UNIVERSITE DE M'SILA,

Faculté des Mathématiques et de l'Informatique

Département d'informatique

TP3

Purpose

This document describes how to set up and configure a single-node Hadoop installation so that you can quickly perform simple operations using Hadoop MapReduce and the Hadoop Distributed File System (HDFS).

Prerequisites

Supported Platforms

- GNU/Linux is supported as a development and production platform. Hadoop has been demonstrated on GNU/Linux clusters with 2000 nodes.
- Windows is also a supported platform but the followings steps are for Linux only. To set up Hadoop on Windows, see wiki page.

Required Software

Required software for Linux include:

1. Java™ must be installed. Recommended Java versions are described at HadoopJavaVersions.
2. ssh must be installed and sshd must be running to use the Hadoop scripts that manage remote Hadoop daemons if the optional start and stop scripts are to be used. Additionally, it is recommmended that pdsh also be installed for better ssh resource management.

Installing Software

If your cluster doesn't have the requisite software you will need to install it.

For example on Ubuntu Linux:

```
$ sudo apt-get install ssh
$ sudo apt-get install pdsh
```

Download

To get a Hadoop distribution, download a recent stable release from one of the Apache Download Mirrors.

Prepare to Start the Hadoop Cluster

Unpack the downloaded Hadoop distribution. In the distribution, edit the file `etc/hadoop/hadoop-env.sh` to define some parameters as follows:

```
  # set to the root of your Java installation
  export JAVA_HOME=/usr/java/latest
```

Try the following command:

```
  $ bin/hadoop
```

This will display the usage documentation for the hadoop script.

Now you are ready to start your Hadoop cluster in one of the three supported modes:

- Local (Standalone) Mode
- Pseudo-Distributed Mode
- Fully-Distributed Mode

Standalone Operation

By default, Hadoop is configured to run in a non-distributed mode, as a single Java process. This is useful for debugging.

The following example copies the unpacked conf directory to use as input and then finds and displays every match of the given regular expression. Output is written to the given output directory.

```
  $ mkdir input
  $ cp etc/hadoop/*.xml input
  $ bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-3.3.1.jar
grep input output 'dfs[a-z.]+'
  $ cat output/*
```

Pseudo-Distributed Operation

Hadoop can also be run on a single-node in a pseudo-distributed mode where each Hadoop daemon runs in a separate Java process.

Configuration

Use the following:

etc/hadoop/core-site.xml:

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

etc/hadoop/hdfs-site.xml:

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

Setup passphraseless ssh

Now check that you can ssh to the localhost without a passphrase:

```
  $ ssh localhost
```

If you cannot ssh to localhost without a passphrase, execute the following commands:

```
  $ ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
  $ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
  $ chmod 0600 ~/.ssh/authorized_keys
```

Execution

The following instructions are to run a MapReduce job locally. If you want to execute a job on YARN, see YARN on Single Node.

1. Format the filesystem:

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

2. Start NameNode daemon and DataNode daemon:

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

   The hadoop daemon log output is written to the `$HADOOP_LOG_DIR` directory (defaults to `$HADOOP_HOME/logs`).

3. Browse the web interface for the NameNode; by default it is available at:
   - NameNode - `http://localhost:9870/`
4. Make the HDFS directories required to execute MapReduce jobs:

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

5. Copy the input files into the distributed filesystem:

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

6. Run some of the examples provided:

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

7. Examine the output files: Copy the output files from the distributed filesystem to the local filesystem and examine them:

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

   or

   View the output files on the distributed filesystem:

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

8. When you're done, stop the daemons with:

   ```
   $ sbin/stop-dfs.sh
   ```

YARN on a Single Node

You can run a MapReduce job on YARN in a pseudo-distributed mode by setting a few parameters and running ResourceManager daemon and NodeManager daemon in addition.

The following instructions assume that 1. ~ 4. steps of the above instructions are already executed.

1. Configure parameters as follows:

   `etc/hadoop/mapred-site.xml`:

   ```
   <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>
   ```

   `etc/hadoop/yarn-site.xml`:

   ```
   <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,CLASS
   PATH_PREPEND_DISTCACHE,HADOOP_YARN_HOME,HADOOP_HOME,PATH,LANG,TZ,HADOOP_MA
   PRED_HOME</value>
       </property>
   </configuration>
   ```

2. Start ResourceManager daemon and NodeManager daemon:

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

3. Browse the web interface for the ResourceManager; by default it is available at:

        - ResourceManager - `http://localhost:8088/`
4. Run a MapReduce job.
5. When you’re done, stop the daemons with:

```
$ sbin/stop-yarn.sh
```

Example: WordCount v1.0

Before we jump into the details, lets walk through an example MapReduce application to get a flavour for how they work.

`WordCount` is a simple application that counts the number of occurrences of each word in a given input set.

This works with a local-standalone, pseudo-distributed or fully-distributed Hadoop installation (Single Node Setup).

Source Code

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

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.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 WordCount {

  public static class TokenizerMapper
       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);
      }
    }
  }

  public static class IntSumReducer
       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);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.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);
  }
}
```

Usage

Assuming environment variables are set as follows:

```
export JAVA_HOME=/usr/java/latest
export PATH=${JAVA_HOME}/bin:${PATH}
export HADOOP_CLASSPATH=${JAVA_HOME}/lib/tools.jar
```

Compile `WordCount.java` and create a jar:

```
$ bin/hadoop com.sun.tools.javac.Main WordCount.java
$ jar cf wc.jar WordCount*.class
```

Assuming that:

- `/user/joe/wordcount/input` - input directory in HDFS
- `/user/joe/wordcount/output` - output directory in HDFS

Sample text-files as input:

```
$ bin/hadoop fs -ls /user/joe/wordcount/input/
/user/joe/wordcount/input/file01
/user/joe/wordcount/input/file02

$ bin/hadoop fs -cat /user/joe/wordcount/input/file01
Hello World Bye World

$ bin/hadoop fs -cat /user/joe/wordcount/input/file02
Hello Hadoop Goodbye Hadoop
```

Run the application:

```
$ bin/hadoop jar wc.jar WordCount /user/joe/wordcount/input
/user/joe/wordcount/output
```

Output:

```
$ bin/hadoop fs -cat /user/joe/wordcount/output/part-r-00000
Bye 1
```

```
Goodbye 1
Hadoop 2
Hello 2
World 2
```

Applications can specify a comma separated list of paths which would be present in the current working directory of the task using the option `-files`. The `-libjars` option allows applications to add jars to the classpaths of the maps and reduces. The option `-archives` allows them to pass comma separated list of archives as arguments. These archives are unarchived and a link with name of the archive is created in the current working directory of tasks. More details about the command line options are available at Commands Guide.

Running `wordcount` example with `-libjars`, `-files` and `-archives`:

```
bin/hadoop jar hadoop-mapreduce-examples-<ver>.jar wordcount -files
cachefile.txt -libjars mylib.jar -archives myarchive.zip input output
```

Here, myarchive.zip will be placed and unzipped into a directory by the name “myarchive.zip”.

Users can specify a different symbolic name for files and archives passed through `-files` and `-archives` option, using #.

For example,

```
bin/hadoop jar hadoop-mapreduce-examples-<ver>.jar wordcount -files
dir1/dict.txt#dict1,dir2/dict.txt#dict2 -archives mytar.tgz#tgzdir input output
```

Here, the files dir1/dict.txt and dir2/dict.txt can be accessed by tasks using the symbolic names dict1 and dict2 respectively. The archive mytar.tgz will be placed and unarchived into a directory by the name “tgzdir”.

Exercice1

1- Modify the `wordcount programto be count only the number of occurrence of the name “Hello”.`