School of Computer Science

COMP47470

Lab 5
Hadoop/HDFS (intro)

Teaching Assistant:	Leandro Batista de Almeida
Coordinator:	Anthony Ventresque
Date:	Thursday 22nd February, 2018
Total Number of Pages:	6

1 Installing Virtualbox and a Virtual Machine on your Laptop

- First, download VirtualBox and install it: `https://www.virtualbox.org/wiki/Downloads`
- Then download a Lubuntu image prepared for you (with Hadoop etc.) `http://csserver.ucd.ie/~aventresque/VMs/COMP47470-hadoop.ova` or `http://hibernia.ucd.ie/VMs/COMP47470-hadoop.ova`
- Import the Lubuntu image from the VirtualBox menu:

  `File > Import Appliance > choose your image.`

- You can start it now and it will boot.
- User-name: `user1` and password: `1234`

2 Setting-up the Environment

2.1 Java

You need to install a JVM and a compiler in order to run Hadoop. You can choose any distribution (OpenJVM, Oracle, IBM), but you need to be sure that a compiler is also available, and the Java version is 8. The virtual machine for this lab already has a JDK installed.

2.2 SSH

Hadoop uses ssh for inter-deamon[1] (and inter-node, but we have only 1 node here) communication, so each node must have ssh installed and set up with a proper authentication (with a system that does not require a password). You must install a ssh server, if not present. We use the key generation and distribution mechanisms to set up ssh (ssh is already installed and running). Run the following commands to create and install a public key that will allow the various Hadoop processes in your VM to know they can communicate with each other:

```
$ ssh-keygen -t rsa -P ""
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

2.3 Preparing the Hadoop Directory

The hadoop package is in the Download folder of the user's home directory. Extract the package (command `tar xzvf filename`) and move the whole directory structure to `/usr/local/hadoop` (command `sudo mv filename destination`).

[1] Daemon is a type of program running in the background in Linux/Unix systems.

2.4 Configuring .bashrc

Edit[2] the `.bashrc` file[3], in the user home directory, and add the following lines at the end of it:

```
#Hadoop Variables
export JAVA_HOME="/usr/lib/jvm/java-8-oracle"
export HADOOP_HOME=/usr/local/hadoop
export PATH=$PATH:$HADOOP_HOME/bin
export PATH=$PATH:$HADOOP_HOME/sbin
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib"
unset JAVA_TOOL_OPTIONS
```

2.5 Configuring Hadoop Files

All the files mentioned in this section are in the directory `$HADOOP_HOME/etc/hadoop` or in some of its sub-directories.

Add the following lines at the end of the files:

- `hadoop-env.sh` file:

```
export JAVA_HOME="/usr/lib/jvm/java-8-oracle"
export HADOOP_OPTS=-Djava.net.preferIPv4Stack=true
```

- `core-site.xml`

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

- copy the file `mapred-site.xml.template` and make it a file named `mapred-site.xml`. Add the following in the later file:

```
<property>
   <name>mapreduce.framework.name</name>
   <value>yarn</value>
</property>
```

- `hdfs-site.xml`

```
<property>
   <name>dfs.replication</name>
   <value>1</value>
</property>
<property>
   <name>dfs.namenode.name.dir</name>
   <value>file:/usr/local/hadoop/hadoop_data/hdfs/namenode</value>
```

[2] Note that you should see a file containing all the lines you have to add to your hadoop set up files in your VM (on the Desktop?). We've added it as copy/paste can be a little tricky to set up with VirtualBox.

[3] `.bashrc` is a file/script that is executed the first time bash runs. It usually contains aliases, i.e., variables, that the shell wants to set up.

```
    </property>
    <property>
       <name>dfs.datanode.data.dir</name>
       <value>file:/usr/local/hadoop/hadoop_data/hdfs/datanode</value>
    </property>
```

- yarn-site.xml

```
    <property>
       <name>yarn.nodemanager.aux-services</name>
       <value>mapreduce_shuffle</value>
    </property>
    <property>
       <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
       <value>org.apache.hadoop.mapred.ShuffleHandler</value>
    </property>
```

2.6 Prepare HDFS (namenode and datanode)

- Create two folders for the namenode and datanode (both are on your unique machine but they would often be on different machines):

```
$ mkdir -p /usr/local/hadoop/hadoop_data/hdfs/namenode
$ mkdir -p /usr/local/hadoop/hadoop_data/hdfs/datanode
$ sudo chown user1:user1 -R /usr/local/hadoop
```

- Format the HDFS filesystem:

```
$ hdfs namenode -format
```

2.7 Start the Cluster and Test the System

Start the daemons using the following commands:

```
$ start-dfs.sh
$ start-yarn.sh
$ mr-jobhistory-daemon.sh start historyserver
```

You need to be sure that all the deamons are running. In order to do this, use the `jps` tool, and verify the results. You should see 6 daemons running.

Use the web management tools to test that everything is running as expected: open a web browser and try the following addresses:

- HDFS admin: `http://localhost:50070`
- YARN admin: `http://localhost:8088`
- History Serve: `http://localhost:19888`

Use TestDFSIO to verify if the cluster is running. You will find the jar file in `HADOOP_HOME/share/hadoop/mapreduce`

```
$ hadoop jar hadoop-mapreduce-client-jobclient-2.7.2-tests.jar \
TestDFSIO -write -nrFiles 5 -fileSize 10
```

3 Load Datasets in HDFS

Make sure that your VM (in VirtualBox) has a network connection of some sort (e.g., through a bridge to your laptop). You might have to play with the "Settings" of your VM from the main interface of VirtualBox (category "Network"). Try "Bridged Adapter" for instance.

Download the following 6 books and upload them in your HDFS:

- `http://www.gutenberg.org/cache/epub/1524/pg1524.txt`
- `http://www.gutenberg.org/cache/epub/1112/pg1112.txt`
- `http://www.gutenberg.org/cache/epub/2267/pg2267.txt`
- `http://www.gutenberg.org/cache/epub/2253/pg2253.txt`
- `http://www.gutenberg.org/cache/epub/1513/pg1513.txt`
- `http://www.gutenberg.org/cache/epub/1120/pg1120.txt`

1. Create a repository in your HDFS datanode

   ```
   $> hdfs dfs -mkdir /dataset1
   ```

   Check (using `hdfs dfs -ls /`) that the repository exists

2. upload one of the books you've downloaded in the repository using

   ```
   $> hdfs dfs -copyFromLocal file targetRepositoryInHDFS
   ```

   or

   ```
   $> hdfs dfs -put file targetRepositoryInHDFS
   ```

3. Check they have been uploaded.
4. Remove the files (`-rm`) and delete the directories (`-rmdir`)
5. Finally, create a directory in your HDFS and upload the books

4 An example of MapReduce job: WordCount

This example[4] aims at counting the words of a corpus/text. You must create a project in any IDE of your choice, and create the classes for the mapper, the reducer and the job controller (note that we have a set up a project in Netbeans for you).

Follow these steps:

1. run the job in local mode (inside the IDE) to test the project
2. start the Hadoop daemons (dfs, yarn) - they're probably already running
3. upload the text files to the hdfs repository
4. test the job in the single-mode hadoop host

[4] `http://hadoop.apache.org/docs/r2.7.3/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html`

4.1 Mappers

```
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class WordCountMapper extends Mapper<Object,
Text, Text, IntWritable> {

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value,
    Context context) throws IOException, InterruptedException {
        StringTokenizer itr = new StringTokenizer(value.toString());
        while (itr.hasMoreTokens()) {
            word.set(itr.nextToken());
            context.write(word, one);
        }
    }
}
```

4.2 Reducer

```
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class WordCountReducer extends Reducer<Text, IntWritable,
Text, IntWritable> {

    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
    Context context) throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        result.set(sum);
        context.write(key, result);
    }
}
```

4.3 Job Control

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

```
public class WordCountJobControl {

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "word count");
        job.setJarByClass(WordCountJobControl.class);
        job.setMapperClass(WordCountMapper.class);
        job.setCombinerClass(WordCountReducer.class);
        job.setReducerClass(WordCountReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }

}
```