Data Analysis

CouchDB

The volume of date and velocity required to import data is crucial in this task. Since the raw data is stored in a 60 Gb json file, we looked forward to a document-based opensource database management system to use. In this project, CouchDB is our choice.

CouchDB is a Document-oriented database management system that can store each tweet in json file as a structured document with unique id. What makes CouchDB user-friendly is that CouchDB has its built-in administration interface named Futon, which is accessed by http://IP_address:_port_number/_utils/#. Users can access many CouchDB's features and it is easy to work with some of the more complex ideas involved. In our project, we created and deleted databases; create and view documents; compose and run views in design document using MapReduce.

Views and MapReduce

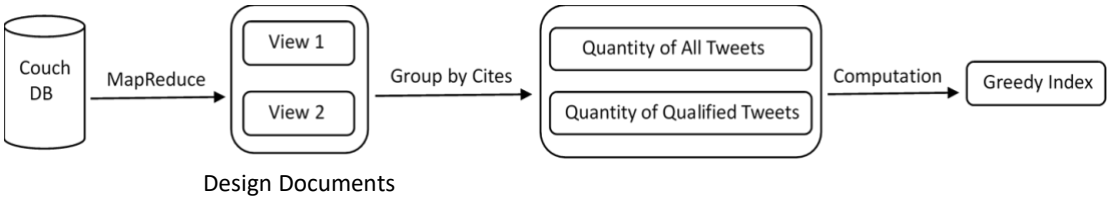

Figure: Data Analysis Process

First, we installed CouchDB in instance 3 and configure it as a standalone mode. Then we build two databases for the tweets. The first one is used to store raw data from Twitter Harvester as a data backup in case of data loss. The second one is used to store Standardized data obtained from preprocessor, on which is the data we do further analysis. When the data was parsing into CouchDB, the duplicated tweets were removed by its default mechanism. Comparing the date size of two database (the first with raw data and the second with standardized data), it is obvious that the size of raw data is as five times as the standardized data.

Second, we created design document and views using MapReduce. In the first view, the data is grouped by the cities in Victoria State and the result returned is the quantity of all the tweets of each city in standardized data. In the second view, the data is grouped by the cities in Victoria State as well and the result returned is the quantity of qualified tweets of each city in standardized data. "Qualified tweets" means that only if the feature we would like to analyze exists, it can be distinguished as qualified. And the exist attribute of the document labels the tweet whether the tweet has the feature we want. If it is true, then it is computed as qualified data, otherwise it is not qualified.

Finally, the greedy index is computed as:

Greedy Index of Each City = Quantity of Qualified Tweets / Quantity of all Tweets

Table 1 shows the views in design document (results) mentioned above.

View		Function	Access URL

all	map	function (doc) {
		if(doc.location != null){
		emit([doc.location], 1);
		}
		}
			http://103.6.254.126:5984/final_result/_design/result/_view/all?group_level=2

	reduce	function (keys, values, rereduce) {
		return sum(values);
		}	

location	map	function (doc) {
		if(doc.location != null && doc.exist){
		emit([doc.location], 1);
		}
		}
			http://103.6.254.126:5984/final_result/_design/result/_view/location?group_level=2

	reduce	function (keys, values, rereduce) {
		return sum(values);
		}	

Table 1: Views

Note: Views can be called by Http in the following format: http://loacalhost:port_number/database_name/_design/design_name/_view/view_name?group_level=2

In our CouchDB, the view is grouped by the city when accessed, thus the result is as follows:

View	Result Example
all	{"rows":[{"key": ["Alpine (S)"],"value":344}, {"key": ["Ararat (RC)"],"value":63}, …}
location	{"rows": [{"key": ["Alpine (S)"],"value":26}, {"key": ["Ararat (RC)"],"value":2}, …}

Table 2: Result Layout

The result in _view/all shows that there are 344 tweets in total in Alpine and 63 tweets in total in RC. And the result of _view/location shows there are 26 qualified tweets in Alpine and 2 qualified tweets in Ararat. Thus the greedy index of Alpine is 26/344 and the greedy index of Ararat is 2/63.