Hadoop Lab Notes

Nicola Tonellotto

November 15, 2010

Contents

1 Hadoop Setup

1.1 Prerequisites

1. Gnu/Linux computer
2. Java 1.6 SDK installed
3. SSH must be installed and SSHD must be running

1.2 Installation

1. Create the `hadoop` user account and login as `hadoop` user.
2. Download `hadoop-??0.20.2.tar.gz` in your home dir.
3. Unpack the downloaded HADOOP distribution in your home dir.
4. Check that you can ssh to localhost without a passphrase:

```
hadoop@localhost$ ssh localhost
```

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

```
hadoop@localhost$ ssh-keygen -t dsa -P '' -f ~/.ssh/id_dsa
hadoop@localhost$ cat ~/.ssh/id_dsa.pub >> ~/.ssh/authorized_keys
```

5. Move to the HADOOP distribution dir:

```
hadoop@localhost$ cd $HOME/hadoop-0.20.2
```

6. Create the `HADOOP_HOME` environment variable:

```
hadoop@localhost$ export HADOOP_HOME=`pwd`
```

7. Edit the file `$HADOOP_HOME/conf/hadoop-env.sh` to define at least `JAVA_HOME` to be the root of your Java installation.
8. Try the following command:

```
hadoop@localhost$ bin/hadoop
```

   This will display the usage documentation for the `hadoop` script.

1.3 Verification

1. By default, HADOOP is configured to run in a non-distributed mode (standalone mode), as a single Java process. This is useful for debugging.

2. 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.

↑Code

```
hadoop@localhost$ mkdir input
hadoop@localhost$ cp conf/*.xml input
hadoop@localhost$ bin/hadoop jar hadoop-0.20.2-examples.jar \
                    grep input output 'dfs[a-z.]+'
hadoop@localhost$ cat output/*
```

↓Code

3. Clean up:

↑Code

```
hadoop@localhost$ rm -rf input output
```

↓Code

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

5. Edit the `conf/core-site.xml` file:

↑Code

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

↓Code

6. Edit the `conf/hdfs-site.xml` file:

↑Code

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

↓Code

7. Edit the `conf/mapred-site.xml` file:

↑Code

```
<configuration>
  <property>
    <name>mapred.job.tracker</name>
    <value>localhost:9001</value>
  </property>
</configuration>
```

↓Code

8. Format a new distributed filesystem:

↑Code

```
hadoop@localhost$ bin/hadoop namenode -format
```

↓Code

9. Start the HADOOP daemons:

↑Code

```
hadoop@localhost$ bin/start-all.sh
```

↓Code

10. Browse the web interface for the NameNode and the JobTracker; by default they are available at:
    - NameNode - `http://localhost:50070`
    - JobTracker - `http://localhost:50030`
11. Copy the input files into the distributed filesystem:

↑Code

```
hadoop@localhost$ bin/hadoop fs -put conf input
```

↓Code

12. Run some of the examples provided:

↑Code

```
hadoop@localhost$ bin/hadoop jar hadoop-*-examples.jar \
                  grep input output 'dfs[a-z.]+'
```

↓Code

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

↑Code

```
hadoop@localhost$ bin/hadoop fs -get output output
hadoop@localhost$ cat output/*
```

↓Code

14. Clean up:

```
hadoop@localhost$ rm -r output
hadoop@localhost$ bin/hadoop fs -rmr input output
```

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

```
hadoop@localhost$ bin/stop-all.sh
```

2 Word Count Exercise

We will see a basic HADOOP implementation of the word count application. Create the following `WordCount.java` source file in the `HADOOP_HOME` dir.

```
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 NewMapper 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 NewReducer 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 = new Job(conf, "wordcount");
    job.setJarByClass(WordCountNew.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    job.setMapperClass(NewMapper.class);
    job.setCombinerClass(NewReducer.class);
    job.setReducerClass(NewReducer.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

↓Code

- Compile `WordCount.java` and create a jar file:

↑Code

```
hadoop@localhost$ cd ${HADOOP_HOME}
hadoop@localhost$ mkdir classes
hadoop@localhost?$ javac -cp hadoop-0.20.2-core.jar -d classes WordCount.java
hadoop@localhost?$ jar -cvf wordcount.jar -C classes/ .
```

↓Code

- Create the following sample files in your `HOME` dir:

↑Code

```
hadoop@localhost$ echo Hello World > file01
hadoop@localhost$ echo Hello Java > file02
hadoop@localhost$ echo Java and MapReduce > file03
```

↓Code

- Create the HDFS input dir:

↑Code

```
hadoop@localhost$ bin/hadoop fs -mkdir /user/hadoop/wordcount/input
```

↓Code

- Copy the sample files in HDFS:

↑Code

```
hadoop@localhost$ bin/hadoop fs -put file0? /user/hadoop/wordcount/input/
```

↓Code

- Check the sample files have been copied:

↑Code

```
hadoop@localhost$ bin/hadoop fs -ls /user/hadoop/wordcount/input/
hadoop@localhost$ bin/hadoop fs -cat /user/hadoop/wordcount/input/file01
hadoop@localhost$ bin/hadoop fs -cat /user/hadoop/wordcount/input/file02
hadoop@localhost$ bin/hadoop fs -cat /user/hadoop/wordcount/input/file03
```

↓Code

- Run the application:

↑Code

```
hadoop@localhost$ bin/hadoop jar wordcount.jar WordCount \
                  /user/hadoop/wordcount/input /user/hadoop/wordcount/output
```

↓Code

- Check the output:

↑Code

```
hadoop@localhost$ bin/hadoop fs -cat /user/hadoop/wordcount/output/part-00000
```

↓Code

- Clean up

↑Code

```
hadoop@localhost$ rm -r WordCount.java wordcount.jar classes/ file0?
hadoop@localhost$ bin/hadoop fs -rmr /user/hadoop/wordcount
```

↓Code