Big Data: distributed storage, processing and computing for data-intensive applications

Lab 3: MapReduce on HDFS in a Hadoop cluster SOLUTION

Dr. Maxime Guériau
maxime.gueriau@insa-rouen.fr

INSA
INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN NORMANDIE

ITI4 – 2022

1. Creating a mini-cluster of virtual machines
2. Configuration of your Hadoop mini-cluster
3. Running a MapReduce job in HDFS
4. Exercise: French weather records

Objectives of this lab

1. Creating a mini-cluster of virtual machines
   - ☐ To import and exploit a ready-to-use Hadoop VM
   - ☐ To clone VMs
2. Configuration of your Hadoop mini-cluster
   - ☐ To set network hostnames and aliases
   - ☐ To configure and run HDFS on multiple nodes
   - ☐ To better understand the difference between NameNode(s) and DataNodes
   - ☐ To configure and run YARN on multiple nodes
   - ☐ To manipulate files on HDFS distributed system
3. Running a MapReduce job in HDFS
   - ☐ To run a MapReduce job on a (mini-)cluster
   - ☐ To monitor HDFS content and YARN nodes using the web interface
4. Bonus exercise: French weather records
   - ☐ To design a MapReduce job on your own and run it on your (mini-)cluster!

1. Creating a mini-cluster of virtual machines

1.1. Launch Oracle VM VirtualBox

1.2. Import the VM named `BGD_VM_TM.ova` located in `/opt/ova`[1] in VirtualBox.
Important: Assign VM resources (virtual drive; with a new name) in `/tmp`!

1.3. Select the Machine/Settings/Network option and configure the network to 'host-only adapter/réseau privé hôte'. Choose the adapter named `vboxnet0`[2]. This will allow the guest machine to be part of a private subnetwork managed by the host machine.

1.4. Rename this VM `BGD_VM_MASTER`.

1.5. Import `BGD_VM_TM.ova` again (use a different hard drive name) and name it `BGD_VM_NODE1`.

1.6. Import `BGD_VM_TM.ova` again (use a different hard drive name) and name it `BGD_VM_NODE2`.

1.7. Start the 3 VMs: `BGD_VM_MASTER`, `BGD_VM_NODE1` and `BGD_VM_NODE2`.

[1] Alternatively, on your own setup, you can download it from :
`https://nuage.insa-rouen.fr/index.php/s/4pHeLATGPsKRDLa/download`

[2] You may need to create it first using the 'Host Network Manager' option of VirtualBox

1. Creating a mini-cluster of virtual machines

On all three VMs:

1.8. Log in (user:'bgd', password: 'password').

1.9. Check all IP addresses and write them somewhere on the host machine.

On a terminal on the host machine:

1.10. Check that the host machine can ping the 3 VMs.

1.11. Check that you can establish a SSH connection to each VM and close all SSH sessions.

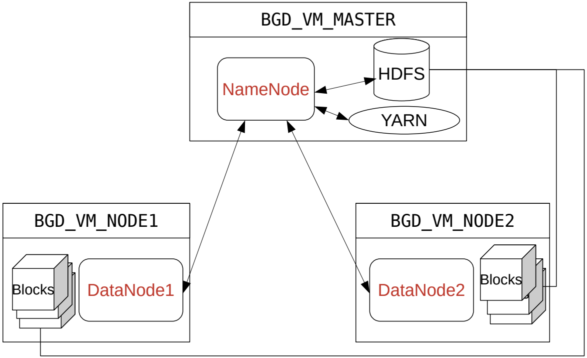
1.12. Shutdown `BGD_VM_NODE1` and `BGD_VM_NODE2`.

1.13. Open a new terminal and establish a SSH connection to `BGD_VM_MASTER`.

2. Configuration of your Hadoop mini-cluster

Your Hadoop cluster will be composed of 3 nodes:

- 1 NameNode:
  - named NameNode on `BGD_VM_MASTER`,
  - in charge of managing both HDFS and YARN processes (master node).
- 2 DataNodes:
  - DataNode1 on `BGD_VM_NODE1`,
  - DataNode2 on `BGD_VM_NODE2`,
  - both used as slaves to run MapReduce jobs and host blocks replications from HDFS.



2. Configuration of your Hadoop mini-cluster

2.1. Network configuration

In a terminal, opened on the host machine and connected via SSH to `BGD_VM_MASTER`:

2.1.1. First, change the VM hostname to "`NameNode`" in `/etc/hostname` file.

```
sudo nano /etc/hostname
```

and replace "bgd-vm" with "NameNode"

2.1.2. Then, edit `/etc/hosts` file: comment "localhost" lines and add one to specify IPs respectively corresponding to `MASTER`, `NODE1` and `NODE2`:

```
sudo nano /etc/hosts
```

Example:

```
192.168.1.31 NameNode
192.168.1.32 DataNode1
192.168.1.33 DataNode2
```

2.1.3. Close the SSH connection and shutdown the VM.

2.1.4. Repeat steps 2.1. to 2.3. with `NODE1` (hostname: "`DataNode1`") and `NODE2` (hostname: "`DataNode2`").

2.1.5. Restart all VMs (they need to be rebooted for the network configuration to be updated).

2. Configuration of your Hadoop mini-cluster

2.1. Network configuration

2.1.6. Establish a SSH connection from host to `MASTER`.

2.1.7. Check if `MASTER` can ping `DataNode1` and `DataNode2` and check if `MASTER` can perform a password-less SSH connection to `DataNode1` and `DataNode2`.

```
ping DataNode1
ping DataNode2
ssh bgd@DataNode1
ssh bgd@DataNode2
```

2. Configuration of your Hadoop mini-cluster

2.2. HDFS configuration

2.2.1 You now need to specify IP and port where HDFS will be available. To do so, open `$HADOOP_CONF_DIR/core-site.xml` file and add update the content of "configuration" tags with the following lines:

```
sudo nano $HADOOP_CONF_DIR/core-site.xml
```

```
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://NameNode:9000</value>
  </property>
</configuration>
```

2. Configuration of your Hadoop mini-cluster

2.2. HDFS configuration

2.2.2 As your cluster contains two DataNodes, you can set HDFS block replication value to 2; update `$HADOOP_CONF_DIR/hdfs-site.xml` with the following lines:

```
sudo nano $HADOOP_CONF_DIR/hdfs-site.xml
```

```
<configuration>
<property>
<name>dfs.replication</name>
<value>2</value>
</property>
<property>
<name>dfs.namenode.name.dir</name>
<value>/home/bgd/hdfs/namenode </value>
</property>
<property>
<name>dfs.datanode.data.dir</name>
<value>/home/bgd/hdfs/datanode </value>
</property>
</configuration>
```

2. Configuration of your Hadoop mini-cluster

2.3. Setting workers nodes

2.3.1. You now need to tell Hadoop what nodes can be used as DataNodes (workers). Thus, change the content of the file `$HADOOP_CONF_DIR/workers` to reflect your cluster setup:

```
sudo nano $HADOOP_CONF_DIR/workers
```

```
DataNode1
DataNode2
```

2.3.2. The full configuration you updated today also needs to be copied to the (slave) DataNodes. Copy the content of `$HADOOP_CONF_DIR` on each DataNode using the following commands:

```
scp $HADOOP_CONF_DIR/* DataNode1:$HADOOP_CONF_DIR/
scp $HADOOP_CONF_DIR/* DataNode2:$HADOOP_CONF_DIR/
```

2. Configuration of your Hadoop mini-cluster

2.4. Starting HDFS

2.4.1. Before starting HDFS, you need to format it on the NameNode:

```
hdfs namenode -format
```

2.4.2. And then you can start HDFS:

```
start-dfs.sh
```

2.4.3. To check if your configuration is correct and HDFS is working, you can show running Java Processes using `jps` command.

On NameNode, it should return:

```
xxxx Jps
xxxx NameNode
xxxx SecondaryNameNode
```

And on each DataNode (you can run `jps` through SSH or directly in each VM):

```
xxxx Jps
xxxx DataNode
```

2.4.4. Finally, on the **host** machine, you can access Hadoop web interface at the URL `http://192.168.xxx.xxx:9870/` (using `MASTER`'s IP address).

Visit the "DataNodes" tab and check you have 2 active nodes.

2. Configuration of your Hadoop mini-cluster

2.5. Configuring and starting YARN

2.5.1. **On both** (slave) **DataNodes**, you need to tell which node of your cluster is hosting YARN; update `$HADOOP_CONF_DIR/yarn-site.xml` with the following:

```
<configuration>
<property>
<name>yarn.resourcemanager.hostname</name>
<value>NameNode</value>
</property>
</configuration>
```

2.5.2. You can now start YARN **on `NameNode`**:

```
start-yarn.sh
```

2.5.3. Check if YARN works correctly using `jps` command (`NameNode` should now run the ResourceManager while each DataNode should run a DataManager process)

2.5.4. And check if your cluster is operational (it should have 2 active nodes) using YARN web interface at the URL `http://192.168.xxx.xxx:8088/cluster` on the host machine.

3. Running a MapReduce job in HDFS

3.1. Storing files on HDFS

In a terminal, opened on the host machine and connected via SSH to `NameNode`:

3.1.1. In HDFS, create a directory named "input" and located at the root of HDFS (tip: you can use the the cheatsheet provided (section 5) to find the right command.

```
hdfs dfs -mkdir /input
```

3.1.2. Check your last command was successful by browsing the content of HDFS through Hadoop web interface (`http://192.168.xxx.xxx:9870/`, "Utilities" tab/"Browse the file system"). The "input" directory "input should be there.

3.1.3. Move the content of "input" on the local file system to the `input` directory located on HDFS

```
hdfs dfs -put input /
```

Check on the web interface that `book.txt` has successfully been moved to the right location (`/input/book.txt` on HDFS).

3. Running a MapReduce job in HDFS

3.2. Back to previous Lab: the return of Wordcount!

3.2.1. Download, unzip, compile, and generate a `.jar` of previous lab WordCount code:

```
mkdir java
wget https://nuage.insa-rouen.fr/index.php/s/TbzdBWrZmQCxy97/download -O java/Wordcount.zip
cd java/
unzip Wordcount.zip
javac WordcountMain.java WordcountMap.java WordcountReduce.java
mkdir -p bgd/hadoop/wordcount
mv Wordcount*.class bgd/hadoop/wordcount/
jar -cvf wordcount.jar -C . bgd
cd ..
```

3.2.2. Now run this program on HDFS using the following template (you can use "/output" as the output folder on HDFS):

```
hadoop jar <pathToJarOnLocalFileSystem> bgd.hadoop.wordcount.WordcountMain
  <pathToInputFileOnHDFS> <pathToOutputDirOnHDFS>
```

```
hadoop jar java/wordcount.jar bgd.hadoop.wordcount.WordcountMain /input/book.txt /output
```

3. Running a MapReduce job in HDFS

3.2. Back to previous Lab: the return of Wordcount!

3.2.3. Check if it worked (by browsing files on HDFS through the web interface)

3.2.4. Copy output files from HDFS to `NameNode` local storage and display the results to check if everything went fine.

```
hdfs dfs -get /output output
cat output/*
```

4. Exercise: French weather records

3.1. Download the the `.csv` (790 Mo) file from
`https://nuage.insa-rouen.fr/index.php/s/Brp9yEQQZgri2no/download`
This file contains records of weather measurements in France for the last decade[3].

3.2. Write (and run) a MapReduce programs able to compute the average temperature per day (ex: "2016-01-22 4.79").

3.3. Using your favourite visualization tool (on your host machine), draw a graph that gathers yearly plots (x-axis: date and y-axis: average temperature).

Tips:

- The file is semi-column (';') separated
- Date is the 2nd column and looks like "2013-04-03T23:00:00+02:00"
- Temperature is the 8th column (given in Kelvin) (you can compute the temperature in Celsius using $T_c = T_k - 273.15$)
- The data include DOM-TOM regions, hence the quite high average value compared to what you experience in Normandy :)

[3] Source: `https://public.opendatasoft.com/explore/dataset/donnees-synop-essentielles-omm`

4. Exercise: French weather records

WeatherMain.java

```java
package bgd.hadoop.weather;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.util.GenericOptionsParser;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.IntWritable;

//The main "driver" class that runs a MapReduce job
public class WeatherMain {

  public static void main(String[] args) throws Exception
  {
    // Create a Hadoop configuration (includes generic parameters)
    Configuration conf=new Configuration();

    // Get non-generic parameters (i.e. written after hadoop ones)
    String[] ourArgs=new GenericOptionsParser(conf, args).getRemainingArgs();

    // Create an Hadoop Job (new Hadoop task) from the configuration (+ naming it)
    Job job=Job.getInstance(conf, "Weather");

    // Assign "driver", "map" and "reduce" classes to the Job
    job.setJarByClass(WeatherMain.class);
    job.setMapperClass(WeatherMap.class);
    job.setReducerClass(WeatherReduce.class);
```

4. Exercise: French weather records

WeatherMain.java

```java
    // Set key/values types for this Hadoop job
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);

    // Define the input and output files/directory
    // Here, remaining arguments given after Hadoop onesin the command line
    FileInputFormat.addInputPath(job, new Path(ourArgs[0]));
    FileOutputFormat.setOutputPath(job, new Path(ourArgs[1]));

    // Run the Hadoop Job
    if(job.waitForCompletion(true))
      System.exit(0); // Job completed
    System.exit(-1); // Job failed
  }
}
```

4. Exercise: French weather records

WeatherMap.java

```java
package bgd.hadoop.weather;

import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.IntWritable;
import java.util.StringTokenizer;
import org.apache.hadoop.mapreduce.Mapper;
import java.io.IOException;
import java.util.Arrays;

// The "mapper" class
// org.apache.hadoop.mapreduce.Mapper<KEYIN,VALUEIN,KEYOUT,VALUEOUT>
public class WeatherMap extends Mapper<Object, Text, Text, Text>
{

  private Text valueout = new Text();
  private Text keyout = new Text();

  public void map(Object key, Text value, Context context) throws IOException, InterruptedException {

    StringTokenizer itr = new StringTokenizer(value.toString());

    while (itr.hasMoreTokens()) {

      String line = itr.nextToken();

      //System.out.println(line);

      String[] values = line.split(";");

```

4. Exercise: French weather records

WeatherMap.java

```
      if(values.length>78){
        String temperature = values[7];
        if(temperature.compareTo("")!=0){
          double tk = Double.parseDouble(temperature);
          double tc = tk-273.15;
          valueout.set(""+tc);
          String date = values[1];
          String[] datebits = date.split("T");
          String day=datebits[0];
          keyout.set(day);
          context.write(keyout, valueout);
        }
      }
    }
  }
}
```

4. Exercise: French weather records

WeatherReduce.java

```java
package bgd.hadoop.weather;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.Reducer;
import java.util.Iterator;
import java.io.IOException;

// The "reducer" class
// org.apache.hadoop.mapreduce.Reducer<KEYIN,VALUEIN,KEYOUT,VALUEOUT>
public class WeatherReduce extends Reducer<Text, Text, Text, Text>
{

  private Text result = new Text();

  public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {

    String output = "";

    double length = 0;
    double sum = 0;

    for(Text val : values){
      sum+= Double.parseDouble(val.toString());
      length++;
    }

```

4. Exercise: French weather records

WeatherReduce.java

```
    double avgTemp = 0.0;

    if(length>0)
      avgTemp = sum/length;

    output = ""+avgTemp;

    result.set(output);
    context.write(key,result);

  }
}
```

Cheatsheet of HDFS commands

```
hdfs dfs -<command> <arg1> <arg2> ...
```

hdfs dfs -df <path>	show free disk space
hdfs dfs -ls <path>	list directory
hdfs dfs -mkdir <path>	create directory
hdfs dfs -put <files> ... <dir>	upload (local) files to hdfs
hdfs dfs -get <paths> ... <dir>	download files from hdfs to a local directory
hdfs dfs -cat <paths>	pipe files from hdfs to stout
hdfs dfs -mv <src> ... <dest>	move or rename files on hdfs
hdfs dfs -cp <src> ... <dest>	copy files on hdfs
hdfs dfs -rm <paths>	delete files on hdfs