# A1

# Part 1

## Task 1.1

1. Look at the contents of the folder output - what are the files place in there? What do they mean?

There are two files: \_SUCCESS and part-r-0000. The first file is a file generated by the MapReduce program in order to let the user know that the run was successfull. The second file is the wordcount itself, with the key, value pairs corresponding to the word found on the text followed by the number of occurrences.

1. In this example we used Hadoop in Local (Standalone) Mode. What is the difference between this mode and the Pseudo-distributed mode?

The standalone mode is supposed to work by itself as a single Java program on the machine. It uses the machine FileSystem.

The pseudo-distributed mode uses Hadoop as a daemon on the machine and it uses several nodes to work, simulating a pseudo-cluster. There may be different Hadoop daemons working at different VM but on a single machine. It uses Hadoop’s File System (HDFS).

## Task 1.2

1. What are the roles of the files core-site.xml and hdfs-site.xml ?

core-site.xml passes the basic configuration for the NameNode running in the cluster to the Hadoop daemon. Between this information we can find the port number, ip address as well as the temporary directory path.

hdfs-site.xml contains required configuration for HDFS daemons. That is the directory path for hdfs, blocksize among other configurations. The HDFS daemons are the DataNode and Secondary NameNode.

1. Describe briefly the roles of the different services listed when executing ‘jps’.

$ ~ jps  
#output  
17281 Jps  
16738 SecondaryNameNode  
16548 DataNode  
16394 NameNode

* jps: JVM Process Status Tool, checks the Hadoop daemons currently working.
* NameNode: Master node, responsible for storing the metadata for files and directories.
* DataNode: the node containing the actual data.
* Secondary NameNode: node responsible to stabilize the system in case the NameNode breaks. It merges information using the edit log.

## Task 1.3

Answer the following questions:

1. Explain the roles of the different classes in the file WordCount.java.

-WordCount: entry point to the program. It contains the main() method which is the first code to be executed.

-TokenizerMapper: Map class, it takes a line as a string, separates it into words/keys and assigns a 1 to each one of these. It creates a tuple (key, 1).

-IntSumReducer: it groups the tuples by key and adds the values until it reaches the number of occurrences of the given word.

1. What is HDFS, and how is it different from the local filesystem on your virtual machine?

On one hand, HDFS is a distributed file system used by the Apache Hadoop project. Distributed file systems rely on having its components spread across multiple machines. The purpose of DFS is to allow users of physically distributed computers to share data and storage resources by using a common file system. In the case of HDFS, the file system stores the data across multiple nodes in a distributed fashion as well, having a master(NameNode)-slave(DataNodes) architecture. The function of each of these components has been explained on Task 1.2.

On the other hand a local file system is the way the OS of a given machine stores the data on disk.

## Task 1.4

For source code see attached file FirstLetterCount.java.

The output of the MapReduce job is the following:

A,12898 B,4861 C,4299 D,2743 E,3492 F,4130 G,1770 H,3106 I,8691 J,344 K,465 L,2689 M,4819 N,2063 O,9720 P,3880 Q,173 R,2505 S,7411 T,18600 U,1110 V,936 W,5886 X,32 Y,479 Z,55

Now we can plot the frequency for each letter (for source code see attached file FirstLetterPlot.py).

A screenshot of a cell phone

Description automatically generated

# 

# Part 2

1. Based on the documentation in the above link, how would you classify the JSON-formatted tweets? Structured, semi-structured or unstructured data? What could be the challenges of using traditional row-based RDBMs to store and analyze this dataset (apart from the possibility of very large datasets)?

As well as other JSON files, tweets can be classified as semi-structured data. This is because they don’t have the usual formatting for structured data (data tables, databases…), instead the file has tags or identifiers that separate the different attributes of a tweet, allowing us to analyze the data.

If we wanted to store this data in a relational database we would need to specify the relationships between the different attributes and entities. Since our data is semi-structured this is not straight forward. The low latency of RDBMs would be an advantage to consider for choosing this type of storage but since in this application (as well as most Big Data applications) latency is not a constraint latency is not worth considering. Scability would be another problem, we are now dealing with a small amount of data (~10Gb) but what if the size increases by a 100 factor? We need to be able to scale horizontally. With RDBMs this is not possible, they make strong guaranties about consistency (ACID properties) requiring communication between different nodes for every transaction. Scaling out, i.e. adding more nodes would need more communication, siignificantly slowing down the processes. Instead, by using HDFS we are able to scale horizontally in a fault-tolerant, high throughput, cost effective and flexible way.

In this part of the lab twitter JSON data was analyzed in order to count the occurrences of the swedish pronouns ‘hon’, ‘han’, ‘den’, ‘det’, ‘denna’, ‘denne’ and ‘hen’. With this purpose Hadoop’s Streaming API was used with Python code. Attached to the report is the file mapper.py (attached to the report). This files prints as standard output a tuple containing the pronoun found and a value of 1 each for each occurrence of the pronoun in the text of a unique tweet. The reducer.py will catch this standard output and add values by key. The final result is the following:

count,2341581  
den,1311322 denna,22582 denne,3985  
det,527481  
han,770154  
hen,33114  
hon,358737

Note that I added an extra touple (count, value) representing the number of unique tweets. This will be used in the plotter.py in order to visualize the frequencies of each pronoun in the dataset.

A screenshot of a cell phone

Description automatically generated

In the plot we can see that the most common pronoun is ‘den’ followed by ‘han’ and ‘det’. The least common pronoun is ‘denne’. The analysis was done on 2341581 unique tweets.

# Part 3

1. Briefly discuss and compare the experiences of using Hive vs. ‘vanilla’ Hadoop/MapReduce and Hadoop streaming. Focus on the user experience. Since we are working on a toy-sized dataset, performance comparisons become a bit shady, but as long as you are aware of that you can still measure and report it. The answer could be 0.5-1 A4 pages, 12pt font.

The steepest part of Hadoop’s learning curve was to understand how it works, what are the different Daemons and how to properly configure it. The logic behind the MapReduce framework is pretty straightforward and is was not hard to think for a solution to the problem algorithmically speaking. The fact that Hadoop’s Streaming API can work with standard input and standard output makes everything easier because I could use whichever programming language I’m more familiar with. The hardest part was to ‘get to know’ the data we were given and how to parse and load it, but this has nothing to do with Hadoop, instead it’s something you have to always deal with in Big Data analysis. There were other difficulties on the way not related to Hadoop such as networking and cloud related terminologies and protocols.

The Hive approach was hard conceptually, since it uses SQL language to do the MapReduce jobs it can easily be confused with RDBSMs. Having to define a schema before loading the data into Hive was another issue since you have to translate you JSON structure into HiveQL. The configuration step was hard as well because we were left alone in the raw documentation, which is more like a real work-case scenario. The writing of the queries although not straight forward, was easier due to the previous knowledge in MySQL. This requires to use a database to store the tables, Derby by default in Hive.

The use of Hive in our data felt a bit rough since it is semi-structured JSON. In addition we were trying to parse text within Hive, which is classified as not structured data, which makes the parsing with Hive’s query language difficult and not straight forward as well as not efficient.

When it comes to performance comparisson I noted while querying the data that Hive was significantly slower than Hadoop’s MapReduce. I can’t give specific stimations since I didn’t plan to test the performance when coding the queries and didn’t time the ‘vanilla’ Hadoop job.

1. Pig (https://pig.apache.org/) is another commonly used part of the Hadoop ecosystem (we will see a third alternative, Spark, in Lab 3). Briefly discuss, contrast and compare Pig vs. Hive. Note that you are not required to repeat the task using Pig.

First of all both approaches work on top of Hadoop’s MapReduce. In addition Apache Pig relies on Pig Latin as a programming language which relies on scripts with functions and mehtods while Apache Hive uses a HiveQL declarative language which will be familiar with developers working with databases. Furthermore, Apache Hive has better access choices and features than the oens in Apache Pig. However, Apache Pig works faster than Apache Hive. In addition Hive can be used for both semi-structured and structured data while Pig can only be used for structured data. As a conclusion I’d say that Hive is more suitable for complex as well as nested data structure but Pig is ideal for Batch Processing, such as OLAP.

1. In this assignment, the Twitter dataset was quite modest in size, only ~10GB of raw JSON data. Here, we could even load it in main memory on a high-end workstation. For sake of argument, assume that it had been 10 times as large, say ~100GB, it would still not have been a very big dataset, but too large to manage in a naive way. During Lectures 2 and Lectures 3 you heard about other types of data management systems supporting analysis. Which of these tools could have been efficient and productive for analyzing this specific dataset?

The main property we are looking for in this case scenario is horizontal scalability in order to process the huge amount of data. Given this premise we can choose either a storage based on a distributed file system such as HDFS (using Apache Hadoop, for instance) or a storage based on NoSQL (Cassandra, for instance). At first glance both technologies seem similar, but while NoSQL is a distributed database infrastructure, Hadoop uses HDFS, which is a file system that allows for massively parallel computing.

The decision of choosing between these two will depend on the job we are dealing with. If we can't lose any information during the processing, don't care about the latency and don't need streaming, Hadoop's HDFS is probably the best option. On the other hand, if we care about latency and can't wait for a Hadoop job to finish it's better to consider NoSQL, which is meant for real-time interactive access to data. As a conclusion I'd say that NoSQL will be a more suitable option since we are dealing with 100Gb of files and Hadoop is not showing its potential yet at this file size. Instead, NoSQL will as well easily handle this amount of data and will be quicker than running Hadoop in batch. If we choose NoSQL we will be able ti handle streaming data, which is a common operation with social media APIs.