Assignment 1 - Karl Nyrén

In this lab we will go through the basic usage of Hadoop/MapReduce models.

Part 1

Task 1.1

```
$ usr/local/hadoop/bin/hadoop jar /usr/local/hadoop/share/hadoop/mapreduce/hadoop*examples*.jar wordcount input output
```

Questions:

1. Lets look at the directory in output:

```
$ ls /home/ubuntu/output
_SUCCESS  part-r-00000
$ less part-r-00000
"A"	2
"Alpha"	1
"Alpha,"	1
"An"	2
"And"	1
"BOILING"	2
"Batesian"	1
"Beta"	2
"Beta"	1
.
..
...
```

The _SUCCESS file is an automatic output from MapReduce runtime, indicating that everything has gone as planned. The part-r-00000 file is containing the counted words inside the input document, probably only seen as separated by white spaces.

2. Standalone mode is used more for debugging of the process, and is not using HDSF or YARN to create a cluster like environment. Pseudo-Distributed mode is going to act more like an actual cluster environment, initiation a master and slave setup, where the Java process is run on the backend.

Task 1.2

1. The core-site.xml will tell Hadoop where the NameNode is running in the cluster, it is also containing information about the input and output of the Hadoop Core. The file HDFS contains information about how the HDFS daemons are to be run. These daemons are such as the NameNode, secondary NameNode and the DataNodes. It is also here that one can specify replication settings, but if left untouched the default setting is used.

2. So what are the following:

    - NameNode - Manages file system namespace, regulates client's access to files, and in HDFS it executes operations such as naming, closing & opening files and directories.
    - Secondary NameNode - Specially dedicated node in th HDFS cluster that will take checkpoints of the metadata on the NameNode's file system namespace. These are used to help the NameNode perform its work, but it is not an replacement for the main NameNode.
    - DataNode - this is the machinery in HDSF that will store data, and this data store is often replicated amongst multiple Hadoop nodes.
    - JPS - Java Virtual Machine Process Status Tool, which is a command to check all Hadoop daemons running on the current machine.

Task 1.3

1. There are two found classes inside WordCount.java; Map and Reduce.
    - Map: the map class has at purpose to mark some value with a key
    - Reduce: will use the keys from the mapper in order to aggregate/reduce the data in some manner.
2. HDSF - Hadoop Distributed File system - is the horizontal data storage in Hadoop applications. HDSF can be accessed by multiple slave nodes in order to compute and is also distributing its data across multiple DataNodes to optimize storage.

Task 1.4

We were to extract the number of times a character started a word using the altered version of WordCount java script. The altered version can be found here, and the results from reading in the example text is plotted below in the graph.

As we can see the Java code is not sensitive to what a definition of a character is, it accepts special characters such as #, [etc. and only start a new word on blank spaces.

Part II

1. I would say that they are semi-structured data in the sense that their meta data is structured - such as date posted, the id etc. - however the texts themselves will be very unstructured. They key part of using RDBMs is to be able to store the items with their unique keys, however if we want to analyse the tweets we cannot get any further than the meta-data in an RDBM before things starts to get very slow. By using key values and MapReduction one could analyse the content as well without slowing down the process to much.

The task is to extract the number of occurrences of the Swedish pronouns han, hon, det, den, denna, denne and hen in tweets. We are to only use the unique tweets and normalize the counts of the pronouns by the number of tweets. The mapper and reducer can be found here and the result gained from the analysis of the Twitter data supplied to us is plotted below.

Part III

```
hive> CREATE EXTERNAL TABLE if NOT EXISTS Tweets (
    > text STRING,
    > retweeted_status STRING)
    > ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe' STORED AS TEXTFILE;
hive> LOAD DATA LOCAL INPATH '/home/ubuntu/tweet_data/tweets/files/*' OVERWRITE INTO TABLE Tweets;
hive> select noun, count(noun)
hive> select count(*) from Tweets where retweeted_status IS NULL AND LOWER(text) rlike '\\bhon\\b'; # select texts with hon
# The same query was repeated for all pronouns, then normalized using
hive> select count(*) from Tweets where retweeted_status IS NULL; # Count number of unique tweets
```

The number of tweets gained containing the pronouns came out different from the streaming framework. This is most likely due toS that my python code did not stop when it encountered one noun, instead it counts all occurrences of the nouns in a tweet, whilst hive will return when it finds the word once in the text string, thus counting it more properly in my opinion. I know very well that some retweeted_status might contain an original text in the tweet as well but I count that every retweet regardless of added text is a non-unique tweet.

1. Comparing the knowledge of language is less when it comes to Hive in the beginning. However, when you get into more serious queries it does get more complicated. The conventionality of Hive can be attractive when you want to present a solution to repetitive queries, since hive works through indexing and thus builds up in speed. I unfortunately don't have the stats of the speed for the jobs done in this example, but from experience I can tell that the Hadoop/MapReduce approach was significantly faster than Hive doing this particular task. This is probably due to the structure of the data we are working with, and that hql is optimized for structured data. One other difficulty that can be seen with Hadoop/MapReduce is that you really need to stick to that mode of thinking, meaning you one want a tuple, with a key and a value. With Hive on the other hand one can break free from this term of thinking and focus directly on the query you want to perform which really shows the perks of having a higher tier interface for Hadoop. What you give up by using Hive is the in depth optimization that can be done in the search steps, and the number of ways a MapReduction can be done is near to endless. I do strongly believe in that using Hadoop/MapReduce to begin with is a better way to learn how the framework is set up, but for commercial applications Hive shines in less requirement of problem solving when doing queries. Hive would also be better applicable if you know that the format of your data will remain constant, thus you never need to change queries and make the user interaction seem seamless with their previous work-process. One could argue that this is true for Hadoop/MapReduce too, however if you even slightly want to change the query a whole new mapper and reducer needs to be created, whilst hive simple requires you to create a new query. In terms of setup both of the services had their own issues, but in the end hive was an odd user experience with the database that it sets up. Without being careful one could clutter up their folders with hive metastores.
2. Pig is a script based program, based on the language Pig Latin. This language is resembling hive in terms of the simplicity. One could say that it is even more simplistic. Pig works different from hive in the way that you can either interact with it directly, as done in hive, or you can submit batch scripts to perform tasks. It will use mapreduction but is able to include frameworks of Spark or Taz if it suits the task. It is a bit closer to Hadoop/MapReduce in the way that you can transform the data inside the script, and then print a bit of an different output. Sure, one could make very intricate nested queries in hive, but they do become messy quite quickly. One might argue that when doing Pig queries you need to load the data every time, which in batch state could be troublesome. However, Pig is able to use spark dynamic allocation, enabling Pig to do multiple processes at the same time and reallocate the resources for different scripts accordingly.

3. I believe that NoSQL solutions could be efficient if we are limited by main memory. Since the data we want to analyze is semistructured NoSQL should not have any issues with processing the data. Since the data can be split up into many smaller chunks we would not have to worry about the loading of the data into main memory when we would search through the tweets.