AVIGNON UNIVERSITÉ

M2 ILSEN & IA – 2023/24

UE Business intelligence & Systèmes décisionnels

ECUE Data Analytics & Big Data

Vincent Labatut

Annexe | Configuration basique de Hadoop sous Ubuntu

Ce document explique comment installer Hadoop sur les machines des salles de TP, de manière à pouvoir exécuter des applications MapReduce en modes *Standalone* (sans parallélisme, pour tester) et *Pseudo-distributed* (parallélisme sur une seule machine). La procédure décrite est adaptée du tutoriel disponible sur le site officiel d'Hadoop [1].

1 Installation

1. Téléchargez Hadoop 3.3.4 sur le site officiel :

`https://hadoop.apache.org/releases.html`

2. Dézippez l'archive dans votre dossier personnel `~`, et renommez le dossier obtenu en `~/hadoop` (en supprimant le numéro de la version). Vous devez avoir une arborescence de la forme `~/hadoop/bin`, `~/hadoop/etc`, etc.

3. Définissez la variable d'environnement de Java, et rajoutez-la au `PATH` :

```
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
export PATH=$PATH:$JAVA_HOME/bin
```

Bien sûr, le chemin est à ajuster à votre propre version de Java et à son chemin d'installation. Testez en requérant la version du JDK :

```
java -version
```

4. Définissez la variable d'environnement de Hadoop :

```
export HADOOP_HOME=~/hadoop
```

Testez en requérant la version de Hadoop :

```
$HADOOP_HOME/bin/hadoop version
```

5. Vérifiez que vous pouvez vous connecter à `localhost` en SSH sans mot de passe :

```
ssh localhost
```

Si c'est le cas, déconnectez-vous de SSH (CTRL-D) et passez à l'étape suivante. Sinon, créez une clé SSH sans mot de passe :

```
ssh-keygen -t rsa
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
chmod 0600 ~/.ssh/authorized_keys
```

2 Test du mode Standalone

6. Affichez l'aide du JAR d'exemple en standalone :

```
$HADOOP_HOME/bin/hadoop jar
                $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.3.4.jar
```

7. Copiez des données bidon, et exécutez ce même JAR sur elles :

1. `https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/SingleCluster.html`

```
mkdir input
cp $HADOOP_HOME/*.txt input
$HADOOP_HOME/bin/hadoop jar
                $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.3.4.jar
                                                                    wordcount input output
cat output/*
```

où `input` est un dossier d'entrée que l'on crée et remplit des fichiers textes quelconques, et `output` est le dossier de sortie créé lors de l'exécution du JAR.

> **Attention :** si le dossier `output` existe déjà (par exemple suite à un test précédent), l'exécution du JAR lèvera une exception.

8. Si le test a marché, vous pouvez supprimer les dossiers `~/input` et `~/output`, dont on n'a plus besoin.

> **Remarque :** il est possible que ce mode d'exécution ne fonctionne plus une fois que le mode pseudo-distribué aura été configuré.

3 Configuration du mode Pseudo-Distributed

9. Pour configurer HDFS, ouvrez le fichier `$HADOOP_HOME/etc/hadoop/core-site.xml` et complétez l'élément `configuration` avec :

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

10. Faites ensuite la même chose pour le fichier `$HADOOP_HOME/etc/hadoop/hdfs-site.xml` avec :

```
<configuration>
    <property>
        <name>dfs.replication</name>
        <value>1</value>
    </property>
</configuration>
```

11. Dans le fichier `$HADOOP_HOME/etc/hadoop/hadoop-env.sh`, complétez la variable d'environnement `JAVA_HOME` exactement comme à l'Étape 3

12. Pour configurer YARN, ouvrez le fichier `$HADOOP_HOME/etc/hadoop/mapred-site.xml` et complétez avec :

```
<configuration>
    <property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
    </property>
    <property>
        <name>mapreduce.application.classpath</name>
        <value>$HADOOP_MAPRED_HOME/share/hadoop/mapreduce/*
                            :$HADOOP_MAPRED_HOME/share/hadoop/mapreduce/lib/*</value>
    </property>
</configuration>
```

13. Faites ensuite la même chose pour le fichier `$HADOOP_HOME/etc/hadoop/yarn-site.xml` avec :

```
<configuration>
    <property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
    </property>
    <property>
        <name>yarn.nodemanager.env-whitelist</name>
        <value>JAVA_HOME,HADOOP_COMMON_HOME,HADOOP_HDFS_HOME,HADOOP_CONF_DIR,
                CLASSPATH_PREPEND_DISTCACHE,HADOOP_YARN_HOME,HADOOP_MAPRED_HOME</value>
    </property>
</configuration>
```

4 Test du mode Pseudo-Distributed

14. Formatez le système de fichier de Hadoop :

```
$HADOOP_HOME/bin/hdfs namenode -format
```

15. Démarrez HDFS (attention au `sbin` au lieu de `bin`) :

```
$HADOOP_HOME/sbin/start-dfs.sh
```

À ce stade, si vous obtenez une erreur de type `rcmd: socket: Permission denied`, il est possible que ce soit dû à la configuration de la fonction `rcmd`. Pour forcer l'utilisation de SSH, utilisez la commande :

```
export PDSH_RCMD_TYPE=ssh
```

et retentez de démarrer HDFS.

Une fois le service lancé, testez l'accès Web à HDFS en vous rendant à l'adresse suivante dans votre navigateur :

http://localhost:9870/

Remarque : l'arrêt de HDFS est réalisé au moyen de la commande `stop-dfs.sh`.

16. Créez les dossiers appropriés grâce à la commande HDFS `mkdir`, en remplaçant `<username>` par votre nom d'utilisateur :

```
$HADOOP_HOME/bin/hdfs dfs -mkdir /user
$HADOOP_HOME/bin/hdfs dfs -mkdir /user/<username>
$HADOOP_HOME/bin/hdfs dfs -mkdir input
```

Copiez des fichiers de test dans le dernier dossier créé, grâce à la commande HDFS `put` :

```
$HADOOP_HOME/bin/hdfs dfs -put $HADOOP_HOME/etc/hadoop/*.xml input
```

Vérifiez que les fichiers sont bien là, grâce à la commande HDFS `ls` :

```
$HADOOP_HOME/bin/hdfs dfs -ls input
```

17. Démarrez YARN (attention au `sbin`) :

```
$HADOOP_HOME/sbin/start-yarn.sh
```

Puis, testez l'accès Web à YARN en vous rendant à l'adresse suivante dans votre navigateur :

http://localhost:8088/

Remarque : l'arrêt de YARN est réalisé au moyen de la commande `stop-yarn.sh`.

18. Testez en exécutant le même JAR que précédemment (mais avec une autre fonctionnalité, histoire de changer un peu) :

```
$HADOOP_HOME/bin/hadoop jar
                $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.3.4.jar
                                                        grep input output 'dfs[a-z.]+'
```

Utilisez la commande suivante pour afficher le résultat :

```
$HADOOP_HOME/bin/hdfs dfs -cat output/*
```

5 Compilation et exécution en mode Pseudo-Distributed

19. Créez une classe Java `Test` contenant le code source suivant :

```
import java.io.IOException;
import java.util.concurrent.ThreadLocalRandom;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class Test
{  public static class TestMapper extends Mapper<LongWritable, Text, Text,
    IntWritable>
   {   public void map(LongWritable key, Text value, Context context) throws
       IOException, InterruptedException
       {  String string = value.toString();
          if(!string.isEmpty())
          {  Text outText = new Text(string.substring(0,1));
             int val = ThreadLocalRandom.current().nextInt(1,100);
             IntWritable outVal = new IntWritable(val);
             context.write(outText, outVal);
          }
     }
  }

  public static class TestReducer extends Reducer<Text, IntWritable, Text,
      IntWritable>
  {  public void reduce(Text key, Iterable<IntWritable> values, Context context)
      throws IOException, InterruptedException
     {  int sum = 0;
        for(IntWritable value: values)
           sum = sum + value.get();
        IntWritable outVal = new IntWritable(sum);
        context.write(key, outVal);
     }
  }

  public static void main(String[] args) throws Exception
  {  Configuration config = new Configuration();
     Job job = Job.getInstance(config, "Test MR program");
     job.setJarByClass(Test.class);
     job.setMapperClass(TestMapper.class);
     job.setReducerClass(TestReducer.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);
```

```
    }
}
```

20. Compilez la classe via Hadoop, puis faites-en un fichier JAR :

```
$HADOOP_HOME/bin/hadoop com.sun.tools.javac.Main Test.java
jar cf test.jar Test*.class
```

21. Au moyen de la commande HDFS `rm`, supprimez le dossier de sortie `output` généré lors du dernier test pour éviter une exception à l'exécution du nouveau JAR :

```
$HADOOP_HOME/bin/hdfs dfs -rm -r output
```

Par contre, le dossier `input` existant doit être conservé, car le programme en a besoin.

22. Exécutez le JAR via Hadoop en procédant comme précédemment :

```
$HADOOP_HOME/bin/hadoop jar test.jar Test input output
```

Et enfin affichez les résultats produits dans `output` :

```
$HADOOP_HOME/bin/hadoop fs -cat output/part-r-00000
```