Large Datasets for Scientific Applications: Assignment 1

The objective of this assignment is to gain familiarity with the MapReduce programming model using the Hadoop framework. To pass the lab (and thus gain 1 point) you need to complete and present part I and part II. Students who wish to try for 2 or 3 points should also hand in part III.

Your report should contain a short introduction to the problem for each part (I, II, and III), answers to the questions in this document as well as plots displaying your results and reflections on those.

The code that you are instructed to hand in (in the end of each part) should be attached as separate documents and handed in together with the report in Studentportalen.

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

Part 0: Getting started with Git

Git is an open-source version control system used to keep track of changes to code. Together with web-based hosting services such as GitHub or BitBucket, it can also be used for remote access control and various collaboration features.

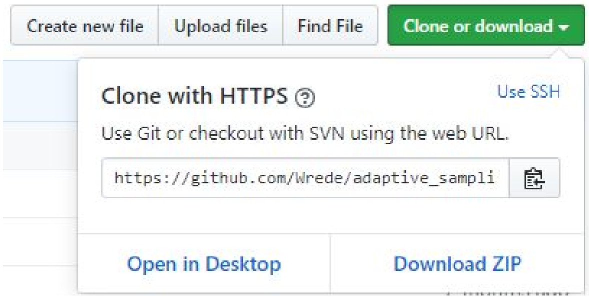
For all assignments of this course, as well as the project, you are required to use Git. This allows you to:
a) Have your code protected in case the instance fail.
b) Work on code on your private computer (and therefore use any code editor you like) and subsequently sync the code on the virtual machines. On the instances, you can use terminal based editors like vim, but it takes some effort to learn how to use them effectively.

We will give a very short introduction to a few Git commands that are necessary, but for further learning we recommend
the book: https://git-scm.com/book/en/v2
and the documentation of Git: https://git-scm.com/docs

The learning curve for Git is steep, but it’s extensively used both in industry and academia.

Instructions:

1. Create a GitHub or a BitBucket account (https://github.com/ or https://bitbucket.org/product/). Both offer private repositories, but GitHub also offers student accounts**.**
2. Login to your account and create a new **private** repository with a convenient name (e.g LDSA). **For security reasons:** make sure that the repository you create for the LDSA course is private.
3. Copy the web URL of your new repository. In GitHub this can be done as the figure below demonstrate



4. Clone the repository onto a local computer (your private one or a lab computer). Unix-like systems have git installed by default (including WSL). For Windows you have to install it.

   Obs: if you are using WSL do NOT alter or edit files that lie inside the WSL filesystem using a Windows editor (might cause corrupted files). However you can modify files in Windows filesystem using WSL. Therefore, we recommend you to git clone inside the Windows filesystem (/mnt/c/) and then use Windows editors for your coding.

   After Git has been installed, in a terminal on your local computer, run the command:

   ```
   git clone <repository-url>.
   ```

   A folder (the repository) with the same name now exists. If you look inside it, there is a hidden folder named .git

5. Add a file and commit it to the repository.
   Create a new folder (e.g A1) inside your repository and create a file inside it (e.g LDSA/A1/test.txt). In your repository folder LDSA, execute the command:

   ```
   git status
   ```

Notice the new folder and the file under “untracked files”. To add it to the “staging area” run the command:

```
git add A1/test.txt
```

Changes to files that have been staged can be saved to the repository by committing:

```
git commit -m “My commit message”
```

(The message should describe the changes made, e.g. “Added a file called test.txt”.)

To see the commit history run the command:

```
git log
```

press “q” to quit.

6. To sync your changes to the GitHub or BitBucket repository run the command:

   ```
   git push
   ```

   type in your account username and credential. Go to your GitHub/BitBucket account and notice that the new folder and file is in your repository.

7. In the next part of the assignment, you will launch a virtual machine and sync the repository to it.

Part I: Intro to HDFS/Hadoop and Wordcount

For this task you will boot a new instance. Follow the instructions below, and review Lab 1 if needed.

1. Use the source “Hadoop-Lab2-volume-snapshot” snapshot.
2. Use the flavour sss.small.
3. Select “Yes” to “Delete Volume on Instance Delete”.
4. Use the security groups “default” and “ldsa-lab2”.
5. Your instance name should contain your own full name so the owner can be identified.
6. 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.

Syncing Git:

On your virtual machine, clone the repository you created in Part 0.

You should be able to see your file test.txt

To update the repository with the latest commits that have been pushed to the repository, run the command

```
git pull
```

We recommend editing code on your local computer, pushing to GitHub/BitBucket and syncing the changes to your virtual machine by using the git pull command.

You can of course also edit code directly on your virtual machine. In this case: make sure to frequently make commits and push them. The virtual machines are not always reliable and if they die you may lose code that has not been pushed to the remote repository.

The snapshot you used to create the virtual machine is a customized image containing a complete Hadoop 2.9.0 framework, see:
https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html

Hadoop is installed in "/usr/local/hadoop"

The executable binaries are in the directory "/usr/local/hadoop/bin". If you like, you can add this directory to the environment variable PATH, but it is not necessary.

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

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

Task 1.1 (Word Count Example)

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 provided links) to understand the executed commands for this lab and to answer the questions.

Create a folder “input” in the directory “wordcount” located in the home directory of your instance. Then download the following data file and place it in that folder:

http://www.gutenberg.org/ebooks/20417.txt.utf-8

Tip: On Ubuntu linux you can use the ‘wget’ command to download data over http.

Then execute the following command to run the canonical MapReduce example:

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

Note that 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.

Answer the following questions:

1. Look at the contents of the folder “output” - what are the files place in there? What do they mean?
2. 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:
Follow the instructions here to set up Hadoop in Pseudo-distributed Operation: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html . Note that all hadoop folder paths are relative to the installation directory. In our case it is ‘/usr/local/hadoop’. To verify that services are running, you can use the command ‘jps’

```
$ jps
```

You should expect to see an output like this:

```
ubuntu@test-hadoop-from-snapshot:~$ jps
17922 NameNode
18266 SecondaryNameNode
18047 DataNode
18383 Jps
```

Answer the following questions:

1. What are the roles of the files core-site.xml and hdfs-site.xml ?
2. Describe briefly the roles of the different services listed when executing ‘jps’.

Task 1.3
Now we will use Hadoop in pseudo-distributed mode to build and run our own version of the MapReduce example. In the ubuntu ‘home’ folder you will see a folder called ‘wordcount’. It contains the java source file for the WordCount example we ran in Task 1.1.

Compile the wordcount example and make a jar-file:

```
$ cd /home/ubuntu/wordcount
$ javac –cp `/usr/local/hadoop/bin/hadoop classpath` WordCount.java
$ jar -cvf wordcount.jar  *.class
```

Obs: the backquotes encapsulating the inner command, this enable nested commands and the inner will be executed first

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

Next, *load the datafiles in the .txt files into hdfs*. They are in the “wordcount/input”-folder. You can stage the input folder in HDFS by:

```
$  /usr/local/hadoop/bin/hdfs dfs -put input
```

Obs: you first need to create /user/ubuntu in HDFS (see tutorial linked above).

Next, run the code on Hadoop.

```
$ /usr/local/hadoop/bin/hadoop jar wordcount.jar WordCount <input_dir> <output_dir>
```

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

```
$ hdfs dfs -ls <output_dir>
```

Then check the content of the output file using the ‘-cat’ argument to ‘hdfs dfs’.

Answer the following questions:

1. Explain the roles of the different classes in the file WordCount.java.
2. What is HDFS, and how is it different from the local filesystem on your virtual machine?

Task 1.4
Your task now is to modify the above example code to, based on the same input files, count the occurrences of words starting with the same first letter (non case-sensitive). The output of the job should thus be a file containing lines like:

```
a   number_of_a
```

(Including all "A")

Make a plot of some sort that shows the counts of each letter and include that in your report. *Do not include the entire output file.*

Tips:

1. Again, use the tutorial link provided in task 1.2 and see how they "get" data from hdfs to the home folder on the instance.
2. For the plotting you will need to transfer data between hosts (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. There is a useful class called "Character" in Java that might come in handy.

Hand in the following code:

- A file FirstLetterCount.java with your modified code.

Part II: Analyzing twitter data using Hadoop streaming and Python

While Hadoop/MapReduce is based on Java, it is not necessary to use Java to write your mapper and reducers. The framework provides the "Streaming API", which lets you use any command line executable that reads from standard in and writes to standard out 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/

Note that:

1. The tutorial above uses Python 2 (which will no longer be maintained past 2020).

You should use Python 3.

2. 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
/usr/local/hadoop/share/hadoop/tools/lib/
(and NOT in /usr/local/hadoop/contrib/streaming/)

In this part of the assignment, we will analyze a dataset of ~5.000.000 Twitter tweets collected using Twitters datastream API. The total size of the dataset is still a modest ~9GB. The data is available as a tarball on the instance in the home folder.

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

Answer the following question:

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)?

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”.

Your task is now to use Python and the Hadoop streaming API to count the number of tweets mentioning each of these pronouns and plot a bar chart of the counts, **normalized** by the total number of **unique** tweets (e.g counts_"hen”/total_unique). In this analysis, only unique tweets should be taken into account, i.e. ‘retweets’ should be disregarded. The count should be case-insensitive, i.e. “Han” and “HAN” should count as mentions of the pronoun “han”.

Hint: *Use the Python library ‘json’ to parse the tweets.*
Tip: *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:

- A file mapper.py with your map code.

Part III: High-level interfaces/tools on top of the Hadoop framework

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

Hadoop has become a frequently used platform for big data analytics, and is used in production with clusters of thousands of nodes. Several high-level interfaces has been developed that attempt to make Hadoop/MapReduce easier to use by reducing the burden of having to write cumbersome MapReduce code. Two such tools are Apache Pig and Apache Hive. We will see a third alternative in Lab3, Apache Spark, which also has performance benefits.

Your task is now to redo the analysis in Part II using Apache Hive, by formulating the analysis as queries in HiveQL, a SQL-like query language. Note that you need to download and install Hive: https://cwiki.apache.org/confluence/display/Hive/GettingStjararted

Include all Hive queries that you use in your report.

Answer the following questions:

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.

2. 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.

3. 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?