Assignment 3: MapReduce & Tweet analysis
Steven Raaijmakers & Marcus van Bergen
10824242 & 10871993

What is MapReduce?

MapReduce is a programming paradigm which was originally developed by Google. It is a technique whereby a Mapper and a Reducer are used in order to process large amounts of data on a Hadoop based cluster.

The job of the Mapper is to take a dataset and break it down into a smaller dataset. This set of data is formatted in a key/value pair fashion. After the Mapper has processed its input data it passes its output data to a Shuffler. This Shuffler will then pass the data to a Reducer. The Shuffler's job is to modify the data of the Mapper to the Reducer. The Reducer will then write the final data which is a Fold job on the Mapper's output. The power of MapReduce is to be able to process a large amount of data (which is not related to each other) simultaneously.

The Assignment:

The above shines a light as to why our Twitter dataset can be applied to a MapReduce job. The dataset we were given is a bunch of tweets which are not related to each-other. Thus to do an operation on these tweets such as checking the top 10 hashtags in the entire dataset, or finding the sentiment of these tweets is perfect for MapReduce; seeing as these operations can be done individually, and the results summed at the end by the Reducer.

Top 10 Hashtags:

Our approach to finding the top 10 hashtags was very simple. At the Mapper stage we receive a single tweet in API-style format. We start by removing everything of the tweet besides its context, keeping only the 'W' of the tweet over. We then remove accented words in the tweet, so these hashtags are categorised properly; e.g.: ö becomes o. After this we use regex to parse the words and check if a hashtag is found. If so, we send the hashtag to the Reducer. In the reducer the counter for said hashtag is found and incremented with every time that hashtag is found. We were able to find the Top 10 hashtags by running a python script over the collected results of the MapReduce.

Sentiment values & Standard deviation (SD):

We then modified then above code in order to get even more information about these hashtags.

Our objective was to perform a sentiment analysis on each tweet via the StanfordCoreNLP pipeline. This pipeline uses a sentiment model and a parsemodel which is set to English. Therefore we first had to check if a tweet was written in English before we could perform the sentiment analysis to it. After detecting the sentiment of a tweet, we assign this sentiment (an integer) to each hashtag in the tweet.

At first we had to detect if the tweet contained any hashtags at all. After which we try to detect the tweet's language. To increase the accuracy of the language detection of a tweet we stripped the tweet context of mentions, links and other twitter slang (like "RT" or the use of #). If the language of the tweet is English we perform a sentiment analysis on the entire tweet, and then assign the calculated sentiment to every hashtag of the tweet, seeing as each tweet has **one** sentiment.

Optimisation: using a do-while loop instead of a while loop because we know at least 1 hashtag exists. The impact of which may seem negligible, however with a larger dataset this won't be the case.

Top 10 HashTags, Sentiment Values & Standard Deviation

HashTag	Occurrence	Occurrence in English	Sentiment Value	SD (in regards to Sentiment)
#IRANELECTION	32	27	1.222	0.41574
#XIXICOCO	18	5	1.200	0.40000
#HONDURAS	14	3	2.000	0.81650
#FORASARNEY	12	6	1.333	0.47140
#TCOT	10	10	1.300	0.64031
#NEDA	9	7	1.143	0.34993
#FOLLOWADD	9	9	3.000	0.00000
#140MAFIA	9	6	2.500	0.50000
#SPYMASTER	8	7	1.714	0.45175
#FB	7	7	1.429	0.72843

We then send the hashtags and sentiment value to the Reducer. In the Reducer we receive these values and are able to calculate the mean of the sentiment value and the SD.
Our implementation yielded the following results.

There are some interesting observations about this table. The first is that there is a difference in certain hashtags occurring in English. To no surprise **#XIXICOCO** only occurs 5/18 of the time in English seeing as it's a French hashtag. The next is the sentiment value (average). In this case the higher the sentiment value the more positive the context of the tweet. **#FOLLOWADD** and **#140MAFIA** are both found in a positive contexts in comparison to **#IRANELECTION**.

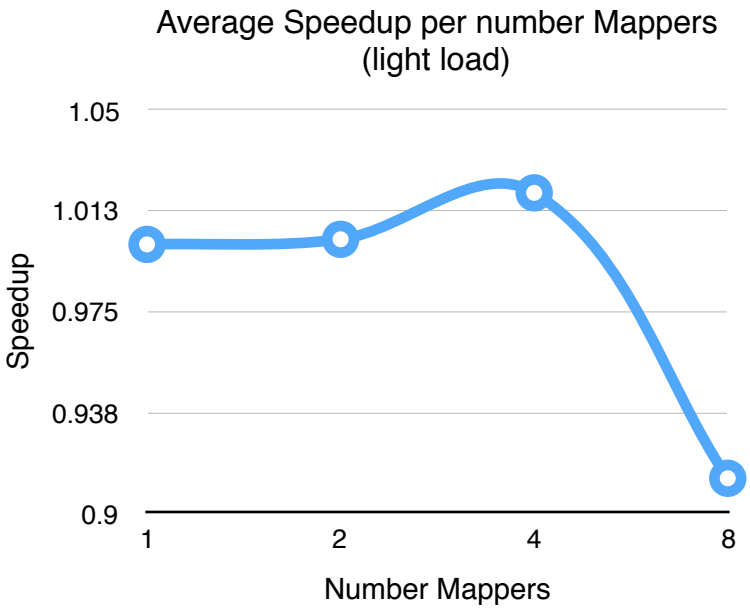
This is where the SD (of the sentiment values) is important. The SD shows us that certain tweets are move controversial than others. For instance **#IRANELECTION** is low in sentiment value, and has a relatively high SD. This means that this hashtag was used in both positive and negative contexts; which is understandable as different people have different opinions about election outcomes. However the hashtag **#FOLLOWADD** has a SD of 0. This is quite obvious because this hashtag was probably used in a movement for a certain person to gain more followers, probably done by fans of said person. Thus there is little to no sway in the found contexts of this hashtag.

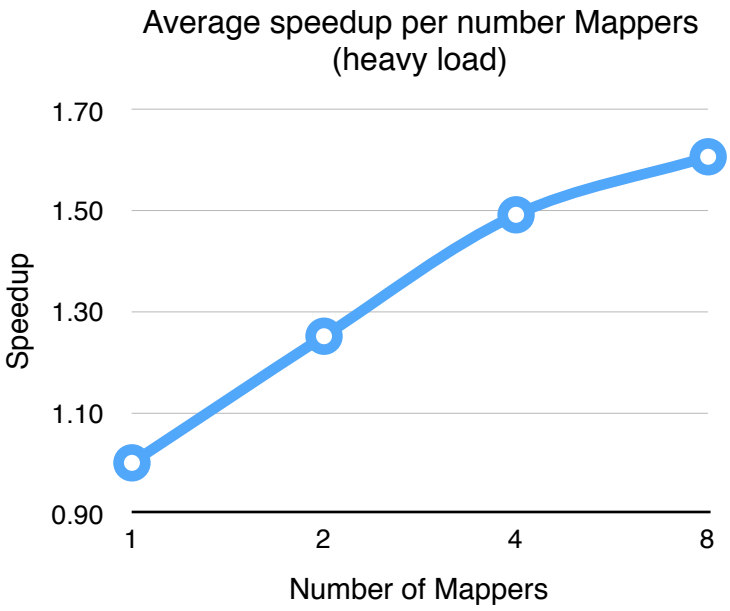
In-depth performance analysis of MapReduce on SURFsara:

Using a python script we were able to execute our sentiment code on SURFsara. We ran four tests with four different number of Mappers (1, 2, 4, 8). Starting a time before the Job.completion( ) function and ending it after we were able to collect the execution time of all these jobs.

The execution times were almost all around 1 minute long; relatively fast as we ran these tests late at night where SURFsara would be under less load. However these average elapsed-times showed that increasing the amount of Mappers didn't per-se lead to more speedup.

Interestingly enough, running the same code when SURFsara was under load (granted *only* one iteration of 1, 2, 4, 8 Mappers) did show more SpeedUp per number of Mappers. Our results yielded the below shown charts:





As mentioned above certain times the amount of Mappers had an effect on the execution time and sometimes not. We believe that the way MapReduce works, it's optimisations and our small data set (which is I/O dependant due to the tweets being on the HDFS) are all factors that produce this result.

In our case the Mappers need to keep pulling data from the HDFS, which means that the biggest bottleneck to the speed at which Mappers can pump data to its Reducer is the network on which the HDFS (Hadoop File System) is running. Because this network is such a great bottleneck, Network speed vs that of a CPU can process, adding more Mappers doesn't really benefit to the access time of the network, because there is a physical limit to this. It might just help when the cluster is under heavier load because these Mappers now have to be scheduled; in which case the CPU speed is (virtually due to scheduling) less and the network speed (should) relatively stay the same.

If our dataset was less I/O bound, we could understand that adding more Mappers could result in a higher speedup; however in our case the HDFS seems to be the bottleneck to our performance gain.

In our testing (four iterations) we did find varying results. Thus we produced the following chart with the error bars. At 8 Mappers, we have a very big error bar, as the SD of the results which we collected is quite large (due to there being such a difference in times in the runs).

Even though, in some rare cases, adding more Mappers results in a lower execution time (higher speedup) the general consensus is that it doesn't particularly help in this application.

This is thus due to what we believe to be the bottleneck of this application in MapReduce, which would have to be the HDFS.