Cluster and Cloud Computing Assignment 2 -Australian City Analytics

—Team10, Melbourne, Australia

Cong Yue, 682020 Lingjuan Lyu, 701197 Yun Wang, 672323 Mingyu Gao, 692634
Huabin Liu, 658274

I. Introduction

In this project, we analysed the tweets and Aurin data of Melbourne from 4 main aspects, namely, mining multiple patterns of Melbourne tweets regarding movie, mining the patterns between sentiment and household income, discussion of topic preference and active tweeters and language ratio in each suburb, and mining the time based sentiment pattern of melbourne twitters. In the first scenario, we analysed the attitude people hold towards some popular movies of 2015 and 2016, and investigated whether there is a relationship between tweets and IMDB rating. In the sentiment and household income scenario, we analysed whether the sentiment score varies with their income level, and discovers the sentiment score distribution of different suburbs. In the topic preference and active tweeters, the top topics and tweeters in each suburb is visualised, along with the language distribution of the corresponding suburb. In the final scenario, we analysed the relationship between sentiment and time, to see on which day people in Melbourne are happier, and if peoples feelings change during the day. We developed our application on NeCTAR Research Cloud for supporting large data harvesting, storing, processing and presenting. The Nectar Research Cloud provides computing infrastructure, software and services that allow Australias research community to access and share computational models, tools, data and collaboration environments (http://cloud.nectar.org.au/). On NeCTAR Research Cloud, we achieved automatically exploiting 4 VM instances, setting up CouchDB and building our Website. For harvesting as much related tweets as possible, both Twitter Stream API and Twitter Search REST API are being used. In this project, geolocation is set to restrict collecting area as Melbourne. In addition to the attributes that Twitter API provided, some useful extra tags were added as well for the following scenarios analysis. The data of our project are all stored in CouchDB. As a Document-oriented DBMS, CouchDB allows users to store data in JSON formats and conveniently query the data with a web browser. CouchDB supports on-the-fly document transformation and real-time change notification, which make app development much more easier for both enterprises or individual users. We use Ansible to install our working environment and to deploy our software remotely. Ansible is a software platform that perform cross-node software management and configuration management. With Ansible plus SSH tools, engineers can easily copy a whole software and paste onto another node, no matter on Local PC or Cloud.

The front-end visualisation is achieved with Highcharts and Bootstrap. Highcharts is a simple but flexible chart API for data analysts. It not only provides users with well-designed charts which can be easily applied to different browsers, but also allows rich customisation to meet customers various needs. Bootstrap is another handy library that allow users to easily adjust the layout of page.

II. Architecture

Generally, we deploy CouchDB[1] as well as web server on the Nectar Cloud[2]. In CouchDB, we stores two types of data with respect to sources: one is from Internet including Twitter[3], Aurin[4] and IMDB[5] movie ratings; another source is from our analysis, we store the results of our analysis in CouchDB, then web server can fetch those data and display them in an appropriate structure on website. On analysis aspect of the application, we use python and CouchDB embedded Map-Reduce methods to implement analytic logic. On website aspect of the application, we use package bottle and bootstrap to display results in forms of charts and images. The big picture of our design is shown in Figure 1.

III. The architecture design of tweet harvest application

A. Twitter Search REST API

Request sent to Twitter API must be valid and authorised. Search API method supports both application-user authentication and application-only authentication (http://www.karambelkar.info/2015/01/how-to-use-twitters-search-rest-api-most-effectively./). Instead of makes requests on behalf of a particular user, application-only authentication authenticates requests on its own behalf and rate limits are determined globally for the entire application. More precisely, Search API with application-only authentication, which is applied in this project, gives up to 450 calls per per 15 minute window, and 2.5 times higher than application-user authentication.

[1]http://couchdb.apache.org/

[2]nectar.org.au/research-cloud

[3]twitter.com

[4]http://aurin.org.au/

[5]http://www.imdb.com/

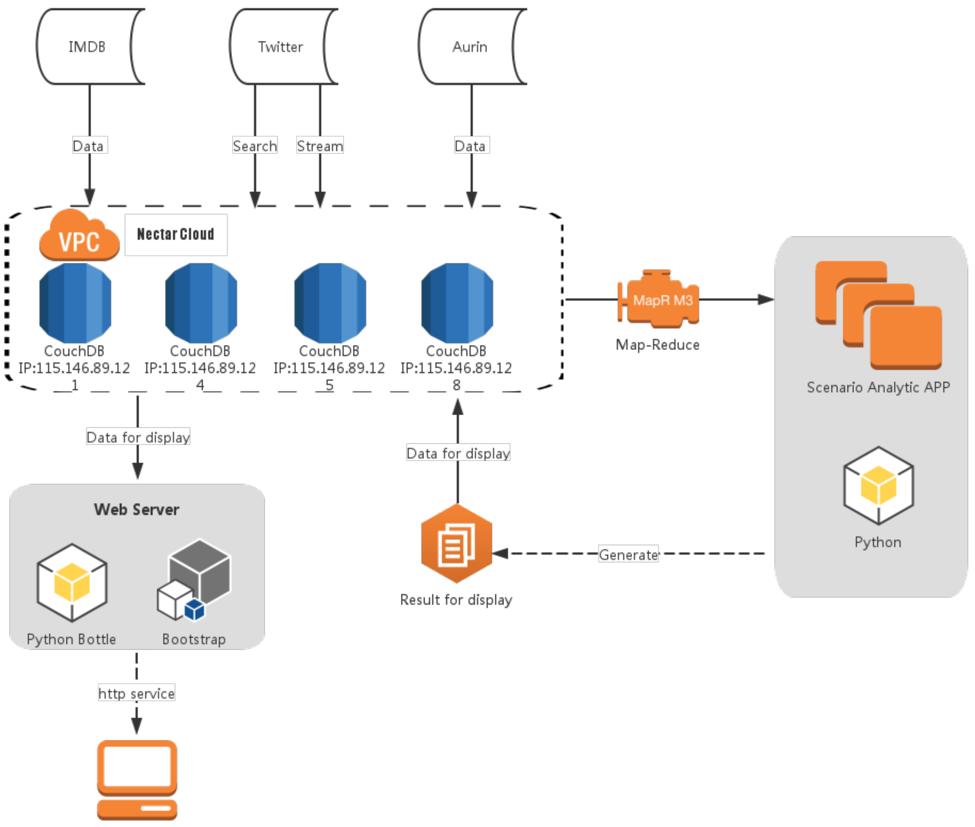


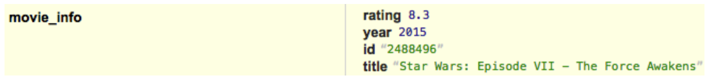
Fig. 1: The architecture design of tweet harvest application.

As an important part of Twitter's REST API, the GET search/tweets provides programmatic method to harvest historical tweets in terms of relevance. Not all tweets are available via this interface, the returned result only contains matched tweets published within the latest week. And a set of parameters, e.g. geolocalisation, can be added to have a better control of the results (https://dev.twitter.com/rest/public).

Search API search against on-going and previous context for a given search term. The search query string q is keyword(s) from title of movie, e.g.'Star Wars VII OR Force Awakens' for Star Wars: Episode VII - The Force Awakens. And as our research focuses on melbourne, geocode is set as a round area with -37.814396 for latitude, 144.963616 for longitude and 40km for radius. Language of text is restricted to english and each returned page contains 100 tweets.

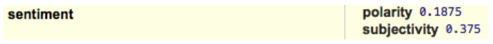
Information of a tweet is obtained in format of JSON. Before saving it into CouchDB as a document, two additional fields are needed to add. The polarity and polarity score for sentiment field is gained by processing text of tweet using a third-party package called TextBlob. And movie_info field contains IMDb id, movie title, IMDb rating and release year. The detailed information is shown in Figure 2 and

Figure 3. Each tweets document has the identical id as that of Twitter.



```
movie_info      rating 8.3
                year 2015
                id "2488496"
                title "Star Wars: Episode VII - The Force Awakens"
```

Fig. 2: Movie information.



```
sentiment       polarity 0.1875
                subjectivity 0.375
```

Fig. 3: Movie sentiment.

B. Twitter Streaming API

Differ from Twitter Search REST API, Twitter Streaming API need a persistent HTTP connection open all the way for receiving new coming tweets. Once client makes an initial call to the Streaming API, server will not respond immediately. Instead, server will monitoring data on the stream and until when there is new data available, it will push the data back to client. So in this case, the data received through Streaming API is always the newer one. Typically, there is a process handles HTTP requests while another process maintains the Streaming connection.

Moreover, the streaming process gets the input Tweets and performs any parsing, filtering, and/or aggregation needed before storing the result to a data store. The HTTP handling process queries the data store for results in response to user requests (https://dev.twitter.com/streaming/overview). Also, Streaming API need authentication and each account can only create one connection to public endpoints. However, what makes it so convenient is that there is no need to consider the rate limit at all and it reduces the network latency significantly.

In this project, we used the public stream which contains public data flowing through Twitter. We used tweepy stream for Python for codes programming. In the detail, we passed the bounding box of Melbourne parameter as the filter into tweepy stream to limit only harvesting those tweets from Melbourne with the following command:

myStream.filterlocation=[144.593742,-38.433859,145.512529,-37.511274],async=True

However, this filter may be unstable since it sometimes return some tweets outside the Melbourne as well. So to be more accurate, in the stream listener, we add a condition to check whether each tweet's place full name is 'Melbourne, Victoria' or not. Also note that we only seeking for those tweets that have names, profile images, texts and places.

Furthermore, in order to do the following scenario analysis, before storing data into Couchdb, we add two tags for each tweet. One is the sentiment score of each tweet text detected via Textblob Python package. Another one is the suburb name and postcode gained via Google Maps API for the tweet that has specific location coordinates. Each tweets document stored in the Couchdb has their identical id as that of Twitter.

C. Aurin

Aurin provides a series of datasets developed and contributed by Australia's leading researchers (http://aurin.org.au/). The datasets consist of several area ranges from part small areas of suburbs to states in Australia. It supports displaying the datasets on the particular range map as well as downloading as csv, json or shp file. More attractive, it has many analysis tools to help us analyse datasets more convenient and deeper. In this project, we restrict the range to be Greater Melbourne (shown in Figure 4), that is, containing all the suburbs in Melbourne.

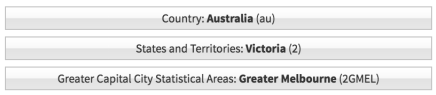
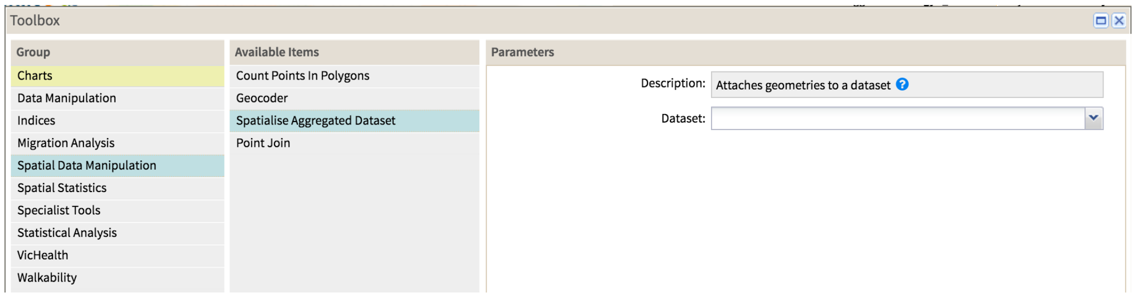

Country: **Australia** (au)
States and Territories: **Victoria** (2)
Greater Capital City Statistical Areas: **Greater Melbourne** (2GMEL)

Fig. 4: Aurin range restriction.

To deal with the dataset, we used the spatial data manipulation tool (Figure 5) to aggregate dataset with spatial information for our scenarios analysis based on suburbs.



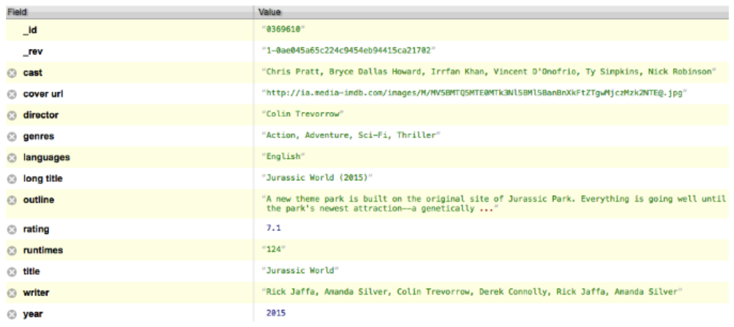
Description: Attaches geometries to a dataset
Dataset:

Fig. 5: Aggregate dataset with spatial information.

D. IMDb

The Internet Movie Database (abbreviated IMDb) is the world's most popular and authoritative source for movie, TV and celebrity content. A third-party package IMDbPY is used to retrieve and manage the useful data. IMDb function returns an imdb object of IMDbBase class to access the IMDb data. For this project, search_movie(title) interface and get_movie(movieID) interface of imdb object are called. And a Movie object with a dictionary-like interface is returned when query for a particular movie.

Method get_movie(movieID) returns a Movie object. And the information of a movie can be accessed by the dictionary-like interface of Movie class. For this project, each movie document in CouchDB shares the same id as IMDb. Keys of Movie object, for instance title, rating, genres, year, outline, director, cast, cover url, are restored as fields, as shown in Figure 6.



Field	Value
_id	"0369610"
_rev	"1-0ae045a65c224c9454eb94415ca21702"
cast	"Chris Pratt, Bryce Dallas Howard, Irrfan Khan, Vincent D'Onofrio, Ty Simpkins, Nick Robinson"
cover url	"http://ia.media-imdb.com/images/M/MV5BMTQ5MTE0MTk3Nl5BMl5BanBnXkFtZTgwMjczMzk2NTE@.jpg"
director	"Colin Trevorrow"
genres	"Action, Adventure, Sci-Fi, Thriller"
languages	"English"
long title	"Jurassic World (2015)"
outline	"A new theme park is built on the original site of Jurassic Park. Everything is going well until the park's newest attraction––a genetically ..."
rating	7.1
runtimes	"124"
title	"Jurassic World"
writer	"Rick Jaffa, Amanda Silver, Colin Trevorrow, Derek Connolly, Rick Jaffa, Amanda Silver"
year	2015

Fig. 6: Movie document.

IV. The architecture design of different analytic application

A. *The design of analytic scenario 1*

Scenario 1: Mining multiple patterns of Melbourne tweets regarding movie

Scenario 1 focuses on both movie based and rating based analytics. And the core component is data analysis. As the UML class diagram shown in Figure 6, the are two main provided interfaces for data analysis – rate() and movie(). For various ranges of rating, method rate() provides analysis in terms of tweets per movie and average sentiment value. And for all target movies, method movie() supports analysis in terms of number of relevant tweets and average sentiment value. The implementation of both two methods rely on the build-in Map/Reduce functions in CouchDB.

In terms of choosing a suitable set of target movies, first, the IMDb most popular movie list is a significant reference. The unreleased movies in this list are ignored. And movies with ’tricky’ name are excluded as well. These ’tricky’ names are tend to be short and composed of high frequency words. More particularly, for example, there is a movie called Room. When making a query for keywords ’Room’, most tweets returned are not actually related to movie Room and yield a huge amount of noise for later analysis. 25 sample movies are chosen after this step. Then testing queries are made to Twitter API.

Deviations exists between popular movie list and results from Melbourne movie tweets within the last 7 days. A movie should also be filtered if the matched tweets is less than 1% of total movie tweets, because the lack of tweets would reduce the accuracy when calculating average sentiment score. Finally, 18 movies are formally taken into account for the later tweet harvesting and data analysing.

The tweets analysed in this project are all posted in 2016 and at Great Melbourne area. The following pie graph in Figure 7 shows the percentage of tweets regarding each movie. Movies released at this year trigger more discussion than these of 2015. Captain America: Civil War, Star Wars: Episode VII - The Force Awakens, Deadpool and Batman v Superman: Dawn of Justice are the top 4 most popular movie with share of tweets above 10%. From the respective of genre, all the these popular movies share the same tags as Action, Adventure and Sci-Fi.

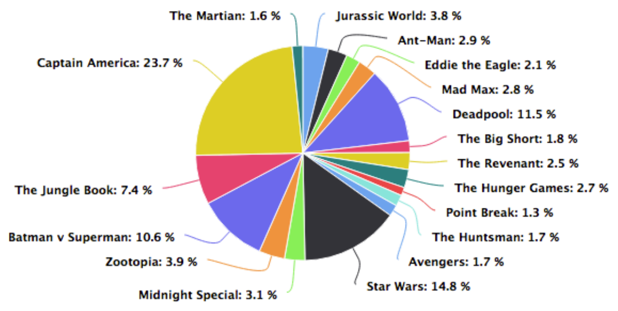


Fig. 7: Percentage of tweets regarding for movies.

The following bar chart in Figure 8 illustrates the average sentiment level of each movie based on related Melbourne tweets. It is apparent that people only have negative feelings towards Mad Max: Fury Road. And from all movies with positive attitude, the polarity score of Eddie the Eagle and Midnight Special are considerably more higher than that of the other movie.

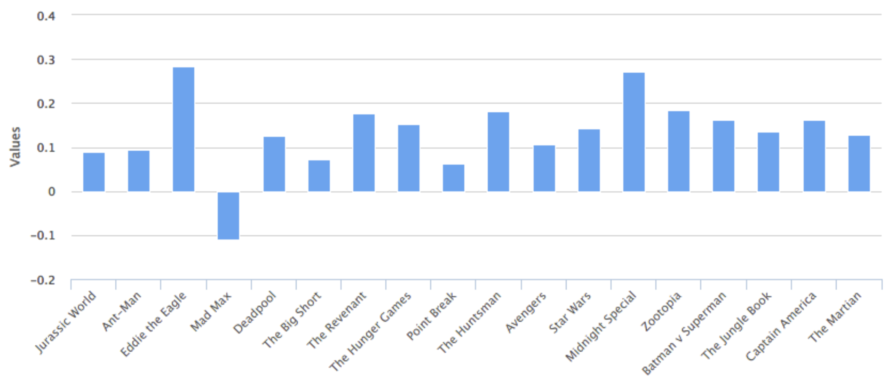


Fig. 8: Average sentiment level for movies.

The diagram in Figure 9 reveals the difference among different ranges of movie rating in terms of average sentiment score and the number of tweets per movie. As the 18 target movies are not evenly belongs to the rating interval, simply comparing the number of movie or tweets does not make any sense, therefore tweets per movie is calculated. And form the tweets gathered, the popularity of movie on Twitter and the rating of movie see a same trend. Which means a higher quality movie gains more user attentions according to the data from social network. However, the sentiment towards movie are not closed related to its rating.

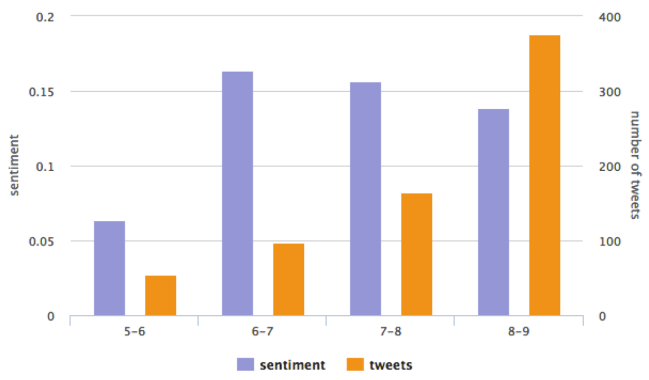
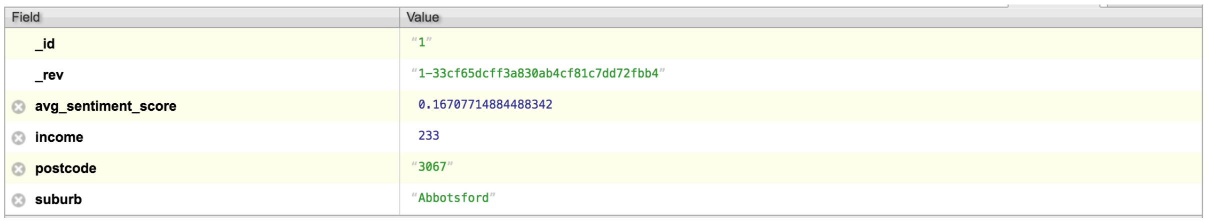


Fig. 9: Difference among different ranges of movie rating in terms of average sentiment score and the number of tweets per movie.

B. *The design of analytic scenario 2*

Scenario 2: Mining the patterns between sentiment and household income

In this project, the task of tweets analysis for each suburb is necessary. For the second scenario, we focus on three dimensions in total: suburbs in Melbourne, income of each suburb and the sentiment of each suburb. The main idea of this is to find out whether a suburb with more income per week will be more happy than the lower ones. Therefore, we not only harvested those tweets with specific suburb tags we added ourselves by using Google Maps API, but also downloaded the recent income data of every suburb within Melbourne from Aurin. For every tweet we collected, before storing in our Couchdb, we also added the sentiment score tag by using Textblob Python package. To create a database that contains data if and only if they have suburb tags, we transferred the data that have suburb tags from the original ’melbourne tweets’ database into a new database named ’has_suburb_tweet’ by applying cURL utility with a filter in Couchdb design document. And then in the new database, we gained average sentiment score of each suburb by implementing Map/Reduce functions in Couchdb. Finally, the average sentiment scores and incomes were aggregated according to the suburb names and postcodes storing in the Couchdb, as illustrated in Figure 10.



Field	Value
_id	"1"
_rev	"1-33cf65dcff3a830ab4cf81c7dd72fbb4"
avg_sentiment_score	0.16707714884488342
income	233
postcode	"3067"
suburb	"Abbotsford"

Fig. 10: tweet example with suburb tag.

In this scenario, we aimed for exploring the sentiment of people with relatively high household income level separated by suburbs in Greater Melbourne. Thus, from Aurin, we downloaded the datasets of total household income (weekly) by household composition in 2011 where the incomes were set as two level: '$3500-$3999' and '$4000 or more'. Then we added them up to make up a dataset presenting the amount of household of every suburb with more than $3500 income per week shown as the blue line in Figure 11.

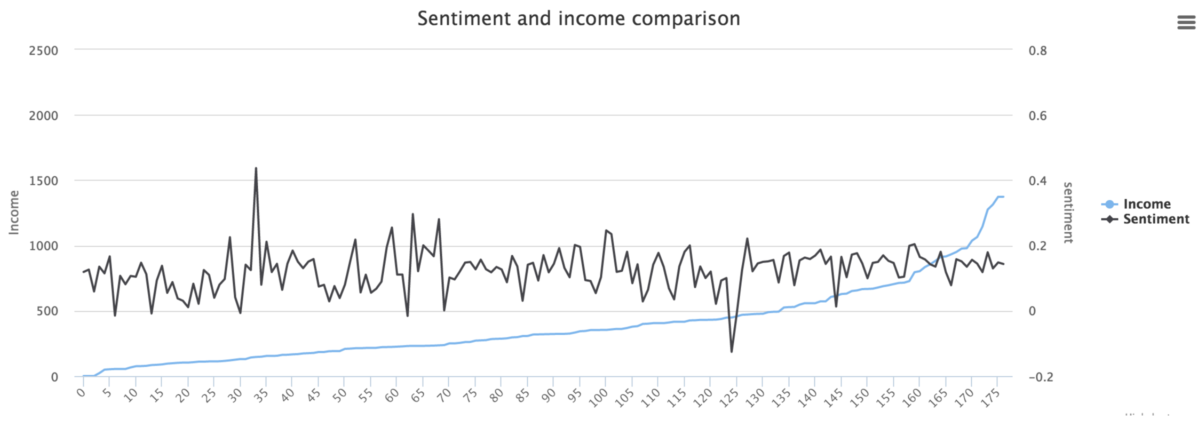


Fig. 11: Sentiment vs income.

Then, to generate the average sentiment scores of each suburb, we used Map/Reduce functions built-in Couchdb. The map function returns the key as [suburb name, postcode]. In this step, we just considered those suburbs that have more than 16 tweets overall. This is because that those which have less than 16 tweets may be wrong combinations of suburb name and postcode owing to the error of Google Maps API. Even though it is right, so small data size makes no sense at all. Finally, according to each point of suburb on the blue line, we draw another line to represent the average sentiment score varying trend while amount of household with high income growing up shown as the black line in Figure 11. This seems the sentiments just fluctuate randomly and there is no relationship between sentiment and income as the first glance. However, when we went deeper, we found something really interesting. First, the most happy suburb is 'Noble Park' while the most unhappy suburb is 'Kensington'. But the income level of 'Noble Park' is far below the 'Kensington'. So does that mean higher income level makes people feel unhappier? No. These two particular suburbs may perform like this owing to their own suburb environments since the former part of line is very irregular. When we looked at the latter part of line, after income level reaches 628 (after No.145 suburb in the Figure 11), although the black line still fluctuates in a small range, it maintains a relatively high positive average sentiment level. So this may represent that people in those high income level suburbs are commonly happier.

There is another image we generated showing the whole Greater Melbourne sentiment picture as below (Figure 12). Note that since we get rid of those suburbs with small data size, there are many blank areas in the map. But on another perspective, this indicates that people in these areas, such as 'Brooklyn' and 'Mount Waverley', are perhaps not likely to use Twitter. In this map, the main color is green indicating much more people in Greater Melbourne are happy. And while 'Noble Park' becomes the champion of positive sentiment, 'Kensington' (the red area in the central of map) becomes the most negative suburb as mentioned previously.

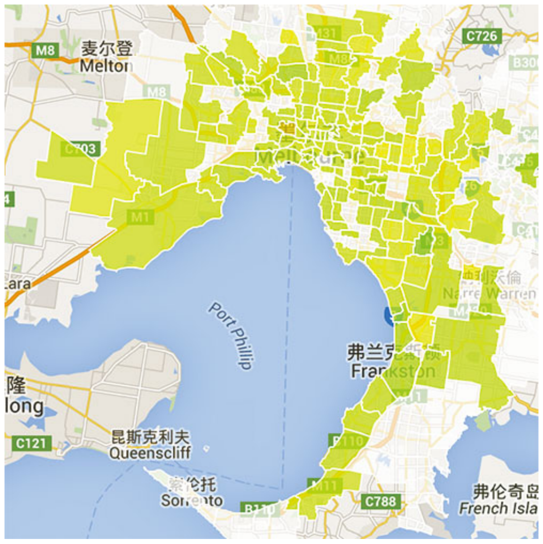


Fig. 12: Greater Melbourne sentiment.

C. *The design of analytic scenario 3*

Scenario 3: Mining the topic preference and active twitters, language ratio in each suburb

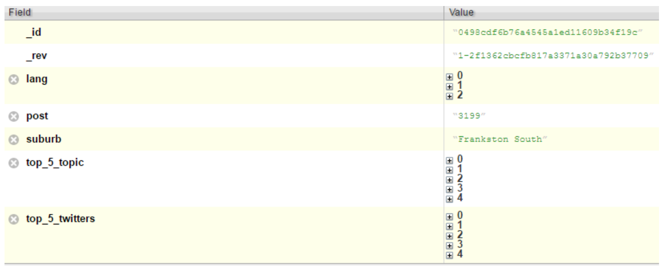
In this scenario, the tweets analysis for each suburb continues. The fine-grained analysis result based on postcode can provide valuable feedback for government planning and social investigation. Thus, we analyze the tweets from Melbourne tweet data from the following three perspectives: top5 topics, top5 twitters and language ratio. The intention for this task focus on analyzing raw tweet data and infer useful information from the results. Our raw data comes from the database named 'has_suburb_tweet' from 115.146.89.121 as introduced before. We experiment our analysis on all suburbs tweet to provide an overall and detailed statistics.

top5 topics: represent what kind of topics people are interested in during that period of time in the specific suburb. The number of times these topics have been tweeted are also recorded, from which we can figure out whether people in that suburb would like to tweet a lot of times or not.

top5 twitters: represent the popular twitters that is mentioned the most during that period of time in the specific suburb. The top five twitters are ranked by the number of times their twitter usernames occur in the couchdb.

language ratio: represent the language component ratio, from which we can infer the population component ratio in each suburb. A collection for each document contains mainly five fileds shown in Figure 13:

- *language ratio in each suburb*
- *suburb postcode*
- *suburb name*
- *top5 topics*
- *top5 twitters*



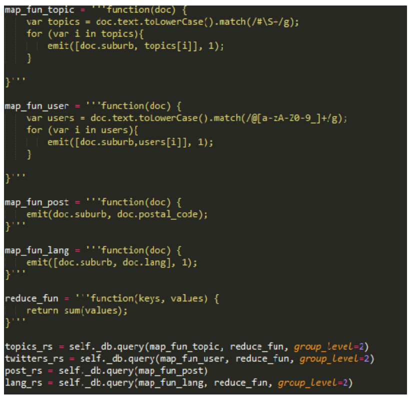
Field	Value
_id	"0498cdf6b76a4545a1ed11609b34f19c"
_rev	"1-2f1362cbcfb817a3371a30a792b37709"
lang	⊞ 0 ⊞ 1 ⊞ 2
post	"3199"
suburb	"Frankston South"
top_5_topic	⊞ 0 ⊞ 1 ⊞ 2 ⊞ 3 ⊞ 4
top_5_twitters	⊞ 0 ⊞ 1 ⊞ 2 ⊞ 3 ⊞ 4

Fig. 13: Topic twitter language CouchDB.

In this section, MapReduce [1] method in the couchDB has been applied to help analyze tweets for all suburbs. After successfully connected with couchdb server, we process tweet and extract the necessary parts according to the format provided by the database. As topics and twitters rely on analyzing the "text" of the tweet, regular expression is adopted to match "#" for topics and "@" for twitters. For language, we pass the ”lang” to language type. We use postcode to distinguish suburbs and emit the suburb name as key[0], topics/twitters/lang in that suburb as key[1]. The next step is to count their values, namely number of topics, twitters and language usage for all suburbs, then sort them by decreasing order and list language usage ratio and top5 popular topics and twitters. We use ”if” condition statement to judge whether a topic/twitter/lang appears before or not. If it does not exist yet, we put it into this array as a key and take ”1” as its initial count value. If it does, we add 1 to its corresponding count value. As couchdb uses JSON to store documents, JavaScript as its query language to transform the documents and query the indices with the web browser, the procedure is realized by MapReduce method based on couchdb query mechanism. The implicitly available emit(key, value) function is called and every invocation of that function, a result row is added to the view. If the tweets match one of keywords, it will be mapped with key which is formed by a suburb name and corresponding topic/ twitter/lang and take ”1” as value. Take topic as an example, the defaultdict is constructed as:

(’key’: [’ suburb’, ’topic’], value: 1)

Then ”reduce” the sum result to get the final statistic, the procedure is shown as shown in Figure 14.

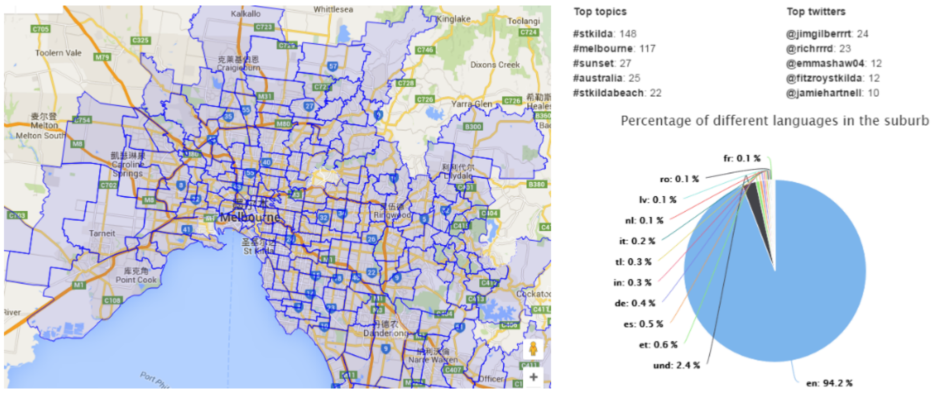


```
map_fun_topic = '''function(doc) {
    var topics = doc.text.toLowerCase().match(/#\S+/g);
    for (var i in topics){
        emit([doc.suburb, topics[i]], 1);
    }

}'''

map_fun_user = '''function(doc) {
    var users = doc.text.toLowerCase().match(/@[a-zA-Z0-9_]+/g);
    for (var i in users){
        emit([doc.suburb,users[i]], 1);
    }

}'''

map_fun_post = '''function(doc) {
    emit(doc.suburb, doc.postal_code);
}'''

map_fun_lang = '''function(doc) {
    emit([doc.suburb, doc.lang], 1);
}'''

reduce_fun = '''function(keys, values) {
    return sum(values);
}'''

topics_rs = self._db.query(map_fun_topic, reduce_fun, group_level=2)
twitters_rs = self._db.query(map_fun_user, reduce_fun, group_level=2)
post_rs = self._db.query(map_fun_post)
lang_rs = self._db.query(map_fun_lang, reduce_fun, group_level=2)
```

Fig. 14: Map reduce procedure.

We query with group_level=2 to get a reduce query run for each unique set of keys, namely get the sum amount as value, the value is then correlated with the corresponding keys: suburb name and topic/twitter/lang, then save them in defaultdict. Finally, the suburb name, postcode and sorted top5 ranked topics and twitters and all language ratio for each suburb are saved as items in the form of documents and store it into the new couchdb. During implementation, database connection, creation and save, along with error handling which is supported with ”try” and ”except” clause is realized in topicV.py and then import it in the main code.

In the produced CouchDB, there is a system defined ”_id” as unique identifier for each doc and revision ”_rev” to implement the MVCC(multi-version concurrency control) to avoid the need to lock the database field during writes. This simplifies the database management. The final web page illustration for the above three scenarios is shown in Figure 15. After loading the Google maps, you can click on any suburb, the corresponding result will be displayed on the right. Take suburb Saint Kilda for example, as Saint Kilda is in Melbourne, Australia and is famous for its sunset and charming beach view, the top topics along with the related tweet number are all about these aspects, people in this suburb mention the twitter ”@jimgilberrrt” the most. The visualization for top5 topics and twitters is shown in Figure 16. From the language ratio, we can figure out Saint Kilda consists of lots of people who tweet with different languages, which implies that it has attracted people from all over the world to tweet its beautiful sight.



Top topics	Top twitters
#stkilda: 148	@jimgilberrrt: 24
#melbourne: 117	@richrrrd: 23
#sunset: 27	@emmashaw04: 12
#australia: 25	@fitzroystkilda: 12
#stkildabeach: 22	@jamiehartnell: 10

Fig. 15: Google map illustration and analyzing result for suburb Saint Kilda.

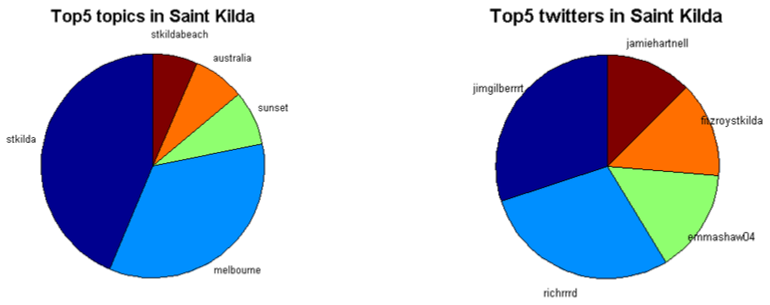


Fig. 16: Top5 topics and twitters in Saint Kilda.

For complexity evaluation, we compare the time spent on completing all the procedures with and without map reduce operation, Figure17 shows the time complexity and it is obviously faster when map-reduce is applied.

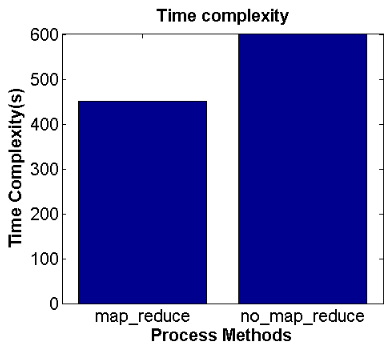


Fig. 17: Time complexity.

D. The design of analytic scenario 4

Scenario 4: Mining the time based sentiment pattern of Melbourne twitters

Scenario 4 focus on different time-slice based analytics. It contains 3 components:

- Data extraction component
- Analysis component
- Result saving component

Both data extraction component and result saving component has some same operation like connecting to couchdb. So the best practise is encapsulating those common operations and related attributes.

Considering the common database operation like ”save”, ”connect”, we build a base class to encapsulate these operations and related attributes, as well as error handling details inside the base class. The following is a UML class graph. Class ”TaskV” is our base class, ”Task” is the real execution class and will be instantiated when executed. The analytic logic is implemented in ”Task”. The UML class graph is displayed in Figure 18.

For the design of data structure in CouchDB, we build an IR application to extract useful key-value data that forms basis of our analysis and store them into CouchDB. The two base data collections are:

- *A collection (database in CouchDB) where each document contains three items:*
- Twitter (the user)
- Day of week that twitter tweets
- The sentiment score in a specific day of week with respect to a specific user. Here, for sentiment score, the positive value indicates the twitter prefer to post more positive tweet; the zero value indicates the twitter's attitude in his tweets is moderate; the negative value indicates the twitter prefer to express negative attitude in his tweets.
- *A collection where each document contains three items:*

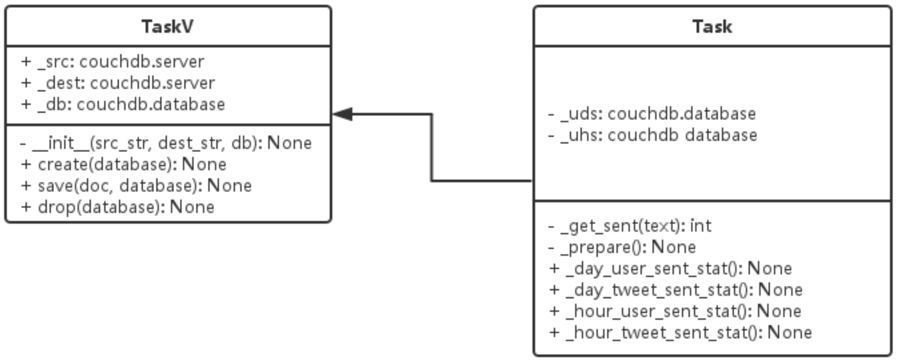
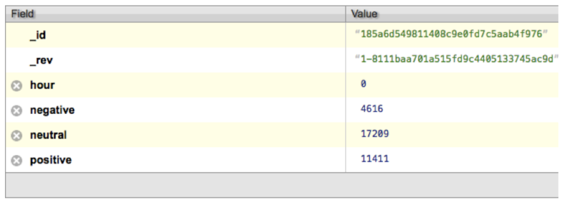


Fig. 18: Cloud UML.

- Twitter (the user)
- Hour in a day
- Sentiment score in a specific hour in a day with respect to each twitter.

Then, we write an analytic program that extract statistics of tweets with respect to ”day of week” and ”hour in a day” separately from the original harvest and store the result in CouchDB. The result collection contains 4 items:

- index of time slice (’day of week’ or ’hour in a day’)
- count of negative tweets
- count of positive tweets
- count of neutral tweets A snapshot of the result of in CouchDB is displayed in Figure 19.



Field	Value
_id	"185a6d549811408c9e0fd7c5aab4f976"
_rev	"1-8111baa701a515fd9c4405133745ac9d"
hour	0
negative	4616
neutral	17209
positive	11411

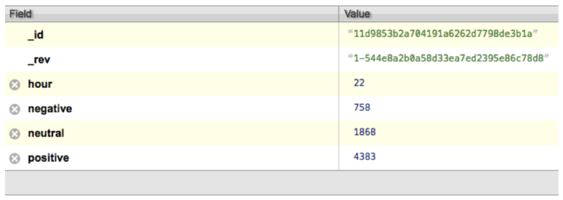
Fig. 19: tweet sentiment CouchDB.

We next extract statistics of tweets with respect to the same two criteria from our base collection discussed above and store the result in CouchDB. The result collection contains 4 items:

- Index of time slice
- Count of positive tweets

- Count of negative tweets
- Count of neutral tweets

A snapshot of the result in CouchDB and the corresponding statistics in website are displayed in Figure 20.



Field	Value
_id	"11d9853b2a704191a6262d7798de3b1a"
_rev	"1-544e8a2b0a58d33ea7ed2395e86c78d8"
hour	22
negative	758
neutral	1868
positive	4383

Fig. 20: User sentiment CouchDB.

Intuitively, we assume people's' behaviors are time sensitive, because our lives are composed of many rich things, we can only spend limited time in twitter. So we want to find the pattern with respect to time behind Melbournians' usage of twitter. As we only harvested tweets in a limited time span, so we choose to analyze based on two types of time slices the day of week, i.e. Sunday, Monday, etc. and 24 hour slices in a day.

There are two reasons that we choose the exact two types of time slices:

1. Both "the day of week" and "hour in a day" has the property of circulation in a larger span of time. So we can aggregate data into different time splice and detect the pattern.

2. Both "the day of week" and "hour in a day" is closely related to some unknown patterns according to human behavior. For example, most of people sleep at night and do nothing during that period; Most of people work from Monday to Friday when they may have less time to tweet.

Based on types of time slice we choose, we analyzed not only statistics of tweets, but also the statistics of users (twitters). Because our main target is analyzing the pattern behind Melbourne twitters more than tweets. To make the task more interesting, we introduced sentiment analysis, which means we want to find if Melbourne twitters have preference of expressing their mood on some specific time slice or a span of time slices.

Generally speaking, Melbournians are prefer to post positive tweets. Because positive tweets account for 55% of all, positive twitters account for 63% of all.

From Figure 21, we can tell that Melbournians prefer post most of their tweets during 18:00 to 22:00 and would not like to tweet after midnight. The trend of tweets of different sentiment nearly same, there is no much interesting to infer. From Figure 22, very unlike Figure 21. First of all, there is no obvious peak and valley for users with negative sentiment, which means there is no preference for most Melburnians to express their bad mood with respect to time. However, regarding to positive curve, we can tell most of Melburnians prefer to post their good mood during daytime than night time.

Also, from the comparison of Figure 21 and 22, we can infer only the former one can represent the ”mood” of Melbourne twitters. Because active twitters post far more than normal twitters so contribute more to numbers shown in ”tweet-sent-bar.png”, so that the sentiment of all is dominated by active users.

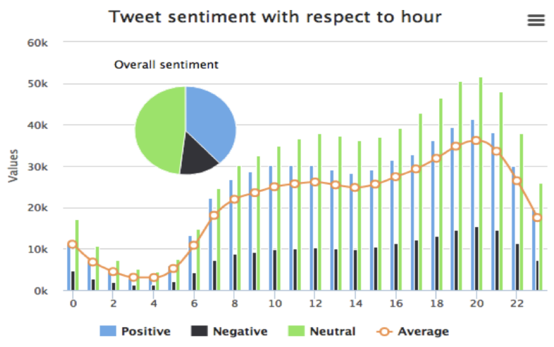


Fig. 21: Tweet sentiment with respect to hour.

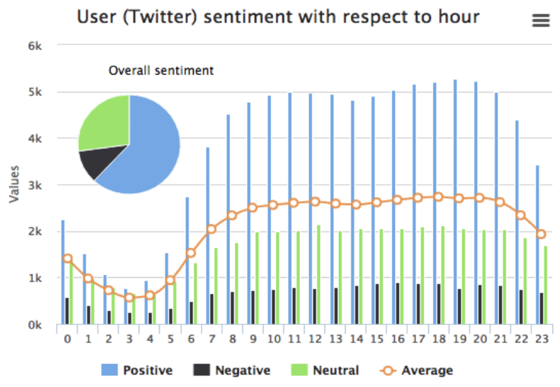


Fig. 22: User sentiment with respect to hour.

V. The architecture design of web server

Our web server use a lightweight python based server, Bottle[6]. We also use a series of css template from Bootstrap[7]. The directory organization is shown in in Figure 23.

[6]http://bottlepy.org/docs/dev/index.html

[7]http://getbootstrap.com/css/

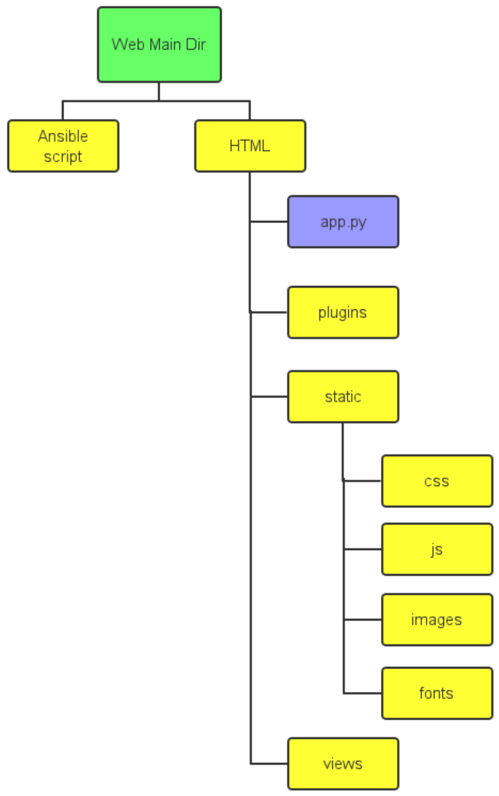


Fig. 23: Architecture design of web server.

VI. The architecture design of display component

After analyzing each scenario, the results are stored into CouchDB as JSON files. In order to visualize the results, the data need to be firstly retrieved from CouchDB. To do this, a python package called CouchDB is used. After the data is retrieved, it is firstly preprocessed in Python to the format which is suitable from visualization on the webpage. To communicate the data between web page and python, a lightweight web framework called Bottle is used.

The fetched data will be fed to the web page, and Highcharts is our choice for visualization. Highcharts makes it easy for developers to set up interactive charts in their web pages, and t is simple yet flexible. In our case, since we need to show the interactions between different parts of the data, also let users to get more involved in the web app, Highcharts turns out to be a great tool, which not only requires minimum time to learn, but also allows sufficient customization.

In addition to Highcharts, Bootstrap is also used as a library to adjust the layout of different charts. In addition to the scenario results data, a document of Aurin data (PSMA_Postcodes__Polygon___May_2015_.json) which contains postcode and suburb geographical boundary information is also used to draw heatmaps. A structure of the whole visualization process is shown in Figure 24.

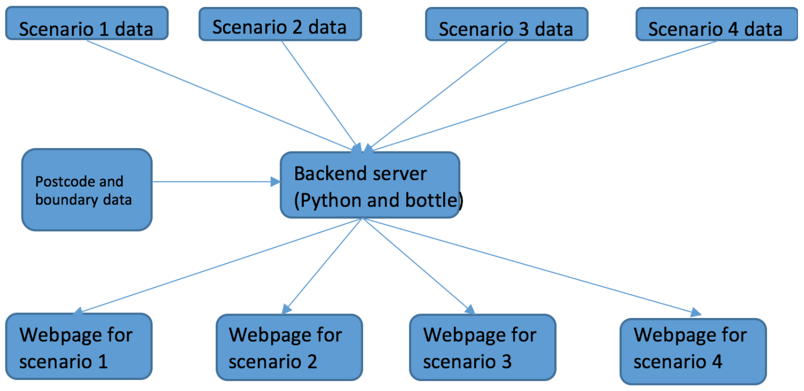
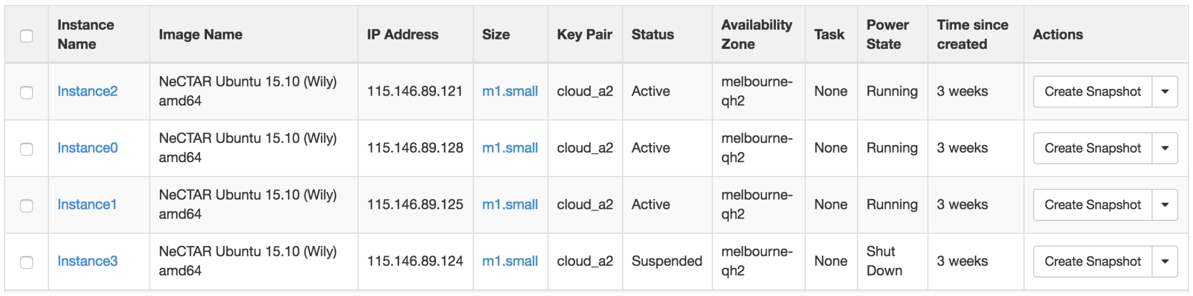


Fig. 24: Visualization structure.

VII. Application Deployment

A. *Nectar instances deployment with Boto*

In our project, we applied Boto 2, a Python interface to Amazon Web Services, to realise creating and running instances on Nectar automatically by only executing one python script. To begin with, we firstly established two security groups, ’ssh’ with port 22 and ’http’ with port 80, 443, and 5984, on Nectar. And then we created a key pair named ’cloud_a2’ for valid connection between local and remote server. Afterwards, we downloaded the EC2 Credentials and used the AWS access key in it to make connection to the service with boto. In the rest of python script, we also set availability zone as ’melbourne-qh2’ and size as ’m1.small’. The final 4 instances created as shown below in Figure 25.



	Instance Name	Image Name	IP Address	Size	Key Pair	Status	Availability Zone	Task	Power State	Time since created	Actions
☐	Instance2	NeCTAR Ubuntu 15.10 (Wily) amd64	115.146.89.121	m1.small	cloud_a2	Active	melbourne-qh2	None	Running	3 weeks	Create Snapshot
☐	Instance0	NeCTAR Ubuntu 15.10 (Wily) amd64	115.146.89.128	m1.small	cloud_a2	Active	melbourne-qh2	None	Running	3 weeks	Create Snapshot
☐	Instance1	NeCTAR Ubuntu 15.10 (Wily) amd64	115.146.89.125	m1.small	cloud_a2	Active	melbourne-qh2	None	Running	3 weeks	Create Snapshot
☐	Instance3	NeCTAR Ubuntu 15.10 (Wily) amd64	115.146.89.124	m1.small	cloud_a2	Suspended	melbourne-qh2	None	Shut Down	3 weeks	Create Snapshot

Fig. 25: Created instances.

B. *Ansible*

We use ansible[8] to deploy system environment including deploying database (CouchDB) and package installation. Also, we use ansible to deploy working scripts on remote server and execute those scripts remotely. We use a so-called ”roles” structure[9] to organise ansible scripts. The ”roles” structure is shown in Figure 26. The finer illustration of instruction is discussed in section ”User Manual”.

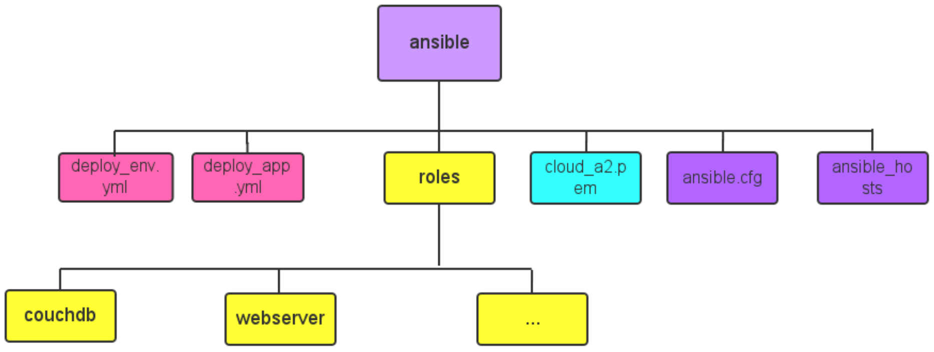


Fig. 26: Cloud Ansible.

VIII. Discussion of Nectar

A. *Nectar Cloud*

Nectar Cloud acts as an IaaS[10], is built based on Open source cloud technology concept and share some common pros and cons with it. Cloud computing is vastly used in recent years, due to the efficiency and cost saving benefits it delivers. Cloud computing refers to the delivery of computing resources as a service. It consists of three tiered architectural computing model, namely SaaS (software as a service), PaaS (platform as a service) and IaaS (infrastructure as a service) [2]. Cloud computing may also include the delivery of other types of IT assets as a service; including: storage as a service, databases as a service, security as a service and backend as a service offerings [3].

There are two main components of cloud computing technology, which are: SOA (Service Oriented Architecture) and cloud virtualization.

[8]https://www.ansible.com/

[9]http://docs.ansible.com/ansible/playbooks_roles.html

[10]https://nectar.org.au/cloudpage/

SOA allows independent web services to communicate with each other via the Internet in real time. It facilitates centralized distribution and component reuse, which significantly reduce the cost of software development and delivery.

Cloud Virtualization is another important aspect which facilitates the efficient delivery of cloud computing services. It mimics the functionality of physical computing resources, serves as a flexible load balancing management tool that allows for the swift adjustment of computing services delivery on demand. Virtualization technology provides organizations with a tool that promotes high levels of availability, scalability and reliability.

Compared with traditional PC operations, the cloud has the potential to transform it and cut costs. Offices running computer networks no longer have to install software and licenses for each computer, thus reduce IT load. Uses of the cloud include data storage, offering remote access to any work related data.

Considering transparency issue, public clouds are susceptible to outside threat that is intrinsic to delivering services via the Internet. Private clouds offer the framework for a more secure computing environment, which is secured by a network firewall. An enterprise private cloud eliminates threats from Internet hackers, as well as reduces the occurrence of intellectual property theft, which could easily occur if a company stores its sensitive data and proprietary information in a public cloud.

The pros and cons of cloud computing are listed below.

Pros: One of the most obvious benefits of using cloud computing technology is the versatility, mobility and flexibility it brings to all users. The flexibility comes from the fact that everything is completely internet based. You can access all your data from anywhere in the world, and from any device or computer. You don't have to be limited to just one computer or even a computer at all, as smart phones, tablets, and other mobile devices can all allow access. Some cloud computing services, such as Google Docs, Dropbox and email services provide convenient access to documents or mail, which is not stored on our PCs, but is available to use because it is stored on a cloud. The second advantage is that the cost of using cloud technology is far reduced as compared to current technology. The third advantage is that everything is online and extremely automatic, so that users have much less to do. You don't have to work through problems yourself, because the cloud computing service takes care of all of it for you. These pros facilitates twitter data analytics anywhere, anytime by anyone conveniently.

Cons: The only disadvantage is the risk of low quality service delivery from a cloud provider. As the wrong cloud computing service provider can be extremely detrimental for any organization. The scalability, availability and flexibility of the cloud solutions should be assessed in advance.

B. Nectar Cloud based analytics

Considering transparency, after deploying our system on Nectar, all analytics can perform as if in a single node. Considering efficiency, we can use Map-Reduce operation to accelerate our program on large dataset. Considering scalability, we can introduce more node, and just use ansible to configure it, then we can add new nodes into system, including all aspects of our application, like CouchDB, web server and analytic program.

C. IaaS, PaaS, and SaaS comparison

The Nectar Research Cloud provides computing infrastructure, software and services to researchers, and allows people to access its services literally any time from any place all over Australia. The role of Nectar is an Infrastructure as a service(IaaS) model, and we will illustrate its advantages and disadvantages compared to other models such as Platform as a service(PaaS) and Software as a service(SaaS) model.

Infrastructure as a service(IaaS)

According to the definition of IETF(Internet Engineering Task Force), the infrastructure in this context mainly refers to physical or virtual machines which have the functions of a computer. There is no need for users to get to know the underlying structure of the computing resources, like the location of physical resource, how the data is partitioned, how to ensure system security and so on. Hypervisors run the virtual machines as guests. Common hypervisors are Xen, Oracle VirtualBox, Oracle VM, and with the help of these hypervisors, many virtual machines can be set up with the cloud and the scale of services can be adjusted according to customers' requirements. Cloud users need to install the software and runtime environments in order to deploy applications.

Platform as a service(PaaS) On the other hand, however, Paas offers a development environment to application developers. Customers can customize their software by utilizing the API provided. Paas helps to facilitate and simplify the process of development, testing and deployment of applications, as well as reduce the cost. Companies benefit from Paas thanks to its reduced amount of coding and automation, since they do not need to manage or control the underlying cloud infrastructure.

Software as a Service(SaaS)

Perhaps the model that is most seen is SaaS model, where users would subscribe to a software which can be accessed online. Users are often charged on a pay-per-use basis, or pay a register fee for the service. Unlike PaaS and IaaS, SaaS customers do not manage the infrastructure and platform, which reduces maintenance and support.

Figure 27 shows the major advantages and drawbacks of different service models. In summary, the more automation the service allows, the less customization users have and the more constrained the service is.

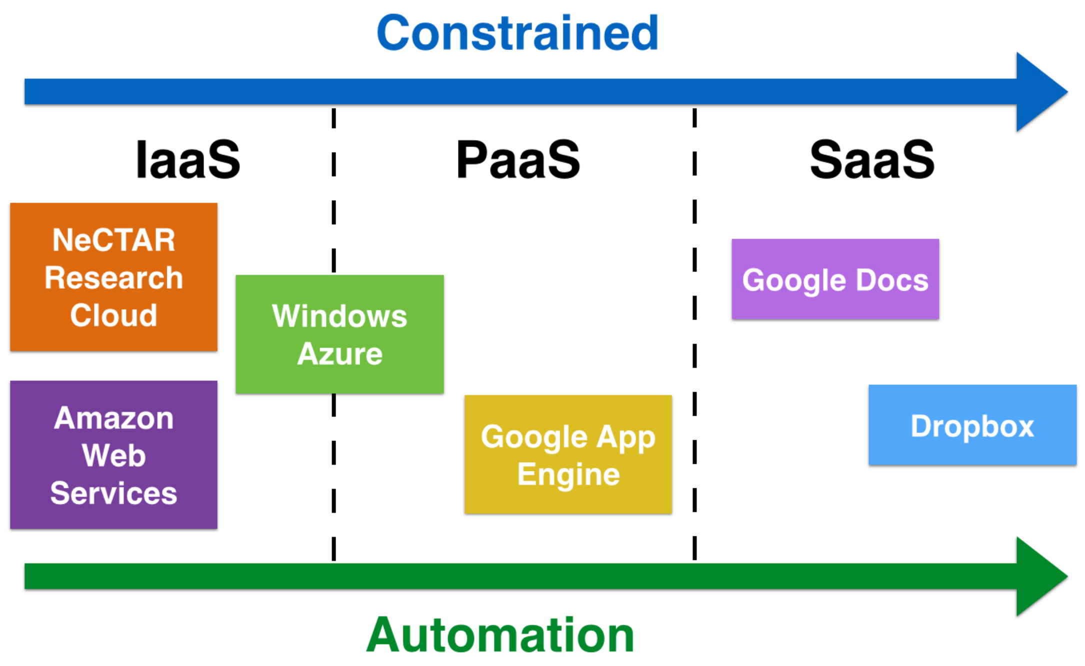


Fig. 27: Comparison of different service models.

IX. Error Handling

- Connection error handling

  As mentioned above, Twitter API v1.1 requires that request must be authenticated. Therefore, in this project, the validation of api is checked before being used. And for Search API, this error is usually caused by unmatched consumer key and consumer secret.
- Resource-get error handling

  Search REST API is rate limited. By using application-only authentication, our application can make at most 450 queries/requests per 15 minutes. An 'Rate limit exceeded' error occurs once hitting the rate limit. To handle this issue, wait_on_rate_limit and wait_on_rate_limit_notify flags are set to be true to ensure Tweepy API call can automatically sleep until expiry of the rate limit window.

  Streaming API might sometimes return the tweets that from outside the bounding box of location filter parameter. So in order to avoid this, we added a condition to check if the place full name of tweets is

’Melbourne, Victoria’ in the stream listener. This kind of double check can make sure all the received tweets are indeed from Melbourne.

The tweet stored in our database share the same id with Twitter. Therefore, before processing a tweet, an id existence check within the database helps to get rid of saving duplicate tweets.

- Component-cross (API) error handling

  Google Maps API might sometimes return wrong combination of suburb name and postcode, that is, a postcode does not belong to the particular suburb. But this happens very low frequently. So, when calculated the average sentiment score, we just used those [suburb name, postcode] keys with more than 16 values.

X. User manual

A. Description

The demonstration directory-tree in the local environment is like in the following Figure 28. The directory ”ansible” contains auto deployment scripts. ”Cloud_a2_boto.py” initializes VM on nectar. All zip files contains scripts that run on remote servers.

B. Instructions

Generally, we give two options for system deployment and invocation: one is that the one can run scripts in respect of only some part of the whole application he interests, we divide the ansible auto-deploy script into modules, each module concerns an aspect of functionality of the application; another is simpler with only two scripts to run. We strongly recommend the demonstrator follow the former scheme, i.e. execute each module separately. Because the analysis script consumes too much time which could hinder the demonstration, so you may want to execute it at last in demonstration situation. Before running anything, you need to make sure the working environment is Linux and pre installed Python and Ansible. In below, we firstly give step-by-step instruction of the first option and meanwhile illustrate functionality of each module:

1. Unzip ”cloud_asg_script.zip”, then there will be a ”cloud_asg” folder in current directory. All directories in ”cloud_asg” will be like Figure 28.

2. Run the following command in ”cloud_asg” directory to initialize 4 Ubuntu VMs on Nectar ”python cloud_a2_boto.py [your nectar access id] [your nectar access key] [instance key name]” You could run the script without any arguments, the script will use the default arguments, but that may fail because the default account may run out of allocated resource.

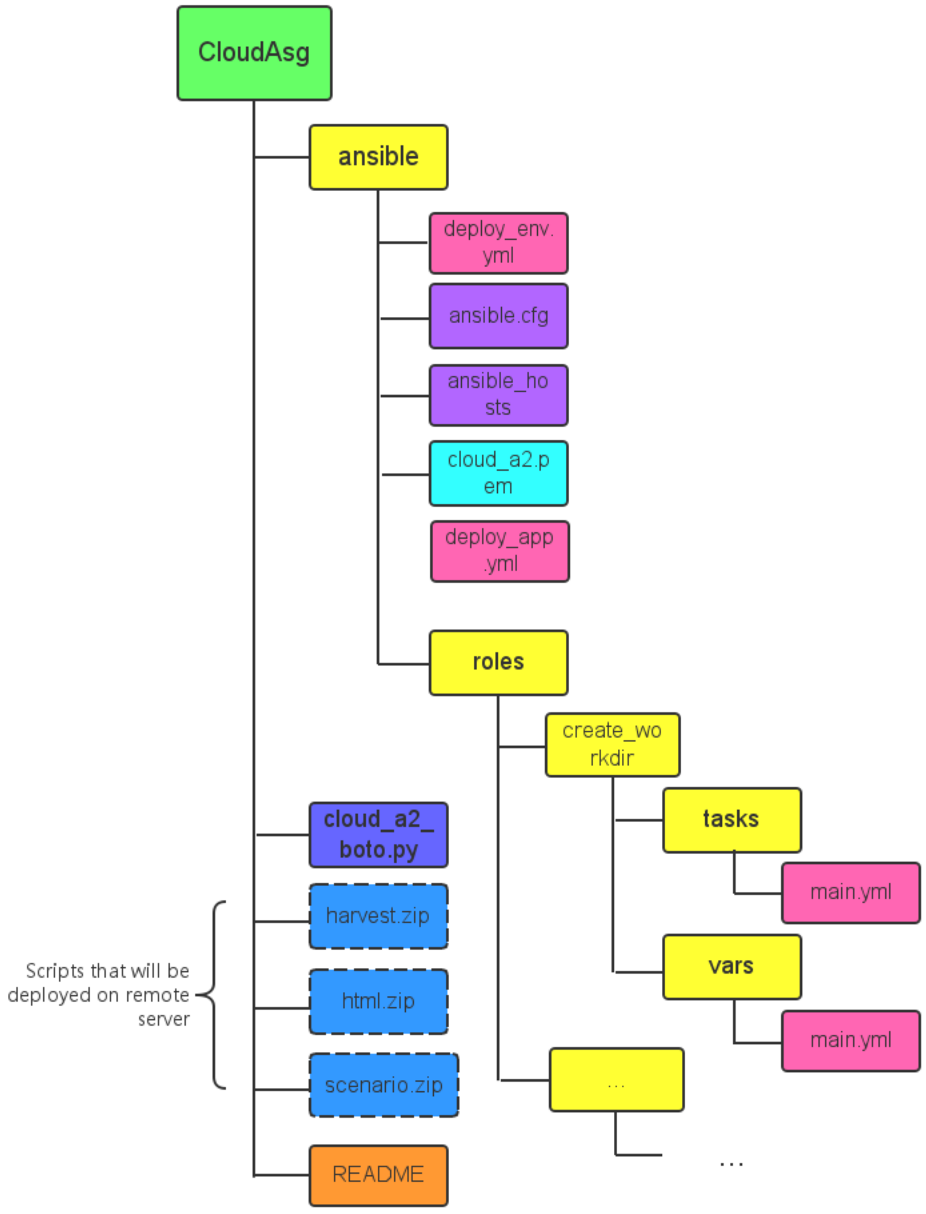


Fig. 28: Demonstration directory tree.

3. Go to directory ”cloud_asg” and edit file ”ansible_hosts” to replace the IP address with your remote VMs on nectar. Make sure IPs under ”[cloud_asg]” include IPs under ”[app]” in file ”ansible_hosts”.

4. Add your Nectar instance access key file in directory ”cloud_asg” and replace the private key file name in ”ansible.cfg” in the same directory. Alternatively, you can do nothing on this step, then the following steps will deploy all on our nectar VMs.

5. Run the following command in ”cloud_asg/ansible” directory to create working directory on remote Nectar VMs ”ansible-playbook roles/create_workdir/tasks/create_workdir.yml”

6. Run the following command in ”cloud_asg/ansible” directory to deploy CouchDB on remote Nectar VMs ”ansible-playbook roles/couchdb/tasks/deploy_couchdb.yml”

7. Run the following command in ”cloud_asg/ansible” directory to install all packages in need on remote Nectar VMs ”ansible-playbook roles/pkg_installation/tasks/install_pkg.yml”

8. Run the following command in ”cloud_asg/ansible” directory to deploy tweets harvest scripts and run harvest scripts on remote Nectar VMs ”ansible-playbook roles/harvest/tasks/deploy_harvest.yml”

9. Run the following command in ”cloud_asg/ansible” directory to deploy analysis scripts and run all analytic scripts on remote Nectar VMs ”ansible-playbook roles/scenario/tasks/deploy_analytics.yml”

10. Run the following command in ”cloud_asg/ansible” directory to deploy web server and start the web server on remote Nectar VMs ”ansible-playbook roles/webserver/tasks/deploy_webserver.yml”

Now, you can check the website and analytic results on ”http://[your webserver IP]/” (IP defined in ”ansible_hosts” under section [app]).

Alternatively, you can run the following command to replace step 5 to step 10:

”ansible-playbook deploy_env.yml” ”ansible-playbook deploy_app.yml”

XI. Conclusion

In this project, we focusing on harvesting and analyzing tweets and building a web application to visualize the distribution results via various charts. The usage of Aurin and IMDb also assists us to discover more interesting patterns. As mentioned above, many trends have been found. In Melbourne, a higher rating movie tend to draw more heated discussion on Twitter; the suburbs with higher income level always maintain a higher level of sentiment as well; Melburnians prefer to post their happiness rather than depression, especially on daytime. By mining the fine-grained topic preference and active twitters in each suburb, social statistics is well analyzed and visualized. The language distribution ratio in each suburb implies the population constitute in different suburbs, which provides valuable feedback for Bureau of Census and also beneficial for population related scientific research. Considering implementation, our project is scalable enough to easily apply to researches on other cities in Australia.

References

[1] J. Dean and S. Ghemawat, “Mapreduce: simplified data processing on large clusters,” *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[2] K. Hwang, J. Dongarra, and G. C. Fox, *Distributed and cloud computing: from parallel processing to the internet of things*. Morgan Kaufmann, 2013.

[3] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, “A view of cloud computing,” *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.