Peter Dely
Karl-Johan Grinnemo
Dept. of Computer Science

Assignment #2

Revision A
2017-08-07

MapReduce Framework

Purpose

This assignment aims at introducing students to the MapReduce framework and the Apache Hadoop implementation of this framework.

Learning Outcomes

- Be able to explain the basic principles behind the MapReduce Framework.
- Be able to translate a simple sequential task into a corresponding parallel task that fits the MapReduce framework, and solve the problem using the Apache Hadoop MapReduce framework.

Reading Instructions

Read intensively the paper: ”MapReduce: Simplified Data Processing on Large Clusters”. The paper is found on “its learning”, in folder: “Assignments/Assignment #2”.
Also, read extensively the Yahoo Hadoop MapReduce tutorial: “Hadoop MapReduce Tutorial (YDN)”. The tutorial is found on “its learning”, in folder: “Assignments/Resources/The MapReduce Framework”. Concentrate on reading the section “Hadoop Streaming”. Resources on C++ and the C++ Standard Library (including STL) are found on “its learning”, in folder “Assignments/Resources/C++”.

Development Environment

The program assignment is done in the Linux (Ubuntu 12.04 LTS) virtual machine, `C:\vmware\dvgc15-ht17`, specifically setup for the course. You start the Linux virtual machine by double-clicking on the VMware settings file, `dvgc15-ht17.vmx`, in folder: `C:\vmware\dvgc15-ht17`.

We use a dedicated Hadoop user account for running this lab, “Hadoop User”, with password “dvgc15”. The programming in this lab is intended to be done in C++, however, students are permitted to choose another programming language such as Perl, Python, etc.

The NetBeans, Eclipse, and Geany IDEs are installed in the Linux virtual machine. Furthermore, editors such as Emacs, vim, gedit, etc. are available.

Assignment Description

An inverted index is a list of words and files/positions where the words occur. Inverted indexes have many applications, e.g., search engines. To better understand what an inverted index is, look at an example. Consider the following two files:

file1.txt

```
This is just a test is
```

file2.txt

```
This is not a file, is it?
```

The inverted index of the two files looks as follows:

```
a        file2.txt:12 file1.txt:13
file,    file2.txt:14
is       file2.txt:20,5 file1.txt:20,5
it?      file2.txt:23
just     file1.txt:8
not      file2.txt:8
test     file1.txt:15
This     file2.txt:0 file1.txt:0
```

For each unique word there is a line specifying in which files the word is found, and at which offsets in the files the word is found.

Your task is to write a MapReduce application that creates an inverted index. The application is to be written for the Apache Hadoop Streaming extension.

Apache Hadoop is a framework that can distribute MapReduce jobs on large clusters of computers. Hadoop Streaming is an extension of Hadoop that allows you to use normal command line applications as map and reducer processes.

You should develop you application in two steps:

1. Write “mapper” and “reducer” applications that can be run locally without Hadoop.
2. Run the “mapper” and “reducer” applications from Step 1 with Hadoop, using Hadoop Streaming on a single-node Hadoop cluster (i.e., on a single computer).

Step 1: Write “mapper” and “reducer” Applications

You need to create two small programs: a “mapper” and a “reducer”. Both programs read its input from `stdin` (standard input) and write their output to `stdout` (standard output). The “mapper” reads the input files, line by line, and emits a key-value pair for each word in the file. The key is the word itself, and the value is the filename and the offset.

For `file1.txt`, the output of the “mapper” program should look as follows:

```
This     file1.txt:0
is       file1.txt:5
just     file1.txt:8
a        file1.txt:13
test     file1.txt:15
is       file1.txt:20
```

And, the output for `file2.txt` should look as follows:

```
This     file2.txt:0
is       file2.txt:5
not      file2.txt:8
a        file2.txt:12
file,    file2.txt:14
is       file2.txt:20
it?      file2.txt:23
```

Please note that the keys and values are separated by a TAB ('\t') character, and that whitespaces are not words.

Both lists are concatenated and sorted, resulting in:

```
a        file1.txt:13
a        file2.txt:12
file,    file2.txt:14
is       file1.txt:20
is       file1.txt:5
is       file2.txt:20
is       file2.txt:5
it?      file2.txt:23
just     file1.txt:8
not      file2.txt:8
test     file1.txt:15
This     file1.txt:0
This     file2.txt:0
```

This file of key-value pairs is then used as input to the "reducer" program, which computes the inverted index as previously shown. The corresponding shell script in bash to produce an inverted index is:

```
#!/bin/sh
export map_input_start=0
for f in file1.txt file2.txt; do
     export map_input_file=$f
     cat $f | ./mapper
done | sort -k 1,1 | ./reducer
```

The "mapper" and "reducer" commands in the script are the programs you need to implement. Store the bash script in a script file, e.g., `inverted-index.sh`, and make the script file executable, e.g., by issuing the command, `chmod +x ./inverted-index.sh`; make a test run. Check the result. It should correspond with the inverted-index example above.

Implementation issues

- Hadoop splits up the input files into several chunks, which are processed by different "mappers". To compute the right position of a word, you need to know at which offset the chunk of the current "mapper" starts. The environment variable, `map_input_start`, is assigned the offset at which the chunk starts. The variable can be read with the `getenv()` function in C++.

- Note that the “mapper” and “reducer” programs do not take any command line arguments and read their input from `stdin`.
- The names of the input text files are provided by Hadoop through an environment variable, `map_input_file`. Hadoop supplies the absolute filename in `map_input_file`, i.e., `hdfs:/path/to/file/filename`. Again, the variable can be read with the `getenv()` function in C++.
- In C++ you can use the iostream library to read input from `stdin`. For example, to read words from `stdin`, one alternative is shown below.

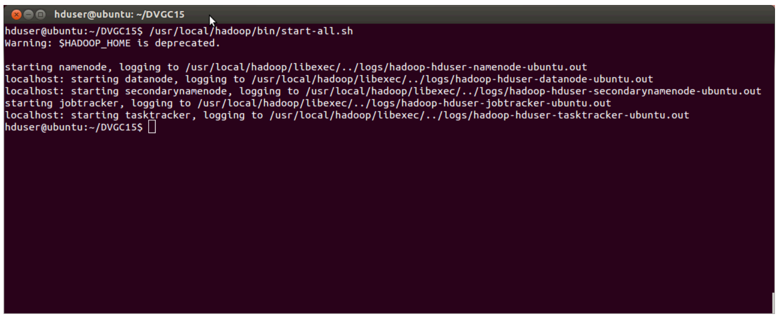
```
#include <iostream>
#include <string>
using namespace std;
int main() {
   string theWord
   do {
      cin >> theWord;
      if ( cin ) {
         cout << theWord << endl;
      }
   } while ( cin );
}
```

- In the “reducer” program, one option is to use a hash table to keep track of the words. One solution is to use the map container in the C++ Standard Library. Consider for example the following program that counts frequency of words from `stdin`:

```
#include <iostream>
#include <map>
#include <string>
using namespace std;
typedef map< string, unsigned > HashTable_t;
typedef HashTable_t::iterator HashTableIter_t;
int main() {
   HashTable_t theHashTable;
   string theWord;
   do {
      cin >> theWord;
      if ( cin ) {
         theHashTable[ theWord ] += 1;
      }
   } while( cin );
   for (  HashTableIter_t it = theHashTable.begin();
          it != theHashTable.end();
          it++ ) {
      cout << it->first << ", " << it->second << endl;
   }
}
```

Step 2: Running the MapReduce Application using Hadoop Streaming

Open up a terminal window, and position yourself in the folder in which your "mapper" and "reducer" applications reside. Hadoop is already installed on the virtual machine, however, has to be manually started. Thus, to begin with, invoke the following shell script in the terminal window: `/usr/local/hadoop/bin/start-all.sh`. This will startup a name node, data node, job tracker and a task tracker on your machine. The output should look similar to Figure 1.



```
hduser@ubuntu:~/DVGC15$ /usr/local/hadoop/bin/start-all.sh
Warning: $HADOOP_HOME is deprecated.

starting namenode, logging to /usr/local/hadoop/libexec/../logs/hadoop-hduser-namenode-ubuntu.out
localhost: starting datanode, logging to /usr/local/hadoop/libexec/../logs/hadoop-hduser-datanode-ubuntu.out
localhost: starting secondarynamenode, logging to /usr/local/hadoop/libexec/../logs/hadoop-hduser-secondarynamenode-ubuntu.out
starting jobtracker, logging to /usr/local/hadoop/libexec/../logs/hadoop-hduser-jobtracker-ubuntu.out
localhost: starting tasktracker, logging to /usr/local/hadoop/libexec/../logs/hadoop-hduser-tasktracker-ubuntu.out
hduser@ubuntu:~/DVGC15$
```

Figure 1. *Starting the Hadoop framework.*

Our "mapper" and "reducer" applications will read files from the `/in` folder and store the result in the `/out` folder of the Hadoop file system (HDFS). Thus, remove any previous output folder and create an input folder in HDFS:

```
fs -rmr /out
fs -mkdir /in
```

Create an inverted index of the two files, `file1.txt` and `file2.txt`; copy the files to the `/in` folder of the HDFS:

```
fs -copyFromLocal file1.txt file2.txt /in
```

To check that the files have been copied, issue the command:

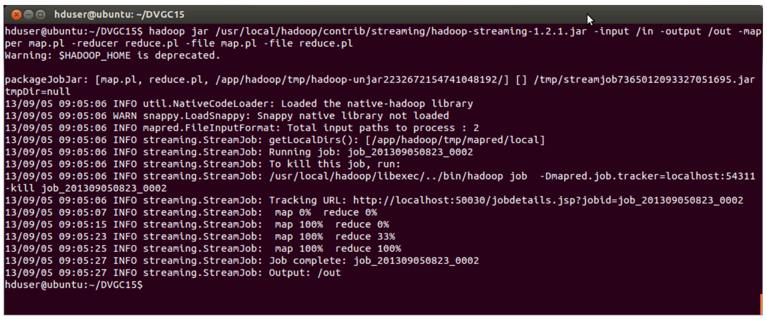
```
fs -ls /in
```

You should see a listing of the files in the `/in` folder.

To run the task with Hadoop Streaming, invoke the following Hadoop command:

```
hadoop jar /usr/local/hadoop/contrib/streaming/ ↵
hadoop-streaming-1.2.1.jar -input /in -output /out ↵
-mapper mapper -reducer reducer -file <pathname to executable>/mapper ↵
-file <pathname to executable>/reducer
```

The "mapper" and "reducer" are the programs you implemented in Step 1. The output should closely correspond with Figure 2.



```
hduser@ubuntu: ~/DVGC15
hduser@ubuntu:~/DVGC15$ hadoop jar /usr/local/hadoop/contrib/streaming/hadoop-streaming-1.2.1.jar -input /in -output /out -map
per map.pl -reducer reduce.pl -file map.pl -file reduce.pl
Warning: $HADOOP_HOME is deprecated.

packageJobJar: [map.pl, reduce.pl, /app/hadoop/tmp/hadoop-unjar2232672154741048192/] [] /tmp/streamjob7365012093327051695.jar
tmpDir=null
13/09/05 09:05:06 INFO util.NativeCodeLoader: Loaded the native-hadoop library
13/09/05 09:05:06 WARN snappy.LoadSnappy: Snappy native library not loaded
13/09/05 09:05:06 INFO mapred.FileInputFormat: Total input paths to process : 2
13/09/05 09:05:06 INFO streaming.StreamJob: getLocalDirs(): [/app/hadoop/tmp/mapred/local]
13/09/05 09:05:06 INFO streaming.StreamJob: Running job: job_201309050823_0002
13/09/05 09:05:06 INFO streaming.StreamJob: To kill this job, run:
13/09/05 09:05:06 INFO streaming.StreamJob: /usr/local/hadoop/libexec/../bin/hadoop job  -Dmapred.job.tracker=localhost:54311
-kill job_201309050823_0002
13/09/05 09:05:06 INFO streaming.StreamJob: Tracking URL: http://localhost:50030/jobdetails.jsp?jobid=job_201309050823_0002
13/09/05 09:05:07 INFO streaming.StreamJob:  map 0%  reduce 0%
13/09/05 09:05:15 INFO streaming.StreamJob:  map 100%  reduce 0%
13/09/05 09:05:23 INFO streaming.StreamJob:  map 100%  reduce 33%
13/09/05 09:05:25 INFO streaming.StreamJob:  map 100%  reduce 100%
13/09/05 09:05:27 INFO streaming.StreamJob: Job complete: job_201309050823_0002
13/09/05 09:05:27 INFO streaming.StreamJob: Output: /out
hduser@ubuntu:~/DVGC15$
```

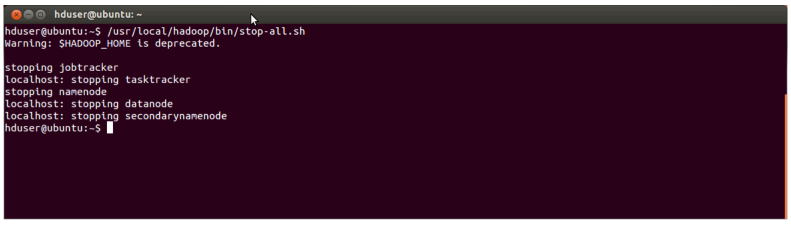
Figure 2. *Running the MapReduce job.*

Get the resulting inverted index of `file1.txt` and `file2.txt`, by issuing the command:

`fs -cat /out/part-00000`

Compare the inverted index with the one obtained in Step 1. Do they correspond?

When you are finished working with Hadoop, you need to shut it down. Invoke the following shell script in the terminal window: `/usr/local/hadoop/bin/stop-all.sh`. This will terminate the previously started processes. The output should look something like Figure 3.



```
hduser@ubuntu: ~
hduser@ubuntu:~$ /usr/local/hadoop/bin/stop-all.sh
Warning: $HADOOP_HOME is deprecated.

stopping jobtracker
localhost: stopping tasktracker
stopping namenode
localhost: stopping datanode
localhost: stopping secondarynamenode
hduser@ubuntu:~$
```

Figure 3. *Stopping the Hadoop framework.*

Assignment Presentation

The assignment is presented at a particular lab session (see course schedule). The presentation is made on the Linux virtual machine, `C:\vmware\dvgc15-ht17`. Make sure to print out and bring with you the marking page for assignment #2 to the lab presentation session. There is one marking page per student – not per group.

End of Assignment #2