A2 - Hadoop and MapReduce

Introduction:

The objective of this assignment is to gain familiarity with the MapReduce programming model using the Hadoop framework.

To pass the assignment (and thus gain 2 points) you need to complete all tasks in part 1. Students who wish to try for 3 or 4 points should also hand in part 2.

You will be required to hand in a **report** for A2 that should contain answers to the questions in this document as well as plots displaying your results and a discussion of your results.

The code that you are instructed to hand in (specified at each task) should be attached **as separate documents** and handed in together with the report in Studium. Upload single file in Studium if possible, avoid compressed file.

Be advised that you will have to consult lecture notes and external resources to complete the assignment. Some useful links are provided, but you will also have to search for additional information on your own.

Part 1

Task 1. Introduction to the Hadoop Framework

Task 1.0: Setup VM with Hadoop

Follow the instructions below to start a new VM, and review A1 if needed.

Under Compute-Instances, click Launch Instance

1. Use the source image: “Ubuntu 20.04 - 2023.12.07”
2. If you plan to work on part 2, set Volume Size to 40GB; otherwise no change is needed.
3. Select “Yes” to “Delete Volume on Instance Delete”.
4. Use the flavor ssc.medium.
5. Use the security group “default”.
6. Your instance name should contain your own full name so the owner can be identified.
7. After the instance has been launched: go to the “Volumes” menu in the OpenStack dashboard and locate the volume attached to your instance, and rename it with your full name.

Setting up your VM:

1. For Hadoop to work in all examples below, you need to set the hostname of your instance in /etc/hosts. Print the hostname of your instance with command: `hostname`. Open the file (you need to be sudo to edit) and modify the first line:

   ```
   127.0.0.1   localhost <your hostname>
   ```

   Tip: use nano/vim editor.
2. Run apt update, and install the packages `openjdk-17-jdk-headless and net-tools`
3. The JAVA_HOME environment variable is used by applications like Hadoop to find the Java installation folder. We need to set this variable. To apply it in future restarts, add this line to /etc/environment :

   ```
   JAVA_HOME=/usr/lib/jvm/java-17-openjdk-amd64/
   ```

   And, to apply the environment variable for this shell session without rebooting:

   ```
   export JAVA_HOME=/usr/lib/jvm/java-17-openjdk-amd64/
   ```

   Check if the environment variable is successfully set by command:

   ```
   $JAVA_HOME
   ```

4. Use `wget` to download the binary release of version 3.3.6 of Hadoop (hadoop-3.3.6.tar.gz, not the “aarch64”). Find the download link on the Hadoop website. Use the `tar` command to extract the archive directly into your home directory /home/ubuntu (pay attention to the different attributes of `tar`, check `tar --help`).
5. Verify that you can query the Hadoop version without error:

   ```
   hadoop-3.3.6/bin/hadoop version
   ```

Task 1.1: Word count example in local (standalone) mode

This task requires successful execution of a basic Hadoop program. The Hadoop framework executes programs using MapReduce. The code to be run is available as a test example shipped with the Hadoop libraries. As a reference to Hadoop MapReduce, consult:

https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

Use this tutorial (and other necessary resources) to understand the executed commands for this lab and to answer the questions.

1. Create a folder “input” in your home directory (/home/ubuntu) on your instance using command `mkdir`.

2. download the following data file and place it in that directory
   http://www.gutenberg.org/ebooks/20417.txt.utf-8
3. Then go back to your home directory to execute the following command to run the canonical MapReduce example:

```
hadoop-3.3.6/bin/hadoop jar ./hadoop-3.3.6/share/hadoop/mapreduce/hadoop-
mapreduce-examples-3.3.6.jar wordcount ./input ./output
```

Hint: Copying commands from a pdf file might not work. If you get errors, try to type in the command instead. It should all be on one line.

Note: the folder names "input" and "output" are arbitrary - you can use any names you like for the input and output directories. But the "input" folder should contain the downloaded file. Note that the output folder is created by Hadoop, so you should not create it manually before running the command above.

4. Answer the following questions:
   a. Look at the contents of the folder "output", what files are placed in there? What do they mean?
   b. Looking at the output files, how many times did the word 'Discovery' (case-sensitive) appear? (hint: commands `grep/cat` could be useful)
   c. In this example we used Hadoop in "Local (Standalone) Mode". What is the difference between this mode and the Pseudo-distributed mode?

Task 1.2: Setup pseudo-distributed mode

1. Follow the instructions in the link below to set up Hadoop in pseudo-distributed operation. Execute **ONLY** the instructions for **Configuration** and step **1 ("format the namenode")** of "**Execution".**

Hint: In the instructions, the hadoop folder paths given are relative to the installation directory. In our case the installation directory is '/home/ubuntu/hadoop-3.3.6'. **You will have to modify the given commands accordingly.** When modifying the xml config files, **ONLY** edit the part with <configuration> </configuration>, leave other parts same as it was.

Warning: Do **NOT** format the namenode more than once, otherwise there will be multiple clusterID, and you will face problems with starting the namenode**.**

https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html#Pseudo-Distributed_Operation

Then, rather than starting the daemons with SSH it is easier to simply start them manually. For this toy example our node (VM) can serve both as datanode and namenode:

```
hadoop-3.3.6/bin/hdfs --daemon start namenode
```

```
            hadoop-3.3.6/bin/hdfs --daemon start datanode
```

2. To verify that services are running, you can use the command 'jps'. You should expect to see an output like this (the numbers are the process IDs (PIDs), these will be different in your output.):

```
      $ jps
      17922 NameNode
      18047 DataNode
      18383 Jps
```

3. HDFS Web GUI - (Optional). List the TCP ports opened by Hadoop on your system (Hint: `netstat -tna`). On port 9870 there should be a Web GUI for HDFS running. Unlike the Python notebook (which has a token), the web GUI is not protected. Anyone in the world would be able to upload files onto your machine! So, use this site to find out the public IP address of your laptop: https://www.whatismyip.net/ , and then configure your security group to allow ingress only from that IP address (instead of 0.0.0.0 - which means "any"). (Alternatively, use SSH port forwarding to avoid the need to open the firewall at all).

Do not leave the port open to the world!

Then, open it in your browser, and explore: http://130.238.xxx.yyy:9870
You can refer back to the GUI in the subsequent questions.

4. Answer the following questions:
   a. What are the roles of files `core-site.xml` and `hdfs-site.xml` ?
   b. Describe the different services listed when executing 'jps'. What are their functions in Hadoop and HDFS?

Task 1.3: Word count in pseudo-distributed mode

Now we will use Hadoop in pseudo-distributed mode to build and run our own version of the MapReduce example.

0. Create a file WordCount.java from the example source code on this page:

https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html#Source_Code

1. To compile the wordcount example and make a jar file:

```
javac -cp `/home/ubuntu/hadoop-3.3.6/bin/hadoop classpath` WordCount.java
```

```
jar -cvf wordcount.jar *WordCount*.class
```

Obs: *Copy and paste this command might not work. The backquotes in the second command are supposed to be the grave accent diacritical mark, it encapsulates the inner command so that the inner command in backquotes will be executed first.*

You should now have a file “wordcount.jar” in your folder.

2. Next, load the text file, whose words we are counting, into HDFS. The file is in the folder `/home/ubuntu/input`. You can stage the input folder in HDFS by first creating the same user path in HDFS and then copy the local folder into HDFS:

```
      hadoop-3.3.6/bin/hdfs dfs -mkdir -p /user/ubuntu

      hadoop-3.3.6/bin/hdfs dfs -put /home/ubuntu/input
```

3. Now you should have added the text file from the file system of your VM to HDFS. Use the command below to verify that the file is indeed in HDFS:

```
      hadoop-3.3.6/bin/hdfs dfs -ls input
```

You should see an output that looks something like this:

```
      Found 1 items
      -rw-r--r--   1 ubuntu supergroup     674684 2024-01-26 15:41
      input/20417.txt.utf-8
```

4. Run the following Hadoop command. Name your own `<output_dir>`.

```
      hadoop-3.3.6/bin/hadoop jar wordcount.jar WordCount input
<output_dir>
```

If the MapReduce job completes normally, verify that the output looks as expected. First check the content of the output directory in hdfs,

```
      hadoop-3.3.6/bin/hdfs dfs -ls <output_dir>
```

5. Then check the content of the output file using the ‘-cat’ argument of HDFS. Verify that it contains the same word counts you saw before when running the word count program locally.

```
      hadoop-3.3.6/bin/hdfs dfs -cat <output_dir>/part-r-00000

      hadoop-3.3.6/bin/hdfs dfs -cat <output_dir>/part-r-00000 | grep -w
Discovery
```

6. Answer the following questions:
    a. Explain the roles of the different classes in the file WordCount.java.
    b. What is HDFS, and how is it different from the local file system on your VM?

Task 1.4: Modified word count example

1. Modify the above example code in WordCount.java so that it, based on the same input files, counts the occurrences of words starting with the same first letter (ignoring case). The output of the job should thus be a file containing lines like (if there were 1234 words beginning with A or a):

```
a 1234
b …
c …
```

2. Make a plot that shows the counts for each letter (i.e. excluding all non-letter symbols) and include that in your report. *Do not include the entire output file.*

Tips:

1. To copy data from HDFS to local filesystem, consider the HDFS argument '-get'.
2. For the plotting you might need to transfer data between your local computer and your instance through ssh, for this there is a command called "scp" (Secure Copy). Don't forget that you need your key (-i option)
3. In Java there is a useful class called "Character" that might come in handy.

Hand in the following code:

- A FirstLetterCount.java file with your modified codes and sufficient comments.

Task 1.5: NoSQL and MongoDB

Background

NoSQL stands for "Not only SQL" and consists of database solutions that don't use the traditional relational model of data in the form of tables. NoSQL databases are frequently used within big data analytics and are especially suited for semi-structured and unstructured data. They are distributed solutions, scale well horizontally and are open-source. One example of NoSQL is the document-store MongoDB which stores data in JSON-like documents (https://www.mongodb.com/).

Answer the following questions:

1. One example of JSON formatted data is Twitter tweets:

https://dev.twitter.com/overview/api/tweets. Based on the twitter documentation, how would you classify the JSON-formatted tweets - structured, semi-structured or unstructured data?

2. Elaborate on pros and cons for SQL and NoSQL solutions, respectively. Give some examples of particular data sets/scenarios that might be suitable for these types of databases. (expected answer length: 0.5 A4 pages)

Part 2

(This part of the assignment is only required if you want to collect points toward higher marks)

Task 2.1: Analyzing twitter data using Hadoop streaming and Python (awards up to 1 point)

Background

While Hadoop/MapReduce is based on Java, it is not necessary to use Java to write your mapper and reducers. The Hadoop framework provides the “Streaming API”, which lets you use any command line executable that reads from *standard input* and writes to *standard output* as the mapper or reducer. The following tutorial, although a bit old, provides an excellent introductory example to using Python and Hadoop streaming:

http://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/

Important notes:

1. The tutorial above uses Python2, which is no longer maintained). You should use **Python 3.**
2. The tutorial above is **not** using Hadoop 3. Although you are expected to follow these steps, please note that there could be some subcommands/options in the tutorial deprecated in Hadoop 3.
   You should manage to fix them by reading error/warning messages.
   Refer to the latest version of the Streaming API docs:
   https://hadoop.apache.org/docs/current/hadoop-streaming/HadoopStreaming.html
3. The location of the jar file for the Hadoop streaming API is in a different location than in the tutorial. On your distribution it is located in **hadoop-3.3.6/share/hadoop/tools/lib/** (and NOT in ./hadoop/contrib/streaming/)

Data

In this part of the assignment, you will analyze a dataset of about 5.000.000 Twitter tweets collected using Twitter’s datastream API. The total size of the dataset is still a modest ~9GB uncompressed.

Each tweet is a JSON document http://en.wikipedia.org/wiki/JSON. JSON is one of the standard Markup formats used on the Web. For the specific case of Twitter tweets, you can read about the possible fields in the documents here:
https://dev.twitter.com/overview/api/tweets

Download the dataset from:
https://uppsala.box.com/shared/static/1qain2m8welz0n7neolfsnr9j5lvrf39.tar

This particular Twitter dataset was collected by filtering the stream of tweets to store those containing the Swedish pronouns “han”, “hon”, “den”, “det”, “denna”, “denne”, and the gender-neutral pronoun “hen”.

Do the following:

1. Use Python and the Hadoop streaming API to count the **number of tweets** mentioning each of these pronouns.
    - Use the tutorial linked above (minding the Important Notes)
    - In this analysis, only unique tweets should be taken into account, i.e. ‘retweets’ should be disregarded (hint: ‘retweeted_status’).
    - The count should be case-insensitive, i.e. “Han” and “HAN” should count as mentions of the pronoun “han”.

2. Plot a bar chart visualizing the counts of each pronoun.
    - The counts that you plot should be expressed as a percentage of the total number of unique tweets.

Hint 1: Use the Python library ‘json’ to parse the tweets.
Hint 2: It can also be good to look up regular expressions for this task. There is a library in python called ‘re’.

Hand in the following code:

- mapper.py and reducer.py files with your map code and reduce code. Include descriptions about your implementation and the results in your report.

Task 2.2: NoSQL and MongoDB (awards up to 1 point)

Do the following:

1. Redo the analysis from Task 2.1, now using MongoDB. (You need to install this yourself).
    - Use the exact same data as in Task 2.1
    - Some suggested options include using the Mongo Shell or PyMongo
    - This is an open problem, it is up to you to experiment with MongoDB and develop a solution to the problem that makes efficient use of MongoDBs capabilities. Add complexity to the problem if you desire. Note that there is no requirement to use a specific language or interface - the implementation is up to you.

- Present plots corresponding to those you obtained for Task 2.1

2. Answer the following question:

   Motivate your chosen implementation and how you did it. What are some pros and cons of the MongoDB solution compared to the implementation you did in Task 2.1?

Hand in the following code:

- All code for your MongoDB implementation, as well as all commands/queries used to obtain the results (e.g. Mongo shell queries).