Realtime Twitter sentiment analysis platform on NeCTAR Research Cloud

Tong He	the2@student.unimelb.edu.au
Huiya Chen	h.chen81@student.unimelb.edu.au
Yao Wang	y.wang453@student.unimelb.edu.au

Abstract

This report presents a Twitter sentiment analysis platform we build on the NeCTAR Research Cloud. The pipeline model of the platform includes Twitter harvester, CouchDB, web server for data visual analysis, all of which are distributedly deployed on the cloud and uses Ansible for IT automation. Firstly, the system functionalities will be discussed as well as a detailed description of the deployment process and intended usage of the system will provide a simple guide for testing. Then we will go into a discussion on the system architecture, technologies used and overall system performance. Next will be sentiment analysis and some graphical analysis and reporting of the results. This report will also include a risk registry, discussion on the UniMelb Research Cloud and then conclude with some final comments.

Table of Contents

List of Figures

Chapter 1. Introduction

Twitter alone has 326 million active users and 500 million tweets per day, combine this with Facebook's two billion active users as well as Instagram, Google, Youtube and many more social media platforms and it becomes fairly obvious that the amount of data generated from Social Media alone is monumental. What makes social media data specifically interesting is the amount and quality of sentiment analysis that can be achieved to understand people and society through data analytics of social media posts. For example, each tweet is written by a user, that user has an ID and a location, simply using the tweet and location we can begin to understand whether people are generally happy in certain areas, based on the words they use in their tweets, or if they are particularly angry in a certain location, perhaps because their Footy team has just lost the grand final.

What makes this problem specifically interesting and challenging is dealing with the large amounts of data generated by users, known as Big Data. For a system to be able to get, store and analyze Big Data, technologies used, architecture and use of computing resources must be specifically chosen, designed and deployed. The notion of Big Data does not just involve the volume of data but also includes:

1. Velocity: The rate at which new data is imported into the system
2. Variety: The variability of DB schema, how well is the schema at adapting to changing environments
3. Veracity: The level of trust in the data accuracy

One widely used solution to this issue is using a Cluster-based architecture that is deployed in the CLoud. The cloud services utilized for this project is the National eResearch Collaboration Tools and Resources research cloud (NeCTAR) which operates based on the OpenStack protocol.

The purpose of this project is to build a cloud-based system which uses the computing resources of multiple VMs and nodes, this system will be used to analyze and explore large sets of social media data as well as validate these findings with data available within the AURIN platform. This must all be achieved in a scalable manner which allows for elastic scaling. The system will have three main components, twitter harvesting application, CouchDB database and a front-end web application. The following five activities outline the five key steps involved in building this system.

1. Cloud-based system: exploits a multitude of machines, can run across any node to harvest and store tweets
2. Harvesting of social media data
3. Storage of social media data
4. Analysis of social media data
5. Validation against real data from AURIN

The following two sections explore the system functionalities based on the above five key activities as well as a user guide for testing. In section five the way different software components are distributed on computers, and the way in which they interact with each other will be discussed.

Lastly, the entire system needs separate components to integrate seamlessly, which involves different network services, different software and frameworks, and different programming languages. This requires the team to have a varied level of programming and members can be involved in different skill directions. The team has to not only consider the completion of individual functions, but also the compatibility issues between them and the way data is delivered. The team needs to have good communication skills and collaboration skills if the final system is to be delivered, making this assignment just as much about teamwork in software development as well as technical skills such as IT and programming skills.

Chapter 2. System Architecture

2.1 Overview

Architecture design along with technologies chosen has been designed with the following main goals in mind. Firstly and most importantly is elasticity and scalability. The system must be able to elastically scale upward and downward with the number of resources required. For example, suppose there was a natural disaster which caused the number of Tweets to drastically increase, the system should have the capability to deploy more twitter harvester nodes to cope with the increased data being input into the system. Secondly, the system must be fault and failure tolerant, meaning that from the perspective of the end user, there should be no effect if one node fails. For example, if a CouchDB node fails there should be no loss of data. Lastly, the system must have scripted deployment capabilities, meaning that all virtual machines can be deployed and configured with a script. Please note that although these are the main goals, this does not imply that other goals such as availability, consistency and overall performance of the system have not been considered.

As previously discussed the three main functions of the system are a CouchDB Service, twitter harvesting service and a web server for visualizing the information.

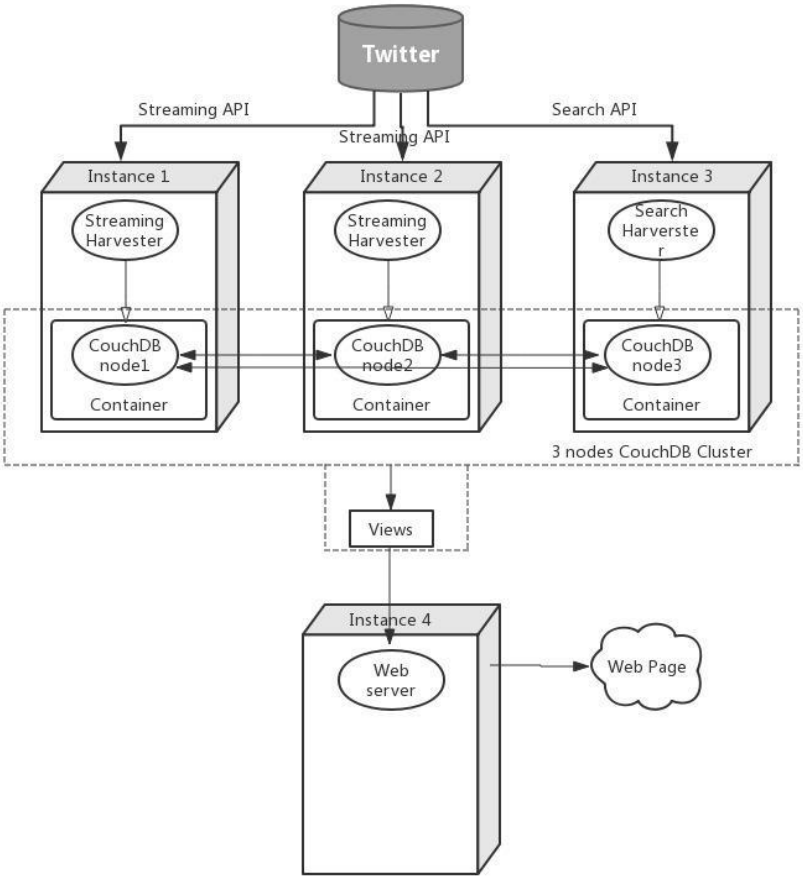


Figure 1: Overall System Architecture

2.2 CouchDB Cluster

Three instances operate as servicing both the database and twitter harvesting functions and the last instance acts as a web server. The CouchDB cluster is deployed and managed by Docker. Docker is a platform for developers and sysadmins to develop, deploy, and run applications with containers. The use of Linux containers to deploy applications is called containerization. This project uses docker to help deploy and manage the CouchDB cluster.

We use image CouchDB: 2.3.0 to create containers, and docker will auto download and install CouchDB and its dependencies. When the container is created, the network is mapped to the host machine network, so that some external ports that can access the CouchDB, such as 5984, called the CouchDB API. The volume path/CouchDB of the instance is mounted to the inside of the container /opt/couchdb/data, which makes data stored during the container's running persistent instead of disappearing as the container is deleted.

Inside each of the Instances running a CouchDB node, there is also a twitter harvester, instance one and two use the Stream API whilst the third instance uses the Search API. Each harvester will submit it's harvested to the CouchDB node in its localhost, then CouchDB will take care of replication across nodes.

2.3 Web Server

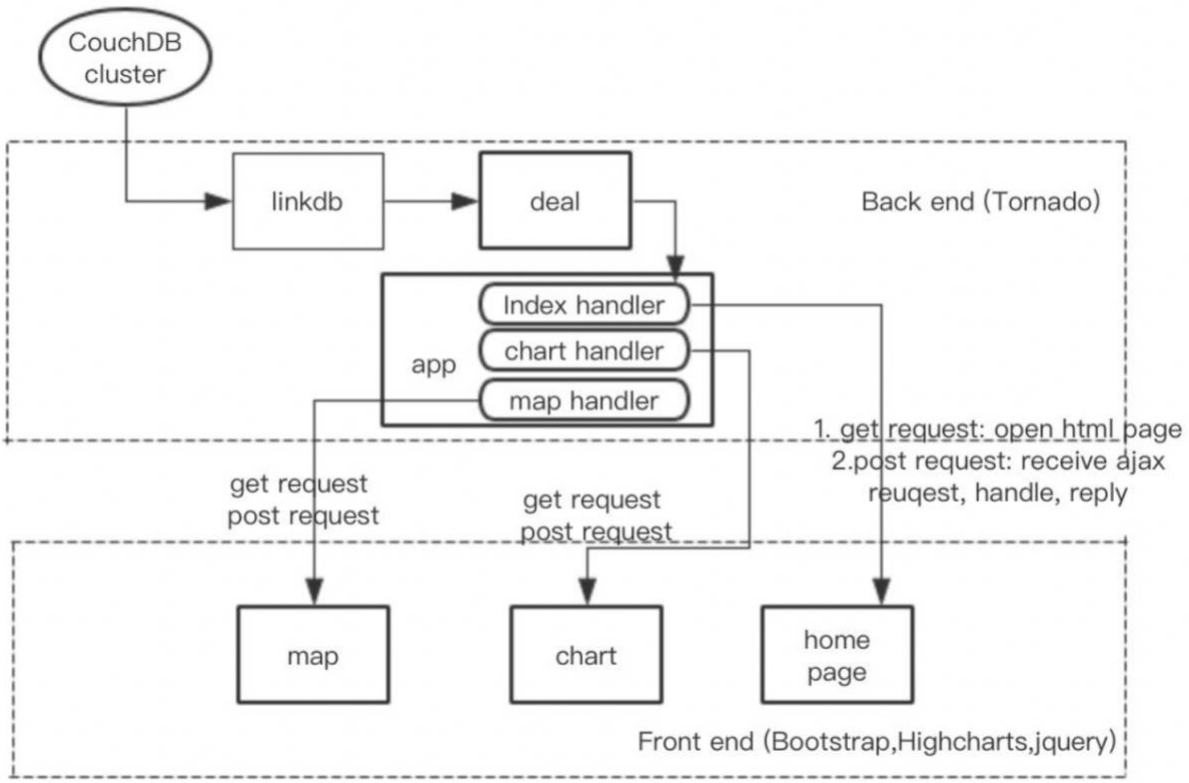


Figure 2: Web Server Architecture

There are three HTML pages in the front end. the home page, chart page and map respectively. The back end is divided into three modules. The linkdb module is responsible for reading the raw data from databases. The app module is a master, responsible for running the corresponding HTML page, processing Ajax requests and returning the corresponding data. It contains three handlers, which correspond to the three pages displayed at the front end. The deal is a data processing module, which is responsible for calling the original data obtained by linkdb, and processing them, finally returning the required data results to the app module.

Chapter 3. System Functionalities

All the source code of this project can be seen at Github repository, the link is shown in section 10.

3.1 Overview

Broadly speaking the system has three main functions. Firstly to harvest tweets from twitter, secondly to store tweets from twitter and thirdly to analyze the data stored in the DB and A front-end web visualizing these data sets/scenarios. Due to a large amount of data that is being written and read from the DB a NoSQL database, CouchDB is chosen. The main reason for NoSQL databases performing faster in regards to read and write is because they are horizontally scalable. All of the data for this assignment as stored in CouchDB including tweets harvested from Twitter and raw data taken from AURIN.

NeCTAR is an infrastructure which offers computing resources as a service, for the purpose of this system four VM's have been utilized for a range of functions. For automated deployment and configuration of VM's ansible playbook has been used. At a high level, one VM is for CouchDB, one for a web server and two for twitter harvesting. A python program is utilized for the purpose of harvesting tweets and saving them in CouchDB.

3.2 Harvesting Tweets

Before storing or Twitter data, the harvesting application will make use of Twitter API's to collect tweets from Australia. One of the key criteria of this component is that it must be scalable, meaning it can be run across and node of the Cloud to harvest and store tweets. The scalability of this component will be discussed in the user guide section of the report.

Twitter has two public API's that can be used for harvesting tweets, Streaming and Searching. To access these API's a Python program, "name of the program", is used as well as the Tweepy library for Python. The python program will take advantage of both the Search and Streaming APIs. Also, there is a pre-processing of twitter data so that storage is persistent and consistent. Pre-processing ensures no duplicate entries are added to the DB, JSON documents are cleaned so that only relevant and valuable information is stored. Multiple

instances of the harvester can be deployed into the system, receiving, cleaning and storing tweets in the database. Please note that the following diagram is an abstraction and simplifies the architecture of the CouchDB cluster.

So, what exactly fits inside a Twitter Harvester? The twitter harvester does some pre-processing before storing files in CouchDB. Visually this looks as follows.

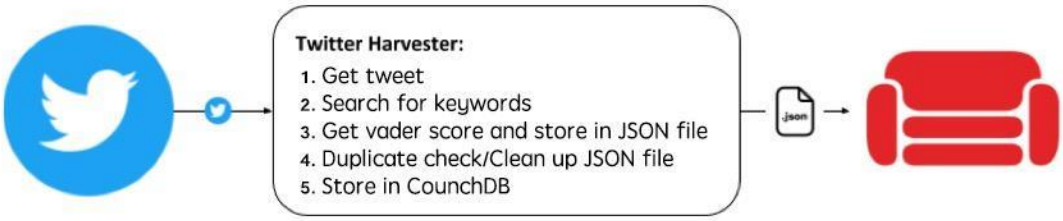


Figure 3: Processing of Tweets algorithm

Step 1. Use twitter API to get tweets where the geolocations are posted in Australia:

Using Twitter streaming and search API's as well as tweepy, a python library, the python program run.py will begin to harvest tweets.

Step 2. Search for keywords.

1. Allocate each tweet to a specific region as partitioned by the Aurin region partition criteria.

2. Check each tweet for keywords that contain car brands (see appendix for a specific list of keywords).

Step 3. Get Vader Score

Vader Sentiment is a library in python that allows for sentiment analysis on social media texts. This is an open source library and is widely used and popular because it not only tells about the Positivity and Negativity score but also tells us about how positive or negative a sentiment is. A more in-depth description of how the scoring is achieved can be found at github [4].

In step 3 each tweet is parsed into the function getVaderScore and that score is stored into the JSON file.

Step 4. Duplicate check/Clean up JSON file:

To check for duplicates we simply use the unique ID provided by Twitter and set it to the primary key of the DB. To clean up the JSON file, as shown in TwitterAPItools.py, create a dictionary and only get the key-value pairs we require from the JSON file and store in the temp dic. After all operations are complete on the dic it is then outputted as a clean JSON file.

Step 5: Store in CouchDB

As previously mentioned the Twitter's RESTful API's will be used to access twitter data. API's are advantageous as developers do not need to install libraries or additional software to take advantage of the RESTful API.

3.2.1 Search API

The Search API [2] provided by Twitter comes in several categories, for the purpose of this system the Standard category will be used which restricts our data to just the last seven days, one of the limitations of the system. The second limitation is a maximum of 180 requests per 15 minutes of use and also only a sample of Tweets are made available, not all tweets matching search criteria will be returned, this is because the standard search API is focused on relevance and not completeness. If completeness is more important than paid versions such as premium or enterprise would be needed. Although only standard API has been used, for all purposes the system is capable of dealing with larger quantities of data. To limit the Tweets to only those occurring in Australia the Search API specifies a circle.

3.2.2 Streaming API

As well as the search API, Twitter also provides a Streaming API [3] for streaming real-time tweets. As a standard user, this is limited by 400 keywords, 5000 user IDs and 25 location boxes. Unlike the search API, streaming API ensures completeness as it maintains a persistent client-server connection and also does not limit the count of Tweets that can be harvested.

For the purpose of this application, streaming API will be used to continually add the most recent tweets to our analysis on top of historical tweets already stored in CouchDB. The streaming API gives access to several parameters. For the purpose of this system, the following parameters have been utilized.

- Language = [‘en’]: Only return tweets written in English
- Locations which specifies a bounding box to filter the Tweets by. The box will be specified by the upper left and lower right corners of Australia

Once an HTTP request has been submitted to the Streaming API the stream can be consumed for as long as practical barring any server-side error on the Twitter side.

3.2.3 Comparison of Streaming vs Search API’s

The first difference is the way in which the region is specified, the Search API uses a circle to determine what region a Tweet falls in however the Streaming API uses a rectangle for this operation. This is the downside of using different API’s as it creates a challenge when augmenting the data into a single repository. Secondly, the Streaming API returns tweets one by one however the Search API returns tweets in a batch format. Lastly, because the standard Search API has been used there is no guarantee on completeness, meaning that only a subset of the data will be provided whereas the Streaming API guarantees the completeness of data.

3.2.4 Other data resources used

Since the tweets with geo locations obtained from Search API and Streaming API is too few, extra Twitter data has been sourced from UniMelb Research Cloud using the location and time filers.

3.3 CouchDB

CouchDB provides the main functionality for storing information from both Twitter and Aurin data as well analysis on this information using MapReduce functionality. CouchDB is an open source and focuses on establishing a scalable architecture. It is a documented oriented NoSQL database which uses JSON format to store data. The main functionality of CouchDB which makes it suitable for use is that it allows for distributed architecture with replication, Map/Reduce views for analysis of data and an HTTP API for reading and writing updates to the database. Other benefits include the ease of setup, web-based UI for the admin interface, and inbuilt functionality for sharding, clustering and replication. Version 2.3.0 is the latest current version and the one that will be used for this system.

Two important parameters to take note of are the number of shards, q, and the number of replicas, n. For the purpose of this system the CouchDB defaults will be used which is eight shards and three replicas. A shard is a horizontal partition of the data, once partitioned into shards, n copies of each shard are distributed to the number of nodes in the system. This provides protection against failing nodes in the cluster. With eight shards and three replicas, this leads to 24 shard replicas, meaning that on a three-node system there will be 8 shard replicas on each node.

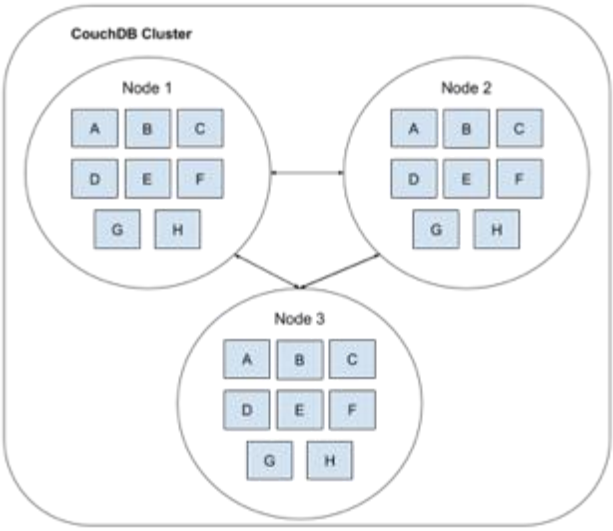


Figure 4: Shard Replica representation across three nodes with 8 shards and 3 replica

When deploying the system it is recommended to use a number of nodes which is evenly divisible by the number of shard replicas, in the case of this system that means using either 2, 3, 4 or 6 node cluster would be acceptable. As the number of nodes increases, CouchDB will distribute the shards evenly amongst nodes ensuring that in the event a node fails the complete contents of the database will still be maintained.

CouchDB also provided an HTTP API that can be used to interface between CouchDB and other components in the system which needs access to the documents stored in CouchDB. For the purpose of this system this API will be used for the following functions:

1. Harvester will use API to store data in CouchDB
2. Aurin Data will be manually uploaded to CouchDB
3. Website for visualization will use API to pull data from CouchDB to be visualized.

The last main functionality provided by CouchDB is the ability to perform Map/Reduce functions on the stored data. Map/Reduce functions work by first running a Map function which produces a list of key-value pairs and then a Reduce function that reduces the list to a single value. This family of functions is well suited for jobs that need to run in parallel. The following is a list of MapReduce functions used. Views are the primary tool used for querying and reporting on CouchDB databases. In this case, we extract each brand of the luxury car appeared in the text file as the key and combine it with the score of the sentiment analysis as the value. When viewing the View, it will pass some parameters such as grouping requirements (group_level). The reduce function will group the key values according to these parameters and calculate the aggregation result of value. For example, group_level = 2, the view is aggregated by the first two keys, the data is grouped by [city, car brand], and the sum of the values of each of their key-value pairs is returned. In fact, we called CouchDB's built-in reduce function _stats to return a dictionary containing the sum, number, maximum and minimum, and sum of squares.

View List:

1. /database/car/_design/car/_view/byRegion
   The database filters out the tweet whose text contains the car brands, grouped by region and car brand, and returns _state
2. /database/car/_design/car/_view/superCar
   Same as above, but the car brand is limited to the luxury car brand of these brands 3. /database/twitter/_design/time/_view/score
   Group by post time, return _stats
   Same as above, but the car brand is limited to the luxury car brand of these brands
3. /database/twitter/_design/time/_view/score
   Group by post time, return _stats

It should be noted that CouchDB only provides ‘eventual’ consistency on the data storage, this is the trade-off for using a distributed system and replication. The development team also aimed to use Spark as a computing engine as Apache Spark provides the ability for more complex Map/Reduce functions to be implemented. However, Apache Spark uses HDFS (Hadoop Distributed File System) which is not compatible with CouchDB, although technically this could have been achieved through using Apache Bahir and spark-cloudant as

the connector, it added an extra layer of complexity such as setting up Spark cluster and also configuring the spark-cloudant connector which is a challenging task.

Chapter 4. User guide for testing

4.1 Overview

Although the system has been developed and configured on the NeCTAR cloud, the system does not rely on any requirements that are specific to the NeCTAR cloud, and can, therefore, be deployed to other cloud infrastructures. To automate the deployment and configuration of the system Ansible has been used. Manuel deployment is risky as it is easy to forget what software is installed, what version and what steps were taken during configuration. Also, it is a manual process which is susceptible to human error and lastly there is no log or record of configuration activities.

Using a scripting tool such as Ansible provides a record of all activities done, also makes the process repeatable but most importantly it makes the process scalable. For example, given the necessary resources, it should be possible to scale up the system just by changing a few lines of YAML code.

For the purpose of this system, there are two main Ansible playbooks that need to be run, the first one creates the Instances and the second configure the instances.

4.2 Creation of Instances

The instances can be created by executing the following command in a terminal window from within the create_instance directory within Ansible.

./run_createinstance.sh

Executing this file will then call the following command which will run the create_instance.yaml playbook and begin deploying the instances.

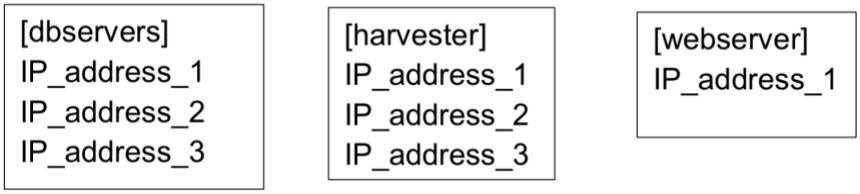
ansible-playbook --ask-become-pass create_instance.yaml

In Ansible each playbook has defined roles which can be found in the nectar.yaml file. For deployment purposes, we have the following six roles. Firstly there is a common role which will install pip, update pip (if localhost already has pip) and finally install the OpenStack

SDK on the localhost machine. The second, third and fourth roles are used for getting images, creating volumes and creating security groups. The final role named instance (create_instance/roles/instance/tasks/main.yaml) will create and deploy the four instances needed for the system. After instances have been created, it is important to note down the IP addresses for the configuration component of the process.

4.3 Configuration of Instances

In this step, Ansible connects to the remote host via ssh using the network IP addresses recorded from the previous step. These IP addresses must be stored in couchCB_cluster/inventory/inventory.ini in the following structure. Similarly, this must be done for the web and harvester inventory.ini files.



Configuring the instances is split up into three major components; CouchDB, harvester and web server. These will each be done in a separate playbook named CouchDB_cluster, web and harvester respectively. All three playbooks have an inventory file for specifying the address of the node and also have one common role named remote_common. This role performs the task of configuring a proxy, this is needed so that the instance can access the wider web to perform git pull requests and install libraries as well as installing the relevant dependencies. As per the following Ansible script.

```
- name: Proxy_env
  become: true
  copy:
    src: ./file/environment
    dest: /etc/environment
    mode: 0755

- name: Install dependencies
  tags: always
  become: yes
  apt:
    name: [ 'python3', 'python3-setuptools', 'python3-pip',]
    state: latest
    install_recommends: no
```

The coordinator node to set up the overall cluster and therefore the task build-cluster will only be run on the coordinator node.

Similarly, the web and harvester Ansible playbooks will configure a web server and the harvesters respectively. The web server has one role which Installs Tornado, clones the Git repo and then starts the web server. The harvester playbook has two roles, one for the search API harvester and one for the steaming API harvester. Configuration can be completed by running each of the three following commands in this order.

```
./run_couchDB_Cluster.sh
./run_harvester.sh
./run  web.sh
```

These commands execute the three following playbooks.

```
. ./openrc.sh; ansible-playbook -i inventory/inventory.ini -u ubuntu --ask-become-pass cluster.yaml
. ./openrc.sh; ansible-playbook -i inventory/inventory.ini -u ubuntu --ask-become-pass harvester.yaml
. ./openrc.sh; ansible-playbook -i inventory/inventory.ini -u ubuntu --ask-become-pass web.yaml
```

After these processors have completed running, the system has been deployed and configured and is ready for use. For a complete demonstration of this deployment please watch the following Youtube video(https://youtu.be/M2bnRExy_VM)

Chapter 5. Security and Fault Tolerance

At the highest level, the overall system relies significantly on the security features and functions of the NeCTAR Research Cloud. For example, all instances are UME instances and therefore are not accessible via a public IP address. All instances are allocated with login security groups allowing minimal inbound packets to instances. In addition to this, a few extra configurations are needed for the overall system to function as intended, these include:

- Inbound – Allow – Port 5984 from (Instance 2 Address): For allowing access to CouchDB service (for replication/backup purpose only)

This extra configuration should not bring any weakness to the overall system. Port 5984 has been opened for replication and communication between nodes in the cluster, this is a default CouchDB configuration when operating with clusters.

5.1 Fault Tolerance

When looking at fault tolerance each component of the system is considered. Firstly, CouchDB is deployed into a cluster of multiple nodes, each of these instances also run the twitter harvesting application and therefore Harvesters can find their localhost database to send processed tweets too. After this, CouchDB does the communication which handles replication. Therefore, even if one node were to break down or fail, the overall system still functions as intended until the failed node and be relaunched.

Similarly, the twitter harvesters are supported by the same fault tolerance mechanism as mentioned above, if one harvester goes down or fails this does not impact the overall ability of the system to perform its function. This is because multiple harvesters are deployed as separate components of the system.

5.2 Removal of Duplication

As there are multiple harvesters running simultaneously it is possible that the Twitter API will send a tweet with the same ID to more than one instance, in this case, there must be a mechanism for ensuring duplicate tweets are not stored in the CouchDB database.

This is important because storage, processing and analysis can all be negatively affected by

the existence of duplicate tweets. Each tweet has an ID as specified by Twitter, this is then used as the primary key for storage in the database. In this way, it is ensured that no duplicate tweets exist.

Chapter 6. Sentiment Analysis

The purpose of the overall system is to provide the necessary functionality to do sentiment analysis on social media data. Sentiment analysis on social media is the process of taking social media posts, in our case Tweets, and abstracting, identifying and quantifying opinions expressed in a piece of text, especially to determine whether the author of the text is neutral, positive or negative to a certain opinion or topic. This can be thought of as a sub-field of Natural Language Processing (NLP). This is a very interesting area as there is lots of insight and value to be taken from social media data. Combining these sentiments with data extracted from AURIN will provide the ability to find a correlation between social media sentiment and a particular area of research. When selecting a Sentiment analysis tool, it was key to consider that social media data is unlike general text as it involves shorthand, slang, memes, images and emotions.

6.1 Vader Sentiment Analysis

As previously mentioned, the python library VADER has been selected for performing Sentiment analysis. Vader is a 'lexicon and rule-based sentiment analysis tool that is *specifically attuned to sentiments expressed in social media*.' Once applying vaderSentiment, the library will return a sentiment results in the form of:

{'neg': 0.0, 'neu': 0.3, 'pos': 0.7, 'compound': 0.8316}

The pos, neu, neg scores are ratios for proportions of text that fall in each category, these will sum to unity. The Compound score is a metric that calculates the sum of all the lexicon ratings which have been normalized between -1 (most extreme negative) and +1 (most extreme positive).

Once the VADER sentiment score has been computed, the compound value is recorded in the tweets document in CouchDB under the key reference 'compound'. This compounded score is used to determine if a tweet is neutral, positive or negative under the following guidelines.

- Positive sentiment: compound score >= 0.05
- Neutral sentiment: -0.05 < compound score < 0.05

- Negative sentiment: compound score <= -0.05

6.2 Scenarios Supported

6.2.1 Data illustration

In this section the scenarios supported and the data that is visualized on the website will be explored. On the home page, there are three key pieces of information. The webpage can be found at http://webserver_ip:8888. Firstly is the chart page which is located at http://webserver_ip:8888/chart and this page has three notable charts. One of the great features of our webpage is the live loading of information which has been achieved through using the AJAX library.

Figure 5: Live Tweet Count Graph

This chart shows the number of tweets processed by harvesters per second. Aside from this, there is a live count of a total number of tweets processed, at the time of this report totally over two and half million tweets. Alongside this is a bar chart which has the breakdown of earners (over $4000 of weekly income). This information has been accessed from AURIN data. Hovering the mouse over each bar will provide the actual count in each city. Clicking the name of the city will provide the user with a more specific breakdown within each city where the high-income earners live, as seen in the following screenshot.

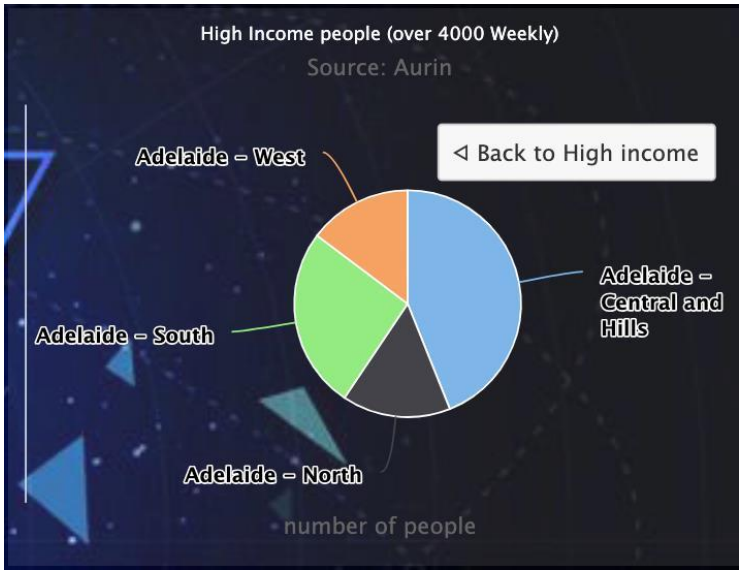


Figure 6: High Income Earners Breakdown

All of this information can be further analyzed and visualized on the Map page of the website, which can be found at http://webserver_ip:8888/map.

The map is color coded so that the regions with the highest income earners are in green and those regions with the lower count of high-income earners are in dark red. To understand how the interactive map is useful lets run an example. Let's suppose that the sentiment or popularity toward a certain car brand in a certain region is of interest, for example, the Inner-south suburbs of Melbourne. The user can use their courser to click on this region and then two following graphs will load.

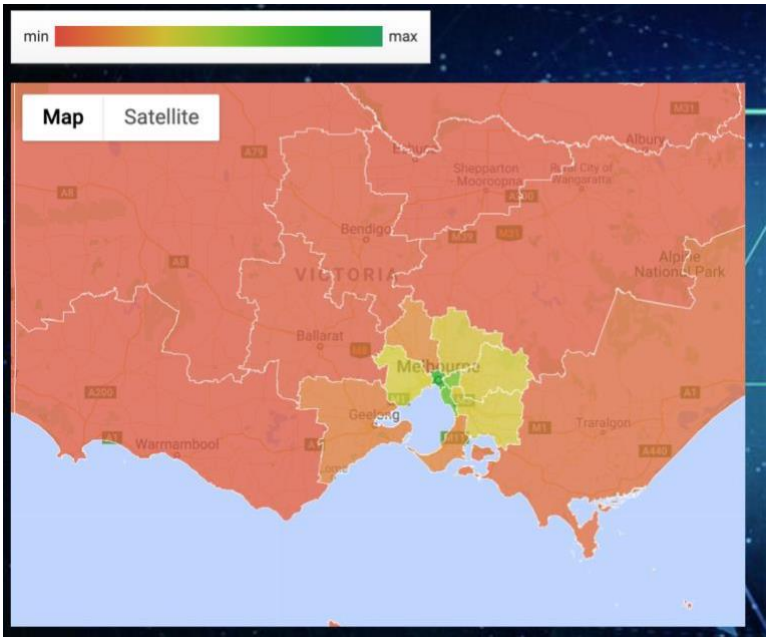


Figure 7: Interactive Map

Figure 8: Vader of all brands in Melbourne

From this information, the user can begin to have an idea of how people in specific regions feel toward specific car brands. For example, in the inner south of Melbourne, BMW and Mercedes seem to have fairly positive sentiment however Rolls Royce and Bentley although still positive sentiment, it is not as positive as the former. Compared to a less wealthy area, for example, North-east of Melbourne, there is lower positive sentiment toward expensive cars such as BMW and Mercedes.

6.2.2 The deadly sins of Envy -- High income vs desire for luxury cars

The aim of this sentiment analysis is to relate the location of high-income earners to the twitter data. For example, we would expect that in a location with high-income earners, they would be most likely to have a positive sentiment toward expensive cars.

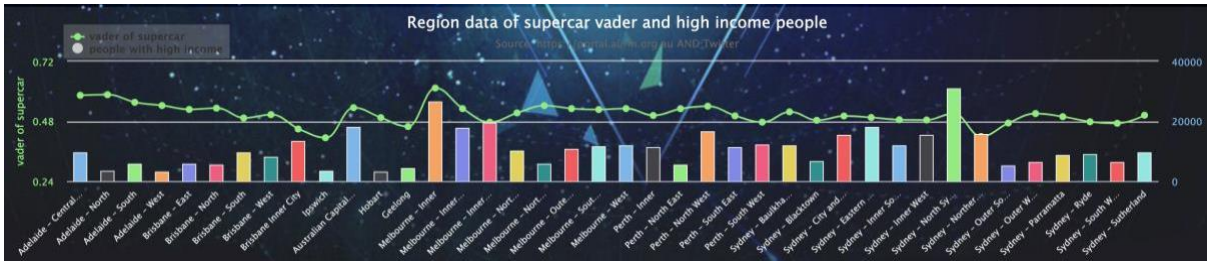


Figure 9: Region data of Supercar Vader and High-Income Earners

Generally speaking, the emotional averages of all regions are positive, indicating that no matter whether a person is poor or rich, they are longing for luxury cars. The luxury car Vader score does not vary too much across Australia, with the lowest score of 0.41 and the highest score of 0.61. As expected, both Melbourne and Sydney have high average Vade*f*r scores on luxury cars. One interesting outcome was that although Adelaide has a small population of high-income earners they tweet a lot about luxury cars and have a high sentiment toward them.

6.2.3 The deadly sins of Sloth-- high income vs hours tweets counts

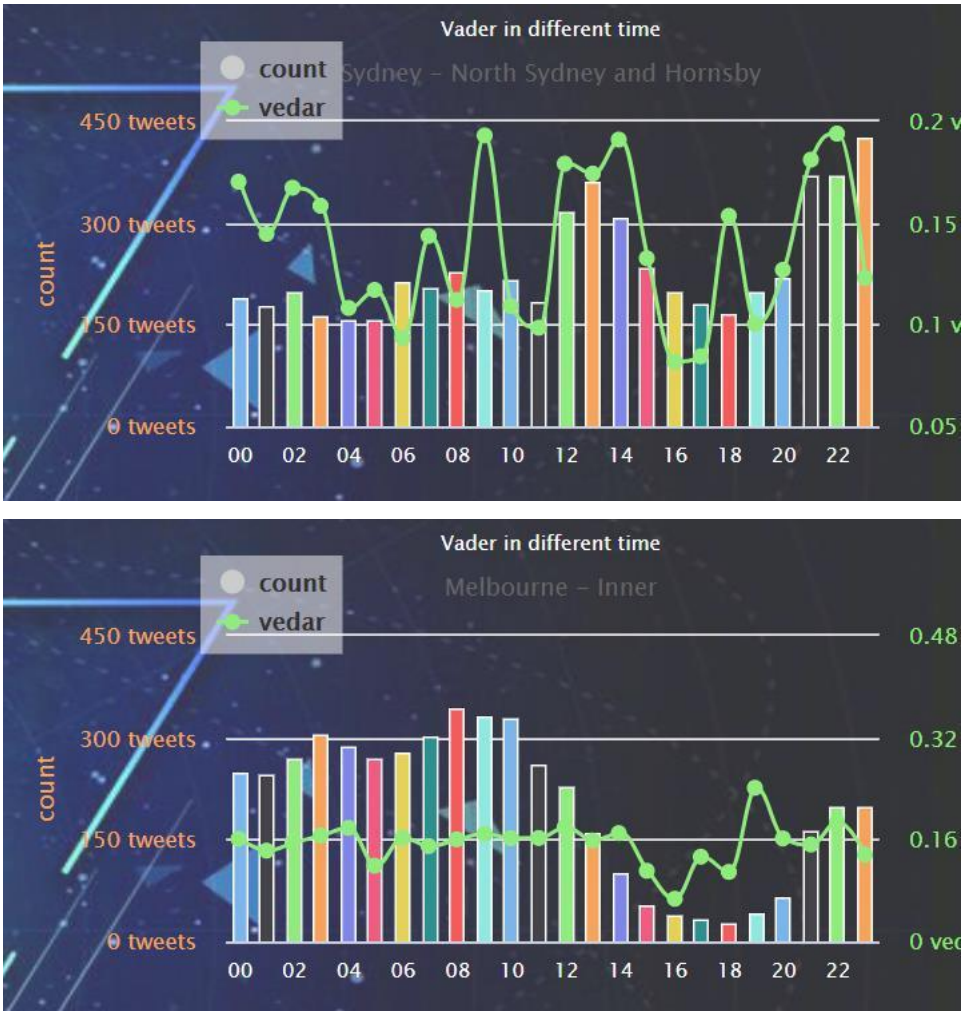


Figure 10: Vader of areas with different income

We estimate that people in high-income areas are not lazy. Their peak time for Twitter is usually in the morning, and the number of Twitters during working hours will be greatly reduced. Through Melbourne and Sydney, we can see that Melbourne conforms to this rule, while Sydney does not.

Chapter 7. Challenges registry

ID	Name	Description
1	First time using UniMelb Research Cloud	Although the team has some experience with Cloud Interfaces such as AWS and Azure, any nuances around UniMelb cloud will need to be understood.
2	CouchDB forgot to change the default password	We used the default password of the CouchDB, which caused the database to be modified by others. Fortunately, we backed up our database in advance, so we re-deploy CouchDB with the new password and upload the backup to the database.
3	Limiting the number of tweets per Twitter developer account	A single account can only return so many tweets until the limit is. To overcome this challenge we made 5 developer accounts to harvest tweets from.
4	Difficult to unify geography granularity between tweets and Aurin data	One key area which took up a lot of time was unifying the Aurin geolocation information with the twitter geolocation information. This is important for visualizing the data in a way that is accurate.
5	For many reasons, created and deleted existing instances many times, which drastically slowed down our development velocity.	We created and deleted existing instances many times for the following reasons: 1.the instability of the network 2. proxy configuration error 3. there are only four instances, so every time you test Ansible, we have to delete the existing instance 4. recording video needs to start from scratch.

Chapter 8. UniMelb Research Cloud Review

The NeCTAR Research Cloud utilizes the OpenStack cloud operating system which is used for controlling, administering and metering large pools of computing, storage and networking resources. This is known as the IaaS – infrastructure as a service cloud model. This cloud offers a range of services, instance types and networking ability. Whilst this discussion will intend to focus specifically on the UniMelb research cloud, many of the pros and cons are a direct result of Cloud computing or the OpenStack platform.

One of the main advantages of this cloud is ease of use. This comes down to a few factors, firstly the dashboard through which you can manage the computing, storage and networking resources of the system is mostly easy to understand and well simplified. Whilst working in a team of developers it was very advantageous to have a 'team' cloud which all members can access. This really speeds up development as it means teams can collaborate well and streamline processes. The use of open source projects with large communities meant that there is a lot of documentation online and useful resources for help, this is another advantage of using the OpenStack system. Further, the availability of the standard images to boot from helps streamline the deployment process. The final main advantage is the API implementation, without this the Ansible and automation of deployment would not have been possible. This is very important as when developing cloud platforms scalability and elasticity are very key characteristics.

There are no major disadvantages encountered during the project. There were some moments of downtime that were noticed however this can most likely be attributed to a large number of users trying to access the clouds resources. It should also be noted that although the web interface greatly simplifies deployment and management of instances, at times it can lag and take a long time to load (minutes).

Chapter 9. Conclusion

This report has demonstrated a working cloud-based system that can be used for the purpose of research and sentiment analysis on social media data. From deployment through to front end visualization the proposed system has been developed with scalability, being able to scale up and scale down the number of CouchDB nodes and also twitter harvesters to meet the number of tweets needing to be processed. This is a key feature as using the cloud can be expensive and using resources efficiently is an important cost factor for research and industry practice.

Secondly, the system has been developed to be highly available and fault tolerant. Multiple CouchDB nodes and multiple harvesters can be deployed and run simultaneously. If one node or instance were to crash this would not affect the entire system. For example, if one harvester were to go down this would not bias the database due to the recursive was the tweets have been harvested. Also, if one CouchDB node was to go down, the entirety of the CouchDB database would still be maintained due to replication.

Lastly the system has been configured and designed in a way which allows for automated deployment through Ansible playbooks. Although this is not necessarily a key characteristic of the cloud, automated deployment greatly improves the ability of the system to be elastically scaled up and down. Further, allows for configuration management of all nodes running across the system.

Reference

[1] Google API for map:

https://developers.google.com/maps/documentation/javascript/tutorial

[2] Twitter Search API:

https://developer.twitter.com/en/docs/tweets/search/overview/standard

[3] Twitter Stream API:

https://developer.twitter.com/en/docs/tweets/filter-realtime/guides/basic-stream- parameters

[4] Vader Score

https://github.com/cjhutto/vaderSentiment

Appendix

YouTube Video:

https://www.youtube.com/watch?v=M2bnRExy_VM&feature=youtu.be

Source code:

https://github.com/410023212/Twitter-sentiment-analysis-deployed-on-NeCTAR-cloud

Work breakdown

Team Member	Responsibilities
Tone He	Architecture design of the system Sentiment analysis design and coding CouchDB deployment on the cloud MapReduce functions Front-end work Documentation
Yao Wang	Harvester deployment on the cloud and design Data pre-processing program Data analysis
Huiya Chen	Ansible Playbook about all the instances creating and configuration Ansible test and debug