CS2510 Computer Operating Systems
Hadoop Examples Guide

The main objective of this document is to acquire some faimiliarity with the MapReduce and Hadoop computational model and distributed file system. Few examples, are explored, including the basic steps to compile, execute and display their output on MapReduce/Hadoop computing and files system infrastructure.

I. Checklist

1. HDFS account
2. You are allowed to access to Pitt Network

II. Background

Recall that you will be working with two different file systems. The first file system is the Andrew File System (AFS). In this file system, you create and compile your java program. Your executable code, however, must be run on Hadoop Distributed File System (HDFS), which runs on a 16-server cluster. Consequently, in addition to your executable code, all input and output files, that are required to run your program, should be located within HDFS.

In the examples discussed in this document, command lines start with either of the following prompts: **bash-S1$> or *$>**, followed by the actual command that you need to type**.** The actual shell prompts will be displayed based on your shell environment variables, defined in your .cshrc or .bash file**,** or whatever shell you chose to use. For example, the prompt command, for csh and tcsh, is set as follows:

set prompt = $HOST'{\!} '

III. Log into CS Cluster

Intentionally left blank

2. Login to Hadoop server
   $>ssh -l {your user name}* ~~*hadoop.cs.pitt.edu*~~ had6110.cs.pitt.edu

Again, use your CS account password. You should be in your AFS home directory now.
To get your home directory path, type following command:

bash-S1$>*pwd*

/afs/cs.pitt.edu/usr0/{your user name}

Tips : This path can be substituted by ***$HOME*** variable.

IV. MaxTemperature Example – Compilation and Execution Steps

Overview

This example determines the maximum weather temperature for every year, for which data are collected. Data sample is taken from National Climatic Data Center (NCDC), which are stored using line-oriented ASCII format, in which each line is a record. Followings are the example of record format of each line :

```
0057
332130          # USAF weather station identifier
99999           # WBAN weather station identifier
19500101        # observation date
0300            # observation time
4
+51317                  # latitude (degrees x 1000)
+028783         # longitude (degrees x 1000)
FM-12
+0171           # elevation (meters)
99999
V020
320              # wind direction (degrees)
1                # quality code
N
0072
1
00450           # sky ceiling height (meters)
1               # quality code
CN
010000          # visibility distance (meters)
1 # quality code
N9
-0128           # air temperature (degrees Celsius x 10)
1               # quality code
-0139           # dew point temperature (degrees Celsius x 10)
1               # quality code
10268           # atmospheric pressure (hectopascals x 10)
1               # quality code
```

This example only uses two data files, data from year 1901 and 1902

Preparation

Copy the required files from following path:

bash-S1$> cp -r /afs/cs.pitt.edu/usr0/znati/public/CS2510/ch02.zip .

(Make sure that you don't miss the '**.**' at the end of this command). This should copy file **ch02.zip** into your home directory.

Next, create a folder (you can name it anything, but this example uses a folder called ***CS2510***) in your home directory

bash-S1$>*mkdir $HOME /CS2510*

Then, extract the file ch02.zip into the folder above:

bash-S1$>*unzip ch02.zip -d $HOME /CS2510*

Change your current directory to ch02

bash-S1$>*cd $HOME /CS2510/ch02*
bash-S1$>*pwd*
/afs/cs.pitt.edu/usr0/{your user name}/CS2510/ch02

There should be 3 folders within ***ch02***. They are ***src, input, and class***. ***src*** is the folder in which all java source files are located. ***Input*** is the folder in which the required input files are located. ***Class*** is the folder in which the *.class file will be created in the subsequent steps. This folder should be empty at this time. There should be three source files under directory ***src***(**.java* files):

- ***MaxTemperatureMapper.java***
- ***MaxTemperatureReducer.java***
- ***MaxTemperature.java***

There are two input files under directory ***input:***

- ***1901***
- ***1902***

Moving input files from AFS to HDFS

Hadoop can only read input from HDFS, hence, we have to move the input files to the HDFS space. First, you need to create the destination folder in your HDFS. Your HDFS root path is ***/user/{your HDFS account}*** (the {your HDFS account} is the same as your CS account). Before proceeding with this step, you must type the following command, to list the files you currently have in the Hadoop space:

bash-S1$> export PATH="$PATH:/opt/hadoop/bin"

bash-S1$>*hadoop fs -ls*

Unless you worked with Hadoop previously, the command should not list anything, because no folders exist within your HDFS space. To proceed, create the folders, *ch02* and *input*, within your HDFS current directory:

bash-S1$>*hadoop fs -mkdir ch02*
bash-S1$>*hadoop fs -mkdir ch02/input*

Now, run the same command again to list the directory under ch02.

bash-S1$*>hadoop fs -ls*
drwx------ - xex1 supergroup 0 2013-01-15 17:51 /user/{your HDFS account name}/input

Then copy the input files to HDFS

bash-S1$*>hadoop fs -put input/* ch02/input*

You can't move forward unless this step has been completed

Compiling the program into a jar file

To compile a MapReduce program, you need the hadoop library. (Use parameter –classpath, library location: ***/usr/share/hadoop/hadoop-core-1.0.1.jar***). After compiling all **.java* files, the ***.****class* files will be put into a dedicated folder, called ***class*** (This folder already exists, as it was created when the files are extracted using unzip).

The nest step is to compile your program, as follows: (Carefull when copying)

```
$> javac -verbose -classpath /opt/hadoop/share/hadoop/common/hadoop-common-2.7.2.jar:/opt/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-client-core-2.7.2.jar -d class src/*.java
```

The compilation outcome (*.class* files) is located in the directory ***class.*** Next step is to pack and combine your **.class* files into one single **.jar* archive file. This is important because all required classes to run your program should be in one package and Hadoop doesn't have to search the class from other locations or libraries to run your program. To do this, you need to use '**jar**' command. This is a Java archive tool. Now, combine your **.class* files into one single *.jar* file:

bash-S1$*> jar -cvf MaxTemperature.jar -C class/* **.**

The above command should create an archive files (as requested by the command options **–cvf**) called ***MaxTemperature.jar***

The command option " **–C class/ .**" tells **jar** command to search the **.class* files on this folder (since our compiling step mentioned "**–d class**" for the output location in previous step) before getting back to current location (denoted by '**.**', which is also ***$HOME /CS2510/ch02***) .The **.jar* file will be created in ***$HOME /CS2510/ch02***

Executing MaxTemperature on Hadoop

To run the program, Hadoop **jar** command is used. Please note that this command is **different** from the one you used in the previous step. The syntax of the command is as follow:

hadoop jar {your jar file} {your main class name} [arg1] [arg2] …

To run your program, type and execute following line :

bash-S1$*>hadoop jar MaxTemperature.jar MaxTemperature ch02/input ch02/output*

- **{your jar file}** is ***MaxTemperature.jar***,
- **{your main class name}** is ***MaxTemperature,***
- **[arg1]** is ***ch02/input,*** and
- **[arg2]** is ***ch02/output***.

The command above will instruct Hadoop to read the ***MaxTemperature.jar*** file and run the executable code, ***MaxTemperature.class***. It reads the files located in ***/user/{your HDFS account***

name}/ch02/input and stores the result of the execution into ***/user/{your HDFS account name}/ch02/output.*** These paths are in HDFS file system. Please note that YOU SHOULD NOT create the ***cho02/output*** folder. It is automatically created by the Hadoop runtime system. If you re-run this example, you must use a different folder name for output or remove the existing output directory before your run the program. For example, the following command deletes the ***output*** folder in ***/user/{your HDFS account name}/ch02***

hadoop fs –rmr ch02/output

Print/copy the result from HDFS to AFS

As mentioned above, if the application executes successfully, the output would be stored in ***/user/{your HDFS account name}/ch02/output***. To see the content, stored in this HDFS folder, type following command:

bash-S1$*>hadoop fs -ls ch02/output/*
-rw------- 3 ...(detail omitted)..... /user/{your HDFS account name}/output/_SUCCESS
-rw------- 3 ...(detail omitted)..... /user/{your HDFS account name}/output/part-r-00000

The output is printed on file part* (this may vary across different examples. In this case, it is ***part-r-00000***). To print the result from HDFS, type command below:

bash-S1$*>hadoop fs -cat ch02/output/part-r-00000*

The cat command should produce the following outcome:

1901 317
1902 244

You can also copy the output to your local directory y and display it to monitor, as follows:

bash-S1$*>hadoop fs -get ch02/output* **.**
bash-S1$*>cat ch02/output/part-r-00000*

V. WordCount Example – Compilation and Execution Steps

Overview

This example uses MapReduce() to map the words and count their numbers as they are read from file(s). This example reads the input files, parse the line using the space character as the delimiter, count the number of occurences of each word, and finally produce the result in the form of <Word count> pair. For the first run, you will use two small files for input, ***File01*** and ***File02***.

- ***File01*** contains following line :
  Hello World Bye World
- ***File02*** contains following line :
  Hello Hadoop Goodbye Hadoop

Preparation

The preparation steps are basically similar to those of MaxTemperature example. Copy the required files from following location:

bash-S1$>*cp –r ~znati/public/CS2510/wordcount.zip $HOME/* This did not work

cp -r /afs/cs.pitt.edu/usr0/znati/public/CS2510/wordcount.zip . This worked instead.

This should copy file ***wordcount.zip*** into your home directory.

The next step consists of extracting the file ***wordcount.zip*** into a folder ***CS2510*** (assuming that folder has existed from previous example (MaxTemperature)); then change your current directory to ***wordcount.*** Type following commands:

bash-S1$>*unzip $HOME/wordcount.zip -d $HOME/CS2510*
bash-S1$>*cd $HOME/CS2510/wordcount*
bash-S1$>*pwd*
/afs/cs.pitt.edu/usr0/{your user name}/CS2510/wordcount

There are 3 folders in this directory. They are ***src***, ***input*** and ***class***, respectively.
There should only one source file under directory ***src***(*.java files):

- ***WordCount.java***

There are two input files under directory ***input:***

- ***File01***
- ***File02***

Moving input files from AFS to HDFS

Hadoop can only read input from HDFS. Thus, we have to move the files to HDFS space. First, create the destination folder in HDFS, then move the input files into the destination folder. The following commands can be used to create the appropriate directories and move the input files in their associated directory:

bash-S1$>*hadoop fs -mkdir wordcount*
bash-S1$>*hadoop fs -mkdir wordcount/input*
bash-S1$>*hadoop fs -put input/* wordcount/input* from afs wordcount folder which has input folder in it

Compiling the program into a jar file

The next step is to compile the program: javac -verbose -classpath /usr/share/hadoop/hadoop-core-1.0.1.jar -d class src/*.java (space after classpath needed) ** this command does not work, use the command for maxtemperature up above

bash-S1$>*javac -verbose -classpath/usr/share/hadoop/hadoop-core-1.0.1.jar -d class src/*.java*

The compiled files (.class) are put in the directory ***class.*** Pack the *.class files into a single *.jar archive file, called ***wordcount.jar*** by, using the following command:

bash-S1$> *jar -cvf wordcount.jar -C class/ .*

Please refer to MaxTemperature example for more detailed explanation on how to perform this step.

Executing WordCount() on Hadoop

To run the WordCount() program, the **hadoop jar** command is used:

bash-S1$>*hadoop jar wordcount.jar WordCount wordcount/input wordcount/output*

Please refer to previous example for more detailed explanation on this command. The command above will run this example on Hadoop. It reads the files located in ***/user/{your HDFS account name}/wordcount/input*** and put the result into ***/user/{your HDFS account name}/wordcount/output*** These paths are in HDFS file system. As mentioned in the previous example, the folder ***wordcount/output*** should not be created before executing this example. The folder will be created by Hadoop runtime system, after executing the program. If you re-run this example, use a different folder name or remove the existing folder. Please refer to previous example on how to remove the folder in HDFS system.

Print the result / copy it from HDFS to AFS

If the application runs successfully, the results will be stored in ***/user/{your HDFS account name}/wordcount/****output.* To see the content stored in the HDFS folder, type following command:

```
bash-S1$>hadoop fs -ls wordcount/output/
-rw-------   3 xex1 supergroup    0 2013-01-16 16:25 /user/{your HDFS account name}/output/_SUCCESS
-rw-------   3 xex1 supergroup 18 2013-01-16 16:25 /user/{your HDFS account name}/output/part-00000
```

To display the result, you can read it from HDFS by typing this following command:

```
bash-S1$>hadoop fs -cat wordcount/output/part-00000
```

And you should see the following:

```
Bye     1
Goodbye 1
Hadoop  2
Hello   2
World   2
```

Or, you can copy it to your AFS space and print it to the screen:

```
bash-S1$>hadoop fs -get wordcount/output .
bash-S1$>cat wordcount/output/part-00000
```

```
Bye     1
Goodbye 1
Hadoop  2
Hello   2
World   2
```

Working with more input files

Up to this point, you have completed a simple task using the word count example. However, that example may not be sufficient to give you a sense of how Map Reduce can work with large data stes. Therefore, the extension to the previous example, discussed below, will guide you on how to use more than one input file, and will show you how MapReduce handles large amount of data.

To get the new set of input files, use the following commands :

bash-S1$>*cp –r ~znati/public/CS2510/shake.zip $HOME*

Then unzip them into the ***CS2510*** folder, as follows:

bash-S1$>*unzip $HOME/shake.zip -d $HOME/CS2510*

The next step is to copy the files to the HDFS space (List them before you copy them, to find out what they are). If you do not want to overwrite an existing input file, you can create a new folder in HDFS, say ***input2***, as shown in the sequence of steps listed below:

bash-S1$>*hadoop fs -mkdir wordcount/input2*
bash-S1$>*hadoop fs -put $HOME/CS2510/shake/* wordcount/input2*

Then, execute your program (you do not need to compile them anymore)

bash-S1$>*hadoop jar wordcount.jar WordCount wordcount/input2 wordcount/output2*

The command above will read the input files from ***/user/{your HDFS account name}/wordcount/input2*** and put the result into ***/user/{your HDFS account name}/wordcount/output2***. This example uses ***output2*** as the folder name in which the result will be stored. To disply the results, follow the steps above (remember to replace ***output*** with ***output2***). hadoop fs -ls wordcount/output/

VI. Interacting with the Hadoop Distributed Filesystem API – Part I

Overview

In this section, we explore the Hadoop File System class, namely the API to interact with the Hadoop filesystem.

Preparation

Copy the required files from following path:

bash-S1$>*cp -r ~znati/public/CS2510/ch03.zip $HOME*

This should copy file **ch03.zip** into your home directory. Next, create a folder (you can name it anything, but this example uses a folder called ***CS2510***) in your home directory (skip this following line if folder exists already).

bash-S1$>*mkdir $HOME/CS2510*

Then, extract the file ch03.zip into the folder above :

bash-S1$>*unzip ch03.zip -d $HOME/CS2510*

Change your current directory to ch03

bash-S1$>*cd $HOME/CS2510/ch03*
bash-S1$>*pwd*

/afs/cs.pitt.edu/usr0/{your user name}/CS2510/ch03

There should be 3 folders within ***ch03***. They are ***src, input, and class*****.** ***src*** is the folder in which all java source files are located. ***input*** is the folder in which required input files are located. ***class*** is the folder in which the *.class file will be created (on later step). This folder should be empty at this time.
There should be three source files under directory ***src***(*.*java* files)

- ***DateRangePathFilter.java***
- ***FileSystemDoubleCat.java***
- ***RegexPathFilter.java***
- ***FileCopyWithProgress.java***
- ***ListStatus.java***
- ***URLCat.java***
- ***FileSystemCat.java***
- ***RegexExcludePathFilter.java***
-

There are two input files under directory ***input***

- ***1400-8.txt***
- ***quangle.txt***

Moving input files from AFS to HDFS

In this section, we will use one of the cluster machines, namely s2.mate.

bash-S1$*>hadoop fs -mkdir ch03*
bash-S1$*>hadoop fs -mkdir ch03/input*
(the line above will create directory ch03/input in your HDFS)
bash-S1$*>hadoop fs -copyFromLocal input/quangle.txt hdfs://s2.mate/user/{your HDFS account name}/ch03/input*

Pay attention to the use of ***hdfs://s2.mate/user/$user/ch03/input*** as the destination.

Compiling the program into a jar file

To compile a MapReduce program, you need the hadoop library. (Use parameter –classpath, library location:***/usr/share/hadoop/hadoop-core-1.0.1.jar***). After compiling all *.*java* files, the ***.****class* files will be put into a dedicated folder, called ***class***(This folder has already existed as it came from extraction step back then)

Next step is to compile the program, as follows:

bash-S1$>*javac -verbose -classpath /usr/share/hadoop/hadoop-core-1.0.1.jar -d class src/*.java*

The compilation result (.*class* files) is located in the directory ***class.*** The next step is to pack and combine your *.*class* files into one single *.*jar* archive file. This is important because all required classes to run your program would be in one package and Hadoop doesn't have to search the class from another location or library to run your program. To do this, you need to use '**jar**' command. This is Java archive tool. Now, combine your*.*class* files into one single .*jar* file:

bash-S1$*> jar -cvf ch03.jar -C class/* **.**

That command would create an archive files (as depicted by the command options **–cvf**) called ***ch03.jar*****.** The command option " **–C class/.***"* tells **jar** command to search the *.*class* files on this folder

(since the compiling step mentioned "**–d class**" for the output location in previous step) before getting back to the current location (denoted by '**.**', which should be ***$HOME /CS2510/ch03***) .The **.jar* file will be created in ***$HOME /CS2510/ch03***

Running the Examples

Example 1 – Reading Data from a Hadoop URL: **URLCat**.
URLCat is a progran for displaying files from Hadoop filesystems on standard output, like the Unix **cat** command

> ***bash-S1$>****hadoop jar ch03.jar URLCat hdfs://s2.mate/user/{your HDFS account name}/ch03/input/quangle.txt*

The following output should be displayed:

```
On the top of the Crumpetty Tree
The Quangle Wangle sat,
But his face you could not see,
On account of his Beaver Hat.
```

Example 2 – Reading Data Using the FileSystem API: **FileSystemCat**
FileSystemCat is a progran that display files from a Hadoop filesystem on standard output by using the FileSystem directly.

> ***bash-S1$>****hadoop jar ch03.jar FileSystemCat hdfs://s2.mate/user/{your HDFS account name}/ch03/input/quangle.txt*

The following output should be displayed:

```
On the top of the Crumpetty Tree
The Quangle Wangle sat,
But his face you could not see,
On account of his Beaver Hat.
```

*Example 3 – Reading Data Using **seek***: **FileSystemDoubleCat**
FileSystemDoubleCat is a progran that display files from a Hadoop filesystem on standard output twice by using **seek**.

bash-S1$>*hadoop jar ch03.jar FileSystemDoubleCat hdfs://s2.mate/user/{your HDFS account name}/ch03/input/quangle.txt*

The following output should be displayed:

```
On the top of the Crumpetty Tree
The Quangle Wangle sat,
But his face you could not see,
On account of his Beaver Hat.
On the top of the Crumpetty Tree
The Quangle Wangle sat,
But his face you could not see,
On account of his Beaver Hat.
```

Example 4 – Writing Data: **FileCopyWithProgress**
FileCopyWithProgress is a progran that copy a local fileto a Hadoop filesystem.
There's also an overloaded method for passing a callback interface, **Progressable**, so this program can be notified about the progress of the data being written to the datanodes:

bash-S1$>*hadoop jar ch03.jar FileCopyWithProgress input/1400-8.txt hdfs://s2.mate/user/{your HDFS account name}/ch03/input/1400-8.txt*

The progress is illustrated by printing a dot every time the **progress()** method is called by Hadoop, which is after each 64K packet of data is written to the data node pipeline.

The following output should be displayed:

```
.............
```

Example 5 – Querying the Filesystem: **ListStatus**
ListStatus is a progran that shows the file statuses for a collection of paths in a Hadoop filesystem.

> ***bash-S1$>****hadoop jar ch03.jar ListStatus hdfs://s2.mate/ hdfs://s2.mate/user/{your HDFS account name}*

The following output should be displayed:

```
hdfs://s2.mate/mapred
hdfs://s2.mate/tmp
hdfs://s2.mate/user
hdfs://s2.mate/user/{user name}/.Trash
hdfs://s2.mate/user/{user name}/.staging
hdfs://s2.mate/user/{user name}/ch02
hdfs://s2.mate/user/{user name}/ch03
hdfs://s2.mate/user/{user name}/input
```

VII. Interacting with Hadoop Distributed Filesystem API – Part II

Overview

In this section, we would work with other the Hadoop's FileSystem class on following purpose : delete the file and filter the list of the files..

1. Preparation

Copy the required files from following path :

> **bash-S1$>***cp -r $HOME/../xex1/public/ch03-2.zip $HOME*

This should copy file **ch03-02.zip** into your home directory.
Next, create a folder (you can name it anything, but this example uses a folder called ***CS2510***) in your home directory (skip this following line if folder already exists)

> **bash-S1$>***mkdir $HOME/CS2510*

Then, extract the file ch03-2.zip into the folder above :

> **bash-S1$>***unzip ch03-2.zip -d $HOME/CS2510*

Change your current directory to ch03-2

bash-S1$>*cd $HOME/CS2510/ch03-2*
bash-S1$>*pwd*
/afs/cs.pitt.edu/usr0/{your user name}/CS2510/ch03-2

There should be 3 folders within ***ch03-2***. They are ***src, input, and class. src*** is the folder in which all java source files are located. ***input*** is the folder in which required input files are located. ***class*** is the folder in which the *.class file will be created (on later step). This folder should be empty at this time.
There should be three source files under directory ***src***(**.java* files)

- ***Delete2.java***
- ***Filter.java***

There are 7 input files under directory ***input.*** They are :

- ***3111***
- ***3112***
- ***3113***
- ***3121***
- ***3122***
- ***3123***
- ***3124***

Moving input files from AFS to HDFS

In this section, we will use one of the cluster machines, namely s2.mate.

bash-S1$*>hadoop fs -mkdir ch03-2*
bash-S1$*>hadoop fs -mkdir ch03-2/input*
(those two lines above create directory ch03/input in your HDFS)
bash-S1$*>hadoop fs -copyFromLocal input/* hdfs://s2.mate/user/{your HDFS account name}/ch03-2/input*

The ***hdfs://s2.mate/user/$user/ch03-2/input*** used as the destination.

Compiling the program into a jar file

The next step is to compile your program:
bash-S1$>*javac -verbose -classpath /usr/share/hadoop/hadoop-core-1.0.1.jar -d class src/*.java*

The compilation outcome(*.class* files) is located in the directory ***class.*** Next step is to pack and combine your **.class* files into one single **.jar* archive file
Now, combine your**.class* files into one single *.jar* file:
bash-S1$*> jar -cvf ch03-2.jar -C class/* **.**

That command would create an archive files (as depicted by command options **–cvf**) called ***ch03-2.jar***

Runing the the examples

a. Delete a file in Hadoop URI :**Delete2**.
First, list the content of following path in HDFS location
bash-S1$*>hadoop fs -ls hdfs://s2.mate/user/{your HDFS account name}/ch03-2/input*
-rw---- .....(detail omitted)...../user/{your HDFS account name}/ch03-2/input/3111
rw---- .....(detail omitted)...../user/{your HDFS account name}/ch03-2/input/3112

rw---- .....(detail omitted)...../user/{your HDFS account name}/ch03-2/input/3124

(Note : you can also use ListStatus program from previous example to list your directory's content). In this example, we are going to delete file 3124. To do so, execute following command:

bash-S1$*>hadoop jar ch03-2.jar Delete2 hdfs://s2.mate/user/{your HDFS account name}/ch03-2/input/3124*

Now, check your directory content. You should not see that file there any longer

b. Filter a file list

This example will show you how you can filter the file list based on given pattern. Assume that you want to list only the files started with 311*. ('*' means matched to any single or more characters). You can run the example as follow :

bash-S1$*>hadoop jar ch03-2.jar Filter hdfs://s2.mate/user/{your HDFS account name}/ch03-2/input/311**

-rw---- .....(detail omitted)...../user/{your HDFS account name}/ch03-2/input/3111
-rw---- .....(detail omitted)...../user/{your HDFS account name}/ch03-2/input/3112
-rw---- .....(detail omitted)...../user/{your HDFS account name}/ch03-2/input/3113

In this example, you can also exclude one ore more files from the list. For instance, you would like to exclude 3112 from your list. Then you can type as follow :

bash-S1$*>hadoop jar ch03-2.jar Filter hdfs://s2.mate/user/{your HDFS account name}/ch03-2/input/311* ^.*/3112*

hdfs://s2.mate/user/{your HDFS account name}/ch03-2/input/3111
hdfs://s2.mate/user/{your HDFS account name}/ch03-2/input/3113

Remember to type "^**.**" before the excluded pattern. You can also specify the range of the files you want to show or exclude. For example, you want to list only files with last character is '2' or '3' (then only file 3112, 3113, 3122 and 3123 should be displayed). Here is the command :

bash-S1$*>hadoop jar ch03-2.jar Filter hdfs://s2.mate/user/{your HDFS account name}/ch03-2/input/31?[2-3]*

hdfs://s2.mate/user/{your HDFS account name}/ch03-2/input/3112
hdfs://s2.mate/user/{your HDFS account name}/ch03-2/input/3112
hdfs://s2.mate/user/{your HDFS account name}/ch03-2/input/3122
hdfs://s2.mate/user/{your HDFS account name}/ch03-2/input/3123

Useful links

The following links provide additional insight into Hadoop and helpful information about setting and executing MapReduce programs on Hadoop Distributed File System.

[1] Hadoop Project : http://hadoop.apache.org
[2] CS Technical Support :https://intranet.cs.pitt.edu/~tech/
[3] Wordcount Example : http://hadoop.apache.org/docs/r1.0.4/mapred_tutorial.html
[4] Hadoop cCmmand Guide : http://hadoop.apache.org/docs/r1.0.4/commands_manual.html
[5] Hadoop File System Shell Guide :http://hadoop.apache.org/docs/r1.0.4/file_system_shell.html
[6] API Docs Map and Reduce : http://hadoop.apache.org/docs/r1.0.4/api/index.html

Acknowledgement

Several people helped putting together this document. Special thanks go to **Xerandy**, Telecommunication Dept., School of Information Sciences, University of Pittsburgh, and **Youming Liu**, Computer Science Dept., University of Pittsburgh