Hadoop Tutorial
University of Amsterdam BSc Informatica
Concurrency & Parallel Programming

Contents

1 Overview

Hadoop MapReduce is a framework for writing applications which process large volums of data in-parallel.

A MapReduce job usually splits an input data-set into chunks which are processed by map tasks in a parallel manner. The framework sorts the outputs of the maps, which are then input to the reduce tasks. Typically both the input and the output of the job are stored in a file-system.

2 Hadoop Module

Before you do the assignment you need to familiarize yourselves with writing and executing MapReduce jobs on DAS4. This tutorial is based on the one found in `http://hadoop.apache.org/docs/r2.5.0/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html`. To run the it you need to log in to fs0.das4.cs.vu.nl. That is because Hadoop is only installed on the VU cluster. To load the Hadoop module type:

```
$module load hadoop/2.5.0
```

To make sure you have the correct version (2.5.0) you can type:

```
$hadoop version
```

The output should look like this:

```
Hadoop 2.5.0
Subversion http://svn.apache.org/repos/asf/hadoop/common -r 1616291
Compiled by jenkins on 2014-08-06T17:31Z
Compiled with protoc 2.5.0
From source with checksum 423dcd5a752eddd8e45ead6fd5ff9a24
This command was run using /cm/shared/package/hadoop/hadoop-2.5.0/
     share/hadoop/common/hadoop-common-2.5.0.jar
```

3 WordCount Tutorial

For this tutorial you'll execute the famous WordCount example. This is a simple application that counts the number of occurrences of each word in a given input file. You can run this tutorial in two modes:

1. local-standalone: This mode is not using the Hadoop cluster at all. It runs the code on a single machine e.g. your laptop/workstation and uses the local file system. You can use this mode for testing and debugging your code

2. fully-distributed: This mode uses the Hadoop cluster and the Hadoop distributed file system (hdfs). This is entirely different from the local file system on the head node of DAS4. The map and reduce jobs run on different nodes in the cluster therefore, you'll not be able to see debug messages printed in stdout or stderr. Use this mode when running your final experiments

Download the project code from blackboard. The project contains the following folders and files:

- lib/: The libraries of the project. They include the Hadoop API (`http://hadoop.apache.org/`), the Stanford Natural Language Processing API (`http://nlp.stanford.edu/`) and the JLangDetect API (`http://www.jroller.com/melix/entry/jlangdetect_0_3_released_withdependencies`). The last two APIs are necessary for the assignment and they are used for sentiment analysis[1] and language identification[2] in a text

- src/ : The source code

- compile.sh :A script to compile the code

3.1 Configure the Job

Look at the class in `src/nl/uva/AssignmentMapreduce.java`. This class contains the main method and the job configuration. As you can see in method `configureJob()` we set the classes for the map and reduce task. After setting the map class we set the type for the map output key/value pairs:

```
conf.setMapOutputKeyClass(Text.class);
conf.setMapOutputValueClass(DoubleWritable.class);
```

We also need to set the input and output type of the input files for the mappers

```
conf.setInputFormat(TextInputFormat.class);
```

[1] Sentiment analysis aims to determine the attitude of a person

[2] Weather the written language is Dutch, English, etc.

This line is important for specifying the way the input is going to be split. The `TextInputFormat` is meant for plain text files. Files are broken into lines. Keys are the position in the file, and values are the line of text.

The data types emitted by the reducer are set by `setOutputKeyClass()` and `setOutputValueClass()`:

```
conf.setOutputKeyClass(Text.class);
conf.setOutputValueClass(Text.class);
```

Next, we need to set the input and output paths for the entire job. The variable `dataset` is the path of the text file we'll use to count the words, and the `outputFolder` is where we'll put the results.

```
Path localPath = new Path(dataset);
FileInputFormat.setInputPaths(conf, localPath);
FileOutputFormat.setOutputPath(conf, new Path(outputFolder));
```

Attention: The input and output paths can be local in the case we are running the job in local-standalone mode or hdfs paths in case you are running in fully-distributed mode.

Finally, we set the maximum number of mappers and reducers for the job:

```
if (maxMap > -1) {
  conf.setNumReduceTasks(maxMap);
  conf.setNumMapTasks(maxMap);
}
```

3.2 Map

After configuring the job we define the Map job. Look at the class in `src/nl/uva/Map.java`: This class contains the map job. It effectively reads the input file line by line and transmits the key/value pairs. The key in this case is the word and the value is one.

Note that in the class declaration we implement the `Mapper` interface. When implementing the `Mapper` interface, it is important to correctly define the type of the input and output key/value pairs

```
implements Mapper<LongWritable, Text, Text, IntWritable>
```

That same must be done when declaring the `map` method:

```
LongWritable key, Text value, OutputCollector<Text, IntWritable> oc
```

Next in the method we split the incoming lines into words:

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

and simply iterate through the words and emit them to the reducer:

```
while (itr.hasMoreTokens()) {
  word.set(itr.nextToken());
  oc.collect(word, one);
}
```

The rest of the lines are for reporting the mapper's progress:

```
rprtr.incrCounter(Counters.INPUT_LINES, 1);
count++;
if ((++count % 100) == 0) {
  rprtr.setStatus("Finished processing " + count + " records");
}
```

The emitted key/value pairs will be grouped into keys (in this case words) and transmitted to the reducer.

3.3 Reduce

Look at the class in `src/nl/uva/Reduce.java`. As with the `Map` class, in the `Reduce` class declaration we implement the `Reducer` interface. When implementing the `Reducer` interface, it is important to correctly define the type of the input and output key/value pairs:

```
implements Reducer<Text, IntWritable, Text, IntWritable>
```

That same must be done when declaring the `reduce` method:

```
Text key, Iterator<IntWritable> itrtr, OutputCollector<Text,
    IntWritable> output
```

The `Reduce`'s class job is to add the keys (words) emitted from the `Map` class:

```
int sum = 0;
int count = 0;
while (itrtr.hasNext()) {
  sum += itrtr.next().get();
}
```

After the `Reduce` class has the summation it has to save it to the output:

```
output.collect(key, new IntWritable(sum));
```

The rest of the code is for reporting the progress:

```
count++;
if ((++count % 100) == 0) {
    rprtr.setStatus("Finished processing " + count + " records ");
}
rprtr.incrCounter(Counters.OUTPUT_LINES, 1);
```

The output of the Reduce class is saved in the output path you specified in the outputFolder variable in the AssignmentMapreduce class in the file named part-00000.

3.4 Compile & Execute

To execute the code you should run the compile script:

```
$./compile.sh
```

To make the script executable you should run:

```
$chmod +x compile.sh
```

After compiling, a jar file named nl.uva.AssignmentMapreduce.jar should be created. This jar file contains all the libraries and the MappReduce job.

Before running the job you should download the tweets2009-06-brg.txt file from the blackboard. After you have the file on you local disk you can run the job in a local-standalone mode by typing:

```
$java -jar nl.uva.AssignmentMapreduce.jar tweets2009-06-brg.txt out
    1
```

Before running the job in fully-distributed mode you should copy the tweets2009-06-brg.txt on the hdfs. To do that you can type:

```
$hdfs dfs -copyFromLocal tweets2009-06-brg.txt
```

This will copy the tweets2009-06-brg.txt file on the hdfs at /user/$USER. For a full list of commands for the hdfs you can look at : http://hadoop.apache.org/docs/r2.5.1/hadoop-project-dist/hadoop-common/FileSystemShell.html

To run the job in a fully-distributed mode (on DAS4) type:

```
$hadoop jar nl.uva.AssignmentMapreduce.jar tweets2009-06-brg.txt
    out 1
```

Attention: If you attempt to run the job for the second time you will get an error like:

```
org.apache.hadoop.mapred.FileAlreadyExistsException: Output
    directory hdfs://master.ib.cluster:8020/user/$USER/out already
    exists
```

If you want to run the job again, either delete the output folder or specify another one.