Exercise 4
Using Hadoop

The goal of this assignment is to gain some experience on how to write and run programs that are executed in parallel on a cluster with Hadoop.

Introduction

For all of the tasks of this exercise, you will be working on virtual machines set up with CentOS8. Machines are set up on the VMWare server, and are accessible via SSH. From your own computer, you must use SSH to access the computers. All machines can be reached on the lab E425, or through the computer eple.hvl.no.

In the first part of this exercise you will use the Hadoop MapReduce Tutorial to get some basic knowledge about Hadoop. Then you will create your own Hadoop program(s) and redo the calculations of the previous lab exercise.

A Hadoop cluster is set up on the lab.

- 10.0.0.175 (**hadoopname.dat351**) – HDFS NameNode, YARN ResourceManager and YARN Web Application Proxy
- 10.0.0.177 (**hadoopa.dat351**) – YARN nodemamager and HDFS datanode
- 10.0.0.178 (**hadoopb.dat351**) – YARN nodemamager and HDFS datanode
- 10.0.0.179 (**hadoopc.dat351**) – YARN nodemamager and HDFS datanode
- 10.0.0.180 (**hadoopd.dat351**) – YARN nodemamager and HDFS datanode
- 10.0.0.182 (**hadoophistory.dat351**) – MapReduce History Server
- 10.0.0.171 (**hadoopnamesecond.dat351**) – HDFS Secondary NameNode
- 10.0.0.172 (**hadoopedge.dat351**) – Edge node for the Hadoop cluster

Status information from the Hadoop cluster is available with a web browser from the following URLs:

- HDFS file system:
  - http://hadoopname.dat351:9870 or
  - http://10.0.0.175:9870
- YARN resource manager and scheduler:
  - http://hadoopname.dat351:8088 or
  - http://10.0.0.175:8088
- MapReduce job statuses:
  - http://hadoophistory.dat351:19888 or
  - http://10.0.0.182:19888

In order to access the above URLs, you can connect your laptop to the wired network at the lab. If you run Linux, Cygwin or MobaXterm you can also access the URLs through Firefox on the computer **hadoopedge.dat351**. The Hadoop web pages can also be access through a SOCKS5 tunnel from your laptop to Eple.

You can run firefox on the **hadoopedge.dat351** with the command below

```
ssh -X -J dat351@eple.hvl.no studentX@hadoopedge "firefox --no-remote" &
```

To open a SOCKS5 tunnel from your laptop to Eple, run the command below on your laptop:

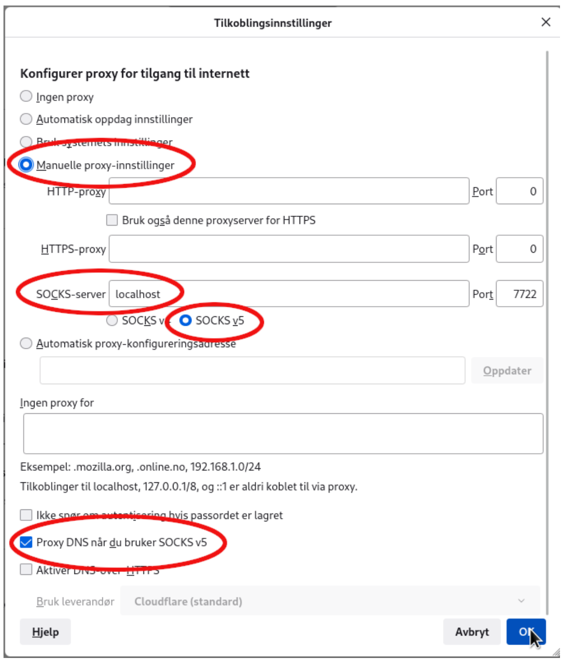
```
ssh -f -D 7722 -N -l dat351 eple.hvl.no
```

To stop the tunnel, you first have to find the PID of the tunnel process:

```
ps -C ssh -o pid,cmd | grep 7722 | awk '{print $1}'
```

You can then stop the tunnel with the **kill** command.

With the tunnel up and running, you can configure your browser to use the tunnel. If using Firefox, you configure the relevant network connections in the browser menu "Edit|Settings|General|Network Settings", then click *Settings* button. The figure below show the correct settings:



Tilkoblingsinnstillinger

Konfigurer proxy for tilgang til internett

- ○ Ingen proxy
- ○ Automatisk oppdag innstillinger
- ○ Bruk systemets innstillinger
- ● Manuelle proxy-innstillinger

HTTP-proxy: Port: 0

☐ Bruk også denne proxyserver for HTTPS

HTTPS-proxy: Port: 0

SOCKS-server: localhost Port: 7722

○ SOCKS v4 ● SOCKS v5

○ Automatisk proxy-konfigureringsadresse

Oppdater

Ingen proxy for

Eksempel: .mozilla.org, .online.no, 192.168.1.0/24

Tilkoblinger til localhost, 127.0.0.1/8, og ::1 er aldri koblet til via proxy.

☐ Ikke spør om autentisering hvis passordet er lagret

☑ Proxy DNS når du bruker SOCKS v5

☐ Aktiver DNS-over-HTTPS

Bruk leverandør: Cloudflare (standard)

Hjelp | Avbryt | OK

If using a WebKit browser (e.g. Opera, Chrome, Edge), you setup your network with the SOCKS5 tunnel in your OS settings. If Gnome, you find the settings under “Settings|Network|Network Proxy”.

All work of this task will be performed while you are logged in to the computer **hadoopedge.dat351**. On this computer, I have created 20 user-accounts for you, **student0** to **student19**. Use the same account as you used in the previous lab. The computer **hadoopedge.dat351** has the the same shared files system and password database as the computers of the first lab exercise. You log on to this computers with the same password as you have used before.

Login is only possible to the **hadoopedge.dat351** computer. The other Hadoop computers only have system users, local files and access to the HDFS file system.

Hadoop uses a distributed file system, *Hadoop Distributed File System* (HDFS). The input and output files of Hadoop have to reside on this file system.

I have created 20 directories in the HDFS file system, as “/user/student0” to “/user/student19”, that belong to the corresponding user account. When logged in to **hadoopedge.dat351**, you have access to HDFS file systems as the same user.

The Hadoop MapReduce tutorial – WordCount

The MapReduce Tutorial demonstrates how to run an example Hadoop program, *WordCount*. Here you will run the *WordCount* program on the Hadoop cluster in the lab.

Task one – Running WordCount

First, create a directory for the *WordCount* project on the HDFS file system.

```
echo $USER
hdfs dfs -mkdir wordcount
hdfs dfs -ls
```

Create the input files for the program.

```
mkdir -p ~/hadoop-files/wordcount/input/

cat > ~/hadoop-files/wordcount/input/file01.txt <<EOF
Hello World Bye World
```

```
EOF

cat > ~/hadoop-files/wordcount/input/file02.txt <<EOF
Hello Hadoop Goodbye Hadoop
EOF

ls -l ~/hadoop-files/wordcount/input
cat ~/hadoop-files/wordcount/input/*
```

Now, upload the input files to the HDFS file system. (This is not strictly necessary since you can ask Hadoop to stage the files when running the program.)

```
hdfs dfs -put ~/hadoop-files/wordcount/input wordcount

hdfs dfs -ls wordcount/input/
hdfs dfs -cat "wordcount/input/*"
```

Next, create a directory where you put the *WordCount* java file.

```
mkdir ~/hadoop-files/wordcount/java/
```

Put the source code of *WordCount* of the tutorial in the above directory.

Byte compile the program and create a JAR file.

```
cd ~/hadoop-files/wordcount/java/
hadoop com.sun.tools.javac.Main WordCount.java
jar cf wc.jar WordCount*.class
```

If you have run the program before, you must clean the output directory.

```
# Remove files from an earlier run of the program
hdfs dfs -rm -skipTrash -r wordcount/output
```

You are now ready to run the *WordCount* program.

```
hadoop jar ~/hadoop-files/wordcount/java/wc.jar \
  WordCount wordcount/input wordcount/output
```

The program shuld run without errors, but will produce a warning.

```
2022-09-21 10:27:10,504 WARN mapreduce.JobResourceUploader: Hadoop
  command-line option parsing not performed. Implement the Tool interface
  and execute your application with ToolRunner to remedy this.
```

When the program has finished, check the output.

```
hdfs dfs -ls wordcount/output
hdfs dfs -cat "wordcount/output/*"
hdfs dfs -cat "wordcount/input/*"
```

Task two – Using the Tool interface

The **Tool** interface let us easily act on the program arguments to the Java program. This will also remove the warning you got when running the *WordCount* example.

You will find, in directory "/share/dat351/java", a version of the *WordCount* progam that use the **Tool** interface.

Study the program code. Rewrite the program to also list all arguments that are given to the program. Then run the program on the input data of the *WordCount* progam.

Write your own Hadoop programs

In the previous lab you were working with three files with numbers (i.e. doubles). These files are available also on the hadoopedge computer in the directory "/share/dat351/input".

You will use Hadoop to parallelize the following tasks:

- Find the count of numbers summed over all three files.
- Find the smallest number.
- Find the largest number.
- Sum all the numbers.
- Count all the digits sums using only the integer part of the numbers.

Except for the last tasks, all tasks were solved also using MPI in the previous lab exercise.

For the last task, assume e.g. a number -165332.987. You skip the sign and decimal fraction and get 165332. Then you sum the digits, i.e. calculate

$$1 + 6 + 5 + 3 + 3 + 2 = 20$$

The number therefore has a digit sum over the integer part equal to 20. The last task is to count all the different digit sums over all the numbers, using only the integer part of the numbers. You should find the number of differrent digit sums.

With the given numbers you should find e.g. 276 numbers with digit sum 0, and 84399 numbers with digit sum 10.

On WordCount and WordCountTools

You can use the *WordCount* or *WordCountTools* programs as a starting point for your own Hadoop programs. I prefer *WordCountTools* as it let me access the program arguments and removes the Hadoop warning.

Let us look at some details of the programs. The discussion below is valid for both *WordCount* and *WordCountTools*.

```
public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>
```

The above code is the first lines of the **Mapper**. Both the **Mapper** and the **Reducer** take four generic type arguments.

The first generic type argument, i.e. **Object** is the type of the input key. With the default splitter, the key identifies the line in the split of the input value. Your program(s) will not need the actual value, so keep this argument as it is.

The second generic type argument, i.e. **Text** is the type of the input value. In your program, each line can be a text with a series of doubles. Keep it as it is. Type **Text** is a Hadoop text type.

The third generic type argument, i.e. **Text** is the type of the output key. Your program(s) can use output keys like *countvalue*, *minvalue*, *maxvalue*, and *sumvalue*. Type **Text** is appropriate.

The last generic type argument, i.e. **IntWritable** is the type of the output value. The output will be real numbers, except for *countvalue*. If you create one program that performs all the calculations, remember that output types can not be mixed, i.e. all the output values must be of the same type.

An appropriate real number type in Hadoop is **DoubleWritable** from package "org.apache.hadoop.io.DoubleWritable". Replace **IntWritable** with **DoubleWritable** where appropriate.

Both the **Mapper** and the **Reducer** store the output in a **Context** object using a method *write*. This method takes two arguments, an output key and an output value, i.e. they must match the third and fourth generic type elements.

Task three – Write your own Hadoop programs

Create Hadoop program(s) that solves the tasks outlined above. Run the program(s) and compare the values with what you got with MPI, and MPI with Slurm.

You can compare the ouput of the last task with the output that you find in the file "/share/dat351/output/digitcountstat.txt".

Discussion

Map and Reduce

The files with real numbers have one and only one number on each line. Discuss the following questions:

- How many times will *map()* be run?
- How many real numbers will each run of *map()* handle?
- If no **Combiner**, how man real numbers will be in the input to *reduce()*?
- If using a **Combiner**, how man real numbers will be in the input to *reduce()* of the **Combiner**?
- If using a **Combiner**, how man real numbers will be in the input to *reduce()* of the **Reducer**?

In directory "/share/dat351/java" you will find four programs that can helpful when discussing the above questions, *MapperCount*, *ReducerCount*, *ReducerCountWith-Combiner* and *CombinerCount*. Run the programs with input from the files with real numbers.

MapReduce and sorting

When finding the minimum or maximum values, you can rely on the internal Hadoop sorting. Hadoop will sort the input to the Reducer, using the input key.

If the real numbers were used as key values, the first key read by the Reducer will be the the smallest value. The only task of the Reducer would then be to store the first key.

Discuss the different approaches. What approach is best, i.e. using a key *minvalue* and let the **Reducer** find the smallest number, or rely on the internal Hadoop sorting?

MapReduce or MPI Reduce

Hopefully, you used the *MPI_Reduce* method of MPI to solve the MPI task, and a **Combiner** in your Hadoop applications.

The problems can be solved using both the *MPI_Reduce* method of MPI, or by using MapReduce. What framework is best suited for our problems? Does the use of a **Combiner** influence the performance of your Hadoop applications? You must discuss the answers.