Hadoop Tutorial
GridKa School 2011

Ahmad Hammad[*], Ariel García[†]
Karlsruhe Institute of Technology

September 7, 2011

Abstract

This tutorial intends to guide you through the basics of Data Intensive Computing with the Hadoop Toolkit. At the end of the course you will hopefully have an overview and hands-on experience about the Map-Reduce computing pattern and its Hadoop implementation, about the Hadoop filesystem HDFS, and about some higher level tools built on top of these like the data processing language Pig.

[*]hammad@kit.edu
[†]garcia@kit.edu

Contents

Hands-on block 1

Preparation

1.1 Logging-in

The tutorial will be performed in an existing Hadoop installation, a cluster of 55 nodes with a Hadoop filesystem HDFS of around 100 TB.

You should log-in via `ssh` into

```
hadoop.lsdf.kit.edu         # Port 22
```

from the login nodes provided to you by the GridKA School:

```
gks-1-101.fzk.de            # Port 24
gks-2-151.fzk.de
```

NOTE: Just use the same credentials (username and password) for both accounts.

1.2 Getting aquainted with HDFS

First we will perform some data management operations on the Hadoop Distributed Filesystem HDFS. The commands have a strong similarity to the standard Unix/Linux ones.

We will denote with the suffixes `HFile` and `HDir` file and folder names in the HDFS filsystem, and use `HPath` for either a file or a folder. Similarly, we use `LFile`, `LDir` and `LPath` for the corresponding objects in the local disk. For instance, some of the most useful HDFS commands are the following:

```
hadoop fs -ls /
hadoop fs -ls myHPath
hadoop fs -cat myHFile
hadoop fs -df

hadoop fs -cp srcHFile destHPath
hadoop fs -mv srcHFile destHPath
```

```
hadoop fs -rm myHFile
hadoop fs -rmr myHDir

hadoop fs -du myHDir
hadoop fs -mkdir myHDir
hadoop fs -get myHFile myCopyLFile
hadoop fs -put myLFile myCopyHFile
```

You can get all possible `fs` subcommands by typing

```
hadoop fs
```

Exercises:

1. List the top level folder of the HDFS filesystem, and find the location and contents of your **HDFS** user folder.
2. Find the size and the available space in the HDFS filesystem.
3. Create a new subfolder in your **HDFS** user directory.
4. Copy the README file from your HDFS user directory into the subfolder you just created, and check its contents.
5. Remove the subfolder you created above.

Hands-on block 2

MapReduce I: Hadoop Streaming

2.1 A simple Streaming example: finding the maximum temperature

The aim of this block is to get some first-hand experience on how Hadoop MapReduce works. We will use a simple Streaming exercise to achieve that, finding the highest temperature for each year in a real world climate data set.

Consider the following weather data set sample:

```
$ cat input_local/sample.txt
0067011990999991950051507004+68750+023550FM-12+038299999V0203301N00671220001CN9999999N9+00001+99999999999
0043011990999991950051512004+68750+023550FM-12+038299999V0203201N00671220001CN9999999N9+00221+99999999999
0043011990999991950051518004+68750+023550FM-12+038299999V0203201N00261220001CN9999999N9-00111+99999999999
0043012650999991949032412004+62300+010750FM-12+048599999V0202701N00461220001CN0500001N9+01111+99999999999
0043012650999991949032418004+62300+010750FM-12+048599999V0202701N00461220001CN0500001N9+00781+99999999999
                ^                                                                      ^    ^
              year                                                         +/- temperature  quality
         positions 16-19                                                   positions 88-92  93
                4                                                                        5  1
```

The temperature is multiplied by 10. The temperature value is considered MISSING if is equal to `+9999`. The value of the quality flag indicates the quality of the measurement. It has to match one of the following values: 0, 1, 4, 5 or 9; otherwise the temperature value is considered invalid.

Exercise: Write two scripts in a script language of your choice (like Bash, Python) to act as Map and Reduce functions for finding the maximum temperature for each year from the sample weather file `sample.txt`. These two scripts should act as described below.

The Map:

- reads the input data from standard input STDIN line-by-line
- parses every line by: **year**, **temperature** and **quality**
- tests if the parsed temperature is valid. That is the case when:

```
temp != "+9999" and re.match("[01459]", quality) // Python code
```

- outputs the **year** and the valid **temperature** as a tab-separated **key-value** pair string to standard output STDOUT.

The Reduce:

- reads data from standard input STDIN line-by-line
- splits the input line by the tab-separator to get the key and its value
- finds the maximum temperature for each year and prints it to STDOUT

2.1.1 Testing the map and reduce files *without* Hadoop

You can test the map and reduce scripts without using Hadoop. This helps to make clear the programming concept. Lets first check what the map output is:

```
$ cd mapreduce1
$ cat ../input_local/sample.txt | ./map.py
```

Now you can run the complete map-reduce chain, to obtain the maximum temperature for each year:

```
$ cat ../input_local/sample.txt | ./map.py | sort | ./reduce.py
```

2.1.2 MapReduce with Hadoop Streaming

1. Run the MapReduce Streaming job on the local file system. What is the calculated maximum temperature? for which year?

   Notice: write the following command all in one line, or use a backslash (\) at the end of each line as shown in point 2.

```
$ hadoop jar /usr/lib/hadoop/contrib/streaming/hadoop-streaming-0.20.2-cdh3u0.jar
    -conf     ../conf/hadoop-local.xml
    -input    ../input_local/sample.txt
    -output   myLocalOutput
    -mapper   ./map.py
    -reducer  ./reduce.py
    -file     ./map.py
    -file     ./reduce.py
```

2. Run the MapReduce Streaming on HDFS. Where and what is the output of the calculated max temperature of the job?

```
$ hadoop jar /usr/lib/hadoop/contrib/streaming/hadoop-streaming-0.20.2-cdh3u0.jar \
    -input    /share/data/gks2011/input/sample.txt \
    -output   myHdfsOutput \
    -mapper   ./map.py      \
    -reducer  ./reduce.py   \
    -file     ./map.py      \
    -file     ./reduce.py
```

Important: Before a repeating a run for a second time you always have to delete the output folder given with `-output` or you must select a new one, otherwise Hadoop will abort the execution.

```
$ hadoop fs -rmr myHdfsOutput
```

In case of the local file sytem run:

```
$ rm -rf myLocalOutput
```

2.1.3 Optional

Can you tell how many MapTasks and ReduceTasks have been started for this MR job?

Hands-on block 3

MapReduce II: Developing MR programs in Java

3.1 Finding the maximum temperature with a Java MR job

In this block you will repeat the calculation of the previous one using a *native* Hadoop MR program.

Exercise: Based on the file `MyJobSkeleton.java` in your `mapreduce2/` folder try to replace all question-mark placeholders (`?`) in the file `MyJob.java` to have a functioning MapReduce Java program, that can find the max temperature for each year as described in the previous block.

To test the program:

```
  # Create a directory for your compiled classes
$ mkdir myclasses
  # Compile your code
$ javac -classpath /usr/lib/hadoop/hadoop-core.jar \
        -d myclasses MyJob.java
  # Create a jar
$ jar -cvf myjob.jar -C myclasses .
  # Run
$ hadoop jar myjob.jar MyJob \
         /share/data/gks2011/input/sample.txt myHdfsOutput
```

Important: replace `gs099` with your actual account name.

3.2 Optional MapReduce exercise

Run the program with the following input:

`/share/data/gks2011/input/bigfile.txt`

1. What is the size of `bigfile.txt`?
2. List and cat the MR output file(s)
3. How many MapTasks and ReduceTasks have been started?
4. How can you make your MapReduce job faster?

Hands-on block 4

MapReduce III: User defined Counters

4.1 Understanding the RecordParser

Hadoop supports a quite sophisticated reporting framework for helping the user to keep track of his Hadoop job status.

Exercise: In the directory `mapreduce3/` you will find two Java files,

```
MyJobWithCounters.java
RecordParser.java
```

Please look into those files und understand what the `RecordParser` class does and how it is used in `MyJobWithCounters.java`.

4.2 Implementing user defined counters

Exercise: Implement your user-defined Counters' `Enum` and call the `incrCounter()` method to increment the right counter at the marked places in the code. Compile, create the jar, and run your MR job with either of the following input data:

```
/share/data/gks2011/input/all
/share/data/gks2011/input/bigfile.txt
```

What do your Counters report?

To compile the program do:

```
$ mkdir myclasses
$ javac -classpath /usr/lib/hadoop/hadoop-core.jar -d myclasses \
      RecordParser.java MyJobWithCounters.java
```

```
$ jar -cvf MyJobWithCounters.jar -C myclasses .
$ hadoop jar MyJobWithCounters.jar MyJobWithCounters \
        input/all outputcounts
$ hadoop jar MyJobWithCounters.jar MyJobWithCounters \
        input/bigfile.txt outputcounts2
```

Hands-on block 5

The *Pig* data processing language

5.1 Working with Pigs

Pig is a data flow language based on Hadoop. The Pig interpreter transforms your Pig commands into MapReduce programs which are then run by Hadoop, usually in the cluster infrastructure, in a way completely transparent for you.

5.1.1 Starting the interpreter

The Pig interpreter (called "Grunt") can be started in either of two modes:

local
: Pig programs are executed locally, **only** local files can be used (no HDFS)

MapReduce
: Pig programs are executed in the full Hadoop environment, with files in HDFS **only**

To run these modes use

```
pig -x local
pig -x mapreduce        # Default
```

Note: You can also create and run Pig scripts in batch (non-interactive) mode:

```
pig myPigScript.pig
```

Exercise: Start the Grunt shell –in local mode for now– and with reduced debugging:

```
pig -x local -d warn
```

Then get aquainted with some of the Pig shell's utility commands shown in Table 5.1. Remember that you started the shell in local mode, therefore you will be browsing the local filesystem –not HDFS!. Try, for instance,

```
grunt> help
grunt> pwd
grunt> fs -ls
grunt> ls
grunt> cp README-PLEASE.txt /tmp
grunt> cat /tmp/README-PLEASE.txt
grunt> fs -rm /tmp/README-PLEASE.txt
...
```

Utility commands	
help	Prints some help :-)
quit	Just that
set debug [on|off]	Enables verbose debugging
fs -<CMD>	HDFS commands (work also for local files)
ls, cp, cat, rm, rmr, ...	Same commands as above (less output)
cd	Change directory

Table 5.1: Grunt shell's utility commands

5.1.2 The Pig Latin language basics

The Pig language supports several data types: 4 scalar numeric types, 2 array types, and 3 composite data types as shown in Table 5.2. Note the examples on the rightmost column: "short" integers and "double floats" are the default types, otherwise the suffixes `L` or `F` need to be used. Important for understanding the Pig language and this tutorial are the tuples and bags.

Simple data types		
int	Signed 32 bit integer	`117`
long	Signed 64 bit integer	`117L`
float	32-bit floating point	`3.14F` or `1.0E6F`
double	64-bit floating point	`3.14` or `1.41421E2`
chararray	UTF8 character array (string)	`"Hello world!"`
bytearray	Byte array (binary object)	

Complex data types		
tuple	Ordered set of fields	`(1,"A",2.0)`
bag	Collection of tuples: unordered, possibly different tuple types	`{(1,2),(2,3)}`
map	Set of key value pairs: keys are unique chararrays	`[key#value]`

Table 5.2: The Pig Latin data types

Having said that, let's start "hands on" :-)

Exercise: Load data from a very minimal (local!) data file and learn how to peak at the data. The data file is a minimal climate data file containing mean daily temperature records, similar to the ones used earlier in this tutorial.

```
grunt> cd pig
grunt> data = LOAD 'sample-data.txt'
                AS (loc:long, date:chararray, temp:float, count:int);
grunt> DUMP data;
...
grunt> part_data = LIMIT data 5;
grunt> DUMP part_data;
...
```

Notice how you can dump all the data or just part of it using an auxiliary variable. Can you explain why one of the tuples in `data` appears as

```
(,YEAR_MO_DAY,,)  ?
```

Notice also that the real processing of data in Pig only takes place when you request some final result, for instance with `DUMP` or `STORE`. Moreover, you can also ask Pig about variables and some "sample data":

```
grunt> DESCRIBE data;
...
grunt> ILLUSTRATE data;
...
```

The variable (a.k.a. *alias*) `data` is a *bag of tuples*. The `illustrate` command illustrates the variable with different sample data each time... sometimes you might see a null pointer exception with our "unprepared" data: can you explain why?

Exercise: Now we will learn to find the maximum temperature in our small data set.

```
grunt> temps = FOREACH data GENERATE temp;
grunt> temps_group = GROUP temps ALL;
grunt> max_temp = FOREACH temps_group GENERATE MAX(temps);
grunt> DUMP max_temp;
(71.6)
```

As you see above, Pig doesn't allow you to directly apply a function (`MAX()`) to your data variables, but on a *single-column bag*.

> **Remember, Pig is not a normal programming language but a data processing language based on MapReduce and Hadoop!** This language structure is required to allow a direct mapping of your processing instructions to MapReduce!

Use `DESCRIBE` and `DUMP` to understand how the Pig instructions above work.

NOTE: if you change and reload a relation, like `temps =` above, you **must** reload also all dependent relations (`temps_group`, `max_temp`) to achieve correct results!

Exercise: Repeat the exercise above but converting the temperature to degrees Celsius instead of Fahrenheit:

$$T_{Celsius} = \frac{5}{9}(T_{Fahrenheit} - 32)$$

Hint: you can use mathematical formulas in the "GENERATE" part of a relation, but you cannot operate with the results of a function like `MAX()`. Don't forget that numbers without decimal dot are interpreted as integers!

Data IO commands	
LOAD	`a_1 = LOAD 'file' [USING function] [AS schema];`
STORE	`STORE a_2 INTO 'folder' [USING function];`
DUMP	`DUMP a_3;`
LIMIT	`a_4 = LIMIT a_3 number;`

Diagnostic commands	
DESCRIBE	`DESCRIBE a_5;` Show the schema (data types) of the relation
EXPLAIN	`EXPLAIN a_6;` Display the execution plan of the relation
ILLUSTRATE	`ILLUSTRATE a_7;` Display step by step how data is transformed (from the LOAD to the desired relation)

5.2 Using more realistic data

Above we have used a tiny data file with 20 lines of sample data. Now we will run Pig in MapReduce mode to process bigger files.

```
pig
```

Remember that now Pig with only allow you to use HDFS...

Exercise: Load data from a 200MB data file and repeat the above calculations. As the data files are now not TAB-separated –as expected by default by Pig– but space-separated, we need to explicitely tell Pig the `LOAD` function to use:

```
grunt> cd /share/data/gks2011/pig/all-years
grunt> data = LOAD 'climate-1973.txt' USING PigStorage(' ')
                AS (loc:long, wban:int, date:chararray,
                    temp:float, count:int);
grunt> part_data = LIMIT data 5;
```

```
grunt> DUMP part_data;
...
```

Check that the data was correctly loaded using the LIMIT or the ILLUSTRATE operators.

```
grunt> temps = FOREACH data GENERATE temp;
grunt> temps_group = GROUP temps ALL;
grunt> max_temp = FOREACH temps_group GENERATE MAX(temps);
grunt> DUMP max_temp;
(109.9)
```

Exercise: Repeat the above exercise measuring the time it takes to find the maximum in that single data file, and then compare with the time it takes to process the whole folder (13 GB instead of 200 MB). Pig can load all files in a folder at once if you pass it a folder path:

```
grunt> data = LOAD '/share/data/gks2011/pig/all-years/'
                USING PigStorage(' ')
                AS (loc:long, wban:int, date:chararray,
                    temp:float, count:int);
```

5.3 A more advanced exercise

In this realistic data set, the data is not perfect or fully cleaned up: if you look carefully, for instance, you will see a message

```
Encountered Warning FIELD_DISCARDED_TYPE_CONVERSION_FAILED 7996 time(s).
```

This is due to the label lines mixed inside the file:

```
STN--- WBAN YEARMODA TEMP ...
```

We will remove those lines from the input data by using the FILTER operator. As the warnings come from the castings in the LOAD operation, we now postpone the casts for a later step, after the filter was done:

```
grunt> cd /share/data/gks2011/pig/all-years
grunt> data_raw = LOAD 'climate-1973.txt' USING PigStorage(' ')
                    AS (loc, wban, date:chararray, temp, count);
grunt> data_flt = FILTER data_raw BY date != 'YEARMODA';
grunt> data = FOREACH data_flt GENERATE (long)loc, (int)wban,
                                  date, (float)temp, (int)count;
grunt> temps = FOREACH data GENERATE ((temp-32.0)*5.0/9.0);
grunt> temps_group = GROUP temps ALL;
grunt> max_temp = FOREACH temps_group GENERATE MAX(temps);
grunt> DUMP max_temp;
(43.27777862548828)
```

Also the mean daily temperatures were obtained from averaging a variable number of measurements: the amount is given in the 5^{th} column, variable count. You might want to filter all mean values obtained with less than –say– 5 measurements out. This is left as an exercise to the reader.

5.4 Some extra Pig commands

Some relational operators	
FILTER	Use it to work with tuples or rows of data
FOREACH	Use it to work with columns of data
GROUP	Use it to group data in a single relation
ORDER	Sort a relation based on one or more fields
...	

Some built-in functions	
AVG	Calculate the average of numeric values in a *single-column bag*
COUNT	Calculate the number of tuples in a bag
MAX/MIN	Calculate the maximum/minimum value in a *single-column bag*
SUM	Calculate the sum of values in a *single-column bag*
...	

Hands-on block 6

Extras

6.1 Installing your own Hadoop

The Hadoop community has its main online presence in:

`http://hadoop.apache.org/`

Although you can download the latest source code and release tarballs from that location, we strongly suggest you to use the more production-ready Cloudera distribution:

`http://www.cloudera.com/`

Cloudera provides ready to use Hadoop Linux packages for several distributions, as well as a Hadoop Installer for configuring your own Hadoop cluster, and also a VMWare appliance preconfigured with Hadoop, Hue, HBase and more.

Appendix A

Additional Information

Hadoop Homepage

Internet:	`http://hadoop.apache.org/`

Cloudera Hadoop Distribution

Internet:	`http://www.cloudera.com/`

Documentation

Tutorial:	`http://hadoop.apache.org/common/docs/r0.20.2/mapred_tutorial.html`
Hadoop API:	`http://hadoop.apache.org/common/docs/r0.20.2/api/`
Pig:	`http://pig.apache.org/docs/r0.9.0/`

Recommended books

Hadoop: The Definitive Guide	Tom White, O'Reilly Media, 2010 (2nd Ed.)
Hadoop in action	Chuck Lam, Manning, 2011