Developing MapReduce applications

Shadi Ibrahim
March 23rd, 2017

MapReduce has emerged as a leading programming model for data-intensive computing. It was originally proposed by Google to simplify development of web search applications on a large number of machines.

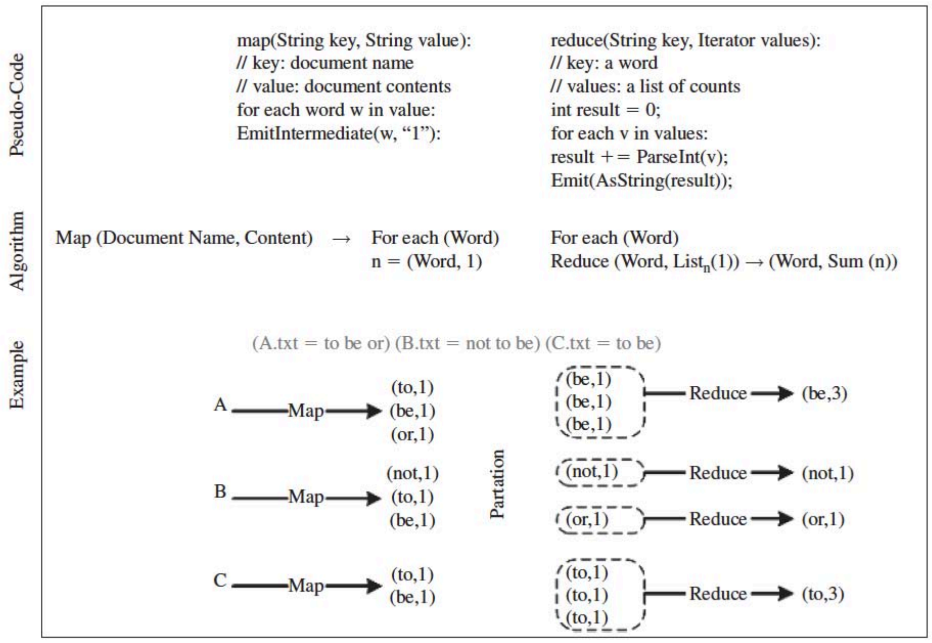
Hadoop is a java open source implementation of MapReduce sponsored by Yahoo! The Hadoop project is a collection of various subprojects for reliable, scalable distributed computing. The two fundamental subprojects are the Hadoop MapReduce framework and the HDFS. HDFS is a distributed file system that provides high throughput access to application data. It is inspired by the GFS. HDFS has master/slave architecture. The master server, called NameNode, splits files into blocks and distributes them across the cluster with replications for fault tolerance. It holds all metadata information about stored files. The HDFS slaves, the actual store of the data blocks called DataNodes, serve read/write requests from clients and propagate replication tasks as directed by the NameNode.

The Hadoop MapReduce is a software framework for distributed processing of large data sets on compute clusters. It runs on the top of the HDFS. Thus data processing is collocated with data storage. It also has master/slave architecture. The master, called Job Tracker (JT), is responsible of : (a) Querying the NameNode for the block locations, (b) considering the information retrieved by the NameNode, JT schedule the tasks on the slaves, called Task Trackers (TT), and (c) monitoring the success and failures of the tasks.

After studying in the first TP, how to operate the Hadoop platform using examples, we will now implement MapReduce applications to solve several types of problems: including numerical calculation and analysis examples.

Exercise 1: My Wordcount application

As a simple illustration of the Map and Reduce functions, Figure 1 shows the pseudo-code and the algorithm and illustrates the process steps using the widely used "Wordcount" example. The Wordcount application counts the number of occurrences of each word in a large collection of documents. The steps of this process are briefly described as follows: The input is read (typically from a distributed file system) and broken up into key/value pairs (e.g., the Map function emits a word and its associated count of occurrence, which is just "1"). The pairs are partitioned into groups for processing, and they are sorted according to their key as they arrive for reduction. Finally, the key/value pairs are reduced, once for each unique key in the sorted list, to produce a combined result (e.g., the Reduce function sums all the counts emitted for a particular word).



```
map(String key, String value):
// key: document name
// value: document contents
for each word w in value:
EmitIntermediate(w, "1"):
```

```
reduce(String key, Iterator values):
// key: a word
// values: a list of counts
int result = 0;
for each v in values:
result += ParseInt(v);
Emit(AsString(result));
```

Figure 1

Question 1.1

To use Eclipse to edit your MapReduce programs, add different Hadoop's packages to the Java Build Path of your project. To do so, after creating a new project in your workspace (MyWordCount), add all the jar files in the root folder of **Hadoop** and those in the **lib** folder to the Java Build Path:

- Select Build Path from the context menu (right click on the file) project then Configure Build Path.
- In the Libraries tab, click Add External JARs;
- Add all jar files in the root folder of the Hadoop and in the lib directory;

Now you are ready to create your first MapReduce program.

Question 1.2
Copy the file **WordCount.java** (you can find it in the resources) to your Eclipse project. By integrating this file in Eclipse, make sure the compilation goes well (no red cross).

Question 1.3
Create you Jar file **MyWordCount.jar**.

Question 1.4
Copy **MyWordCount.jar** to Hadoop folder and then run the wordcount with the data set 5GB.

```
hadoop jar MyWordCount.jar WordCount input output
```

Exercise 2: Modify your WordCount application

Question 2.1
Change the number of Reduce task by adding:

```
job.setNumReduceTasks(X);
```

Create your Jar file and Run the job with 2 and 4 reducers.

Question 2.2
Change the maximum size of map split size (the default size is 64 MB 67108864 bytes)

```
job.getConfiguration().setLong("mapred.max.split.size", X);
```

Create your Jar file and Run the job with two values $n \times 512$.

Question 2.3
Create a new project (MyWourdCount1), copy WordCount.java (named WordCount1.java in the resources) , now add and delete this line:

```
job.setCombinerClass(Reduce.class);;
```

Run the program with and without the combiner function.

Question 2.4
Create a new project that gives you the Line Count (you can find an example in the resources).

Exercise 3(Optional): Developing your Π application

This application estimates the value of pi based on sampling. The estimator first generates random points in a 1×1 area. The map phase checks for each pair if it falls inside a 1-diameter circle; the reduce phase computes the ratio between the number of points inside the circle and the ones outside the circle. This ratio gives an estimate for the value of pi.

Question 3.1

To do this exercise , you will use the skeleton MyPiEstimator.java and the the HaltonSequence.java class (in the resources folder): A Java MapReduce program that will take the number of parameter maps, Reduces the number of and the number of samples per map.

- Define the input files for each map.
- Retrieve and parse the output files.
- Calculate and show the value of Pi.

Question 3.2

Write a Java code for the Map method.

Question 3.3

Write a Java code for the Reduce method.