things they like, for example watching AFL.

There is a small peak in the less unemployed area. It could be that this particular area happens to have a strong affiliation with a particular AFL club, or it could be due to the fact that a lot of the tweets we have harvest may come from the same person, since we used search API to crawl the timelines of tweeters we already had in our databases. This may have caused us to harvest a large volume of AFL-related tweets from the same person, which can be misleading and suggest that there are a lot of people tweeting on this topic when in fact there isn't.

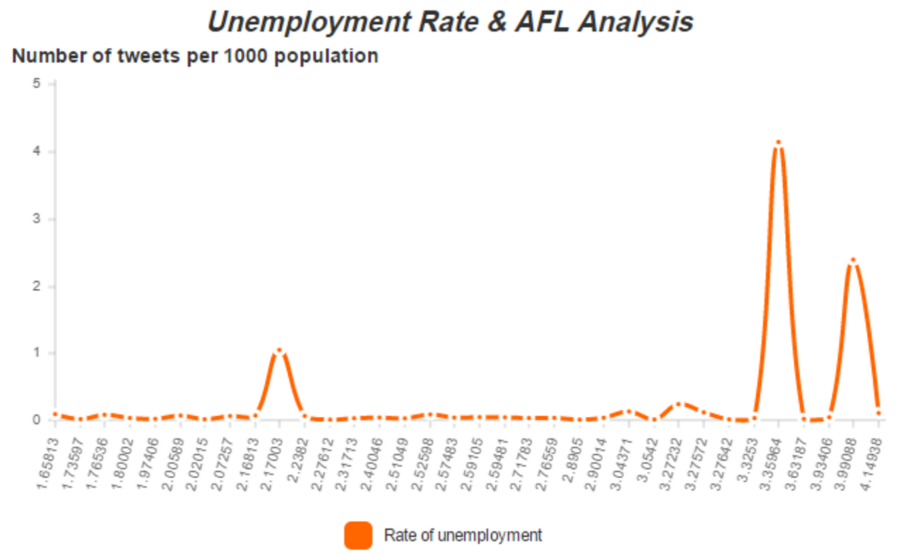

Figure 11: Graph of unemployment against interest in AFL

8 Discussion of NeCTAR Research Cloud and Other Tools/Processes

As a cloud service, NeCTAR shares many of same the advantages and disadvantages that other common cloud services do.

First of all NeCTAR is a wonderful service that provides all the necessary functionality for the project. The main functionality of NeCTAR is its SaAS service (which is also known as the Nova Computing Service). We used this service to create 4 VMs, 1 VM for CouchDB and hosting our django web application. The other 3 VMs for Twitter Mining. All 4 VMs have docker engine installed and all our application are containerized, published and deployed as docker instance to these VMs. The cloud storage (drive) that NecTAR provides (known as the Cinder Volume service). Helped us to extend our available disk space for CouchDB from 30GB to 250GB. This is very handy and allowed us not to worry about disk limitations for our database. The flexibility of this service also allowed us to transfer data across VMs or creating a shared mounted drive on multiple VMs. Object Storage (Known as the Swift Object Storage service) is an interesting one. Although we didn't end up using this service, but this service mimics the Amazon S3 Buckets. Which is optimized for storing and retrieving data in parallel. Dropbox is known to have been dependant on this service for its services. Before it got its own data centers and migrated its data. The security groups service (the Nova Security Group Service, but could be highly likely replaced by Neutron Security Group Service in the future) helped us to managed our firewall settings easier and faster through the Horizon Dashboard. This allowed us not to worry about learning VM level firewall such as ufw, firewalld, iptables and etc. This simplifies the required settings of all our

VMs.

However NeCTAR does have its downside. One of the risks involved with cloud services is downtime. If the cloud service being used experiences unusually high load, it can be rendered unreachable or unusable for unpredictably long amounts of time. When this happens, it can be difficult to continue working on the system if there is no way to obtain a local copy of the work. Related to this risk is that if you do not have network access, it will be hard to access any data or services located on the cloud. In comparison to other service providers such as AWS, Digital Ocean and etc, Although NeCTAR infrastructures is well designed to be fault tolerant, reliable and performance. NeCTAR does not share the same level of pressure to having to maintain its infrastructure to the same degree to keep its user's satisfied. In another word, Cloud Providers that depends on User's using its services to generate revenues, would need to aim for 0 tolerance to failure design (although this is not realistic).

Another disadvantage of using the NeCTAR cloud service is you do not have as much control over the infrastructure compared to if using your own machines (But this applies to all Cloud Provider services). For example, there were limits placed on the NeCTAR virtual machines (memory and storage), which we had no control over.

Overall we believe NeCTAR is definitely a very successful project and is backed by a very healthy opensource openstack community. Comparing to the past reliability and performance of NeCTAR. NeCTAR has definitely become more robust and performant. Some of us remember when we were using NeCTAR for the Distributed System subject. Often we couldn't create VM (even when we followed instructions correctly). And sometimes, the VMs (we created) fails from time to time. This year all our 4 VMs had flawless performance and none had any bugs or failures. We believe the more openstack projects matures itself, the more reliable and performant NeCTAR will be.

The tools we used to auto build, deploy and scale our application is very useful experiences for any future software engineering projects. First of all we used containerization technology to containerize all our system into docker images. We also hosted these docker images on docker hub, ready to be pulled for deployments anytime. All these processes are automated in gulp, which is a javascript framework. So we just had to script all our processes in a javascript file. And call the particular tasks we want to run from terminal. Lastly ansible is probably the most fascinating to learn and develop in this project. We wrote 3 ansible playbooks to auto manage and deploy our latest release to our VM environments. This saves the hassle of having to always do repeated deployments tasks and reduces the time to just a single command. We ended up to fully automate the deployment of latest CouchDB release to the CouchDB VM, our latest twitter mining code to all 4 VMs using the docker-py module and finally our visual web application (using the docker-py module as well) (This was achieved through by using gulp, ansible and git hooks technology together).

9 Conclusion

Through our tweet harvesting, data analysis and data visualisation, we were able to successfully see some correlations between our tweets and data that we retrieved from Aurin. The results did not necessarily always align with patterns that we had expected, but we were usually able come up with possible reasons for why this was the case, or if there were any oversights in our initial scenario analysis which could explain the discrepancy. Having said this, there were a few key points relating to the entire data retrieval and analysis procedures of our system which could have been improved, and we would certainly look to address these in any similar work in the future.

As part of our data analysis, it was necessary to determine whether tweets fell into a certain category, and whether they were relevant to our scenarios. In one of the scenarios, for example, we needed to determine whether tweets were related to the topic of politics. In order to achieve this, a list of words related to politics was compiled, and searched for in the tweets. This was almost certainly not an exhaustive list of words related to politics, and there would also be cases where tweets contain those words, but are in fact not related to politics at all. Introducing more advanced logic to determine whether a tweet fell under the topic of politics (or any given topic) would certainly be an area where the system could be improved.

Similarly, sentiment analysis of tweets could also be improved by introducing more advanced methods. For example, making use of more manually annotated lexicons, or introducing machine learning methods such as logistic regression and artificial neural networks. Such methods would likely require the use of a large volume of training data, which we would have to source from somewhere, or attempt to build ourselves. A major challenge of natural language processing, and hence sentiment analysis, is the detection of sarcasm. Our current system is unable to handle sarcasm very well, and indeed there are not many well-defined methods for handling sarcasm. Any solution capable of handling sarcasm would have to be capable of using contextual clues, not only in the tweet text, but possibly even in the metadata of the tweet.

The volume of tweets harvested prior to the beginning of our analysis work is also an area for improvement. Not only this, the two-pronged method used to harvest tweets (streaming API and search API) has a potential flaw, which is magnified with a smaller volume of tweets. The potential flaw is that all of the tweets gathered could be from a very small and selective group of tweeters. The reason for this is because the search API gathers more tweets from people who we already have in our database. With a small database, that means we are not getting a wide range of tweeters, and are instead getting an overrepresentation of certain tweeters in our database. While we were able to see some correlations and come to some conclusions from the tweets that we did have, a larger volume of tweets would have added an additional layer of believability and legitimacy to our results.

Another issue identified was the discrepancy between the dates of tweets and the dates associated with the Aurin data (2005-2006 or 2011 vs 2016). The data on Aurin was more of a guided number and any conclusion made from this outdated data has the potential to be irresponsible or inaccurate. Also the data was gathered mainly from surveys that were conducted on randomly selected people within that area, so it cannot be guaranteed that the data itself is a perfectly true reflection of the population. Moreover, not all people use Twitter, and for those people that do use Twitter, they do not necessarily tweet about what they are going to do or what their opinion of something is.

Taking all of these points into account, any analysis or conclusions arrived at from this data can be considered speculation for the most part, but still interesting and worth examining. A possible extension to this system would involve using the data gathered in conjunction with other sources of data or analysis to determine whether the conclusions we reached were a true representation of the population.

10 Youtube

`https://www.youtube.com/watch?v=-EhcNr9LZwI`

11 Reference

1. Politics Vocabulary. [`http://www.english-for-students.com/Politics-Vocabulary.html`]. Accessed 8 May 2016.
2. Hu M, Liu B. Mining Opinion Features in Customer Reviews. Proceedings of Nineteenth National Conference on Artificial Intelligence (AAAI-2004), San Jose, USA, July 2004.
3. REST APIs — Twitter Developers. [`https://dev.twitter.com/rest/public`]. Accessed 27 April 2016.
4. Docker Command Line Reference, [`https://docs.docker.com/engine/reference/commandline/cli/`]
5. Ansible Docker Reference. [`http://docs.ansible.com/ansible/docker_module.html`]
6. Automate your tasks easily with gulp.js. [`https://scotch.io/tutorials/automate-your-tasks-easily-with-gulp-js`]
7. Configure Apache using ansible [`https://www.digitalocean.com/community/tutorials/how-to-configure-apache-using-ansible-on-ubuntu-14-04`]
8. Ansible Apt Reference. [`http://docs.ansible.com/ansible/apt_module.html`]
9. Ansible Service Reference: [`http://docs.ansible.com/ansible/service_module.html`]
10. Yargs github library Reference: [`https://github.com/yargs/yargs`]

11. Gulp github library Reference: [https://github.com/gulpjs/gulp]

12. Gulp-shell github library Reference:
urlhttps://github.com/sun-zheng-an/gulp-shell]