**Program-1: Install and run Hadoop in standalone mode, pseudo mode and fully distributed cluster.**

**Step – 1: Java installed on your system**

Open terminal and fire:

$ java -version

If you have already installed Java, move to 2.1. If not, follow the steps:

$ sudo apt-get update  
$ sudo apt-get install default-jdk

Now check the version once again

$ java -version

**Standalone mode.**

**Step-2:** [Download the latest version of Hadoop here](http://hadoop.apache.org/releases.html).

$ tar -xzvf hadoop-2.7.3.tar.gz

*//Change the version number if needed to match the Hadoop version you have downloaded.//*

**Step-3:**Now we are moving this extracted files to /usr/local, suitable for local installs.

$ sudo mv hadoop-2.7.3 /usr/local/hadoop

**Step-4:** Now go to the Hadoop distribution directory using terminal

$ cd /usr/local/hadoop

Lets see whats inside the Hadoop folder

etc — has the configuration files for Hadoop environment.

bin — include various commands useful like Hadoop cmdlet.

share — has the jars that is required when you write MapReduce job. It has Hadoop libraries

**Step-5:** Hadoop command in the bin folder is used to run jobs in Hadoop.

$ bin/hadoop

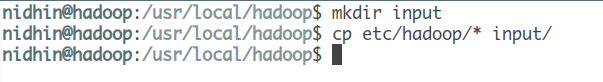
**Step-6:** jar command is used to run the MapReduce jobs on Hadoop cluster

$ bin/hadoop jar

**Step-7:** Now we will run an example MapReduce to ensure that our standalone install works

create a input directory to place the input files and we run MapReduce command on it. These are the configuration and command files along with hadoop, we will use those as text file input for our MapReduce.

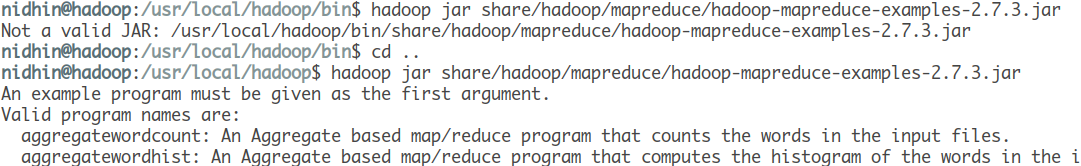
$ mkdir input   
$ cp etc/hadoop/\* input/



This is run using the bin/hadoop command. It is used to run MapReduce. Jarindicates that the MapReduce operation is specified in a Java archive. Here we will use the Hadoop-MapReduce-examples.jar file which come along with installation. Jar name differ based on the version you are installing. Now move on to your Hadoop install directory and type:

$ Hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples- 2.7.3.jar

*If you are not in the correct directory, you will get an error saying “Not a valid JAR” as shown below. If this issue persist, check if the location of Jar file is correct for your system.*



Running example to check working of standalone mode.

This is an MapReduce ran successfully on standalone setup.

**Program-2: Wordcount Map Reduce program using standalone Hadoop.**

### Algorithm

1: class Mapper

2: method Map(docid a; doc d)

3: for all term t 2 doc d do

4: Emit(term t; count 1)

1: class Reducer

2: method Reduce(term t; counts [c1; c2; : : :])

3: sum 0

4: for all count c 2 counts [c1; c2; : : :] do

5: sum sum + c

### 6: Emit(term t; count sum)

## Running WordCound

Run the [wordcount](http://www.science.smith.edu/dftwiki/index.php/Hadoop_WordCount.java) java program from the example directory in hadoop:

hadoop@hadoop1:~/352/dft$ **hadoop jar /home/hadoop/hadoop/hadoop-0.19.2-examples.jar wordcount dft dft-output**

The program takes about **21 seconds** to execute on a 5-PC cluster. The output generated is something like this:

10/03/16 11:40:51 INFO mapred.FileInputFormat: Total input paths to process : 1

10/03/16 11:40:51 INFO mapred.JobClient: Running job: job\_201003161102\_0002

10/03/16 11:40:52 INFO mapred.JobClient: map 0% reduce 0%

10/03/16 11:40:55 INFO mapred.JobClient: map 9% reduce 0%

10/03/16 11:40:56 INFO mapred.JobClient: map 27% reduce 0%

10/03/16 11:40:58 INFO mapred.JobClient: map 45% reduce 0%

10/03/16 11:40:59 INFO mapred.JobClient: map 81% reduce 0%

10/03/16 11:41:01 INFO mapred.JobClient: map 100% reduce 0%

10/03/16 11:41:09 INFO mapred.JobClient: Job complete: job\_201003161102\_0002

10/03/16 11:41:09 INFO mapred.JobClient: Counters: 17

10/03/16 11:41:09 INFO mapred.JobClient: File Systems

10/03/16 11:41:09 INFO mapred.JobClient: HDFS bytes read=1576605

10/03/16 11:41:09 INFO mapred.JobClient: HDFS bytes written=527522

10/03/16 11:41:09 INFO mapred.JobClient: Local bytes read=1219522

10/03/16 11:41:09 INFO mapred.JobClient: Local bytes written=2439412

10/03/16 11:41:09 INFO mapred.JobClient: Job Counters

10/03/16 11:41:09 INFO mapred.JobClient: Launched reduce tasks=1

10/03/16 11:41:09 INFO mapred.JobClient: Rack-local map tasks=6

10/03/16 11:41:09 INFO mapred.JobClient: Launched map tasks=11

10/03/16 11:41:09 INFO mapred.JobClient: Data-local map tasks=5

10/03/16 11:41:09 INFO mapred.JobClient: Map-Reduce Framework

10/03/16 11:41:09 INFO mapred.JobClient: Reduce input groups=50091

10/03/16 11:41:09 INFO mapred.JobClient: Combine output records=88551

10/03/16 11:41:09 INFO mapred.JobClient: Map input records=33055

10/03/16 11:41:09 INFO mapred.JobClient: Reduce output records=50091

10/03/16 11:41:09 INFO mapred.JobClient: Map output bytes=2601773

10/03/16 11:41:09 INFO mapred.JobClient: Map input bytes=1573044

10/03/16 11:41:09 INFO mapred.JobClient: Combine input records=267975

10/03/16 11:41:09 INFO mapred.JobClient: Map output records=267975

10/03/16 11:41:09 INFO mapred.JobClient: Reduce input records=88551

## Getting the Output

* Let's take a look at the output of the program:

hadoop@hadoop1:~/352/dft$ **hadoop dfs -ls**

Found x items

drwxr-xr-x - hadoop supergroup 0 2010-03-16 11:36 /user/hadoop/dft

drwxr-xr-x - hadoop supergroup 0 2010-03-16 11:41 /user/hadoop/dft-output

Verify that a new directory with *-output* at the end of your identifier has been created.

* Look at the contents of this output directory:

hadoop@hadoop1:~/352/dft$ **hadoop dfs -ls dft-output**

Found 2 items

drwxr-xr-x - hadoop supergroup 0 2010-03-16 11:40 /user/hadoop/dft-output/\_logs

-rw-r--r-- 2 hadoop supergroup 527522 2010-03-16 11:41 /user/hadoop/dft-output/part-00000

* Finally, let's take a look at the output

hadoop@hadoop1:~/352/dft$ **hadoop dfs -cat dft-output/part-00000 | less**

And we get

"Come 1

"Defects," 1

"I 1

"Information 1

"J" 1

"Plain 2

"Project 5

.

.

.

zest. 1

zigzag 2

zigzagging 1

zigzags, 1

zivio, 1

zmellz 1

zodiac 1

zodiac. 1

zodiacal 2

zoe)\_ 1

zones: 1

zoo. 1

zoological 1

zouave's 1

zrads, 2

zrads. 1

**Program-3: Adding the combiner step to the Wordcount Map Reduce program.**

**Combiner/Reducer Code**

*ReduceClass.java*

|  |  |
| --- | --- |
| 01 | package com.javacodegeeks.examples.wordcount; |
| 02 |  |

|  |  |
| --- | --- |
| 03 | import java.io.IOException; |
| 04 | import java.util.Iterator; |

|  |  |
| --- | --- |
| 05 |  |
| 06 | import org.apache.hadoop.io.IntWritable; |

|  |  |
| --- | --- |
| 07 | import org.apache.hadoop.io.Text; |
| 08 | import org.apache.hadoop.mapreduce.Reducer; |

|  |  |
| --- | --- |
| 09 |  |
| 10 | public class ReduceClass extends Reducer{ |

|  |  |
| --- | --- |
| 11 |  |
| 12 | @Override |

|  |  |
| --- | --- |
| 13 | protected void reduce(Text key, Iterable values, |
| 14 | Context context) |

|  |  |
| --- | --- |
| 15 | throws IOException, InterruptedException { |
| 16 |  |

|  |  |
| --- | --- |
| 17 | int sum = 0; |
| 18 | Iterator valuesIt = values.iterator(); |

|  |  |
| --- | --- |
| 19 |  |
| 20 | //For each key value pair, get the value and adds to the sum |

|  |  |
| --- | --- |
| 21 | //to get the total occurances of a word |
| 22 | while(valuesIt.hasNext()){ |

|  |  |
| --- | --- |
| 23 | sum = sum + valuesIt.next().get(); |
| 24 | } |

|  |  |
| --- | --- |
| 25 |  |
| 26 | //Writes the word and total occurances as key-value pair to the context |

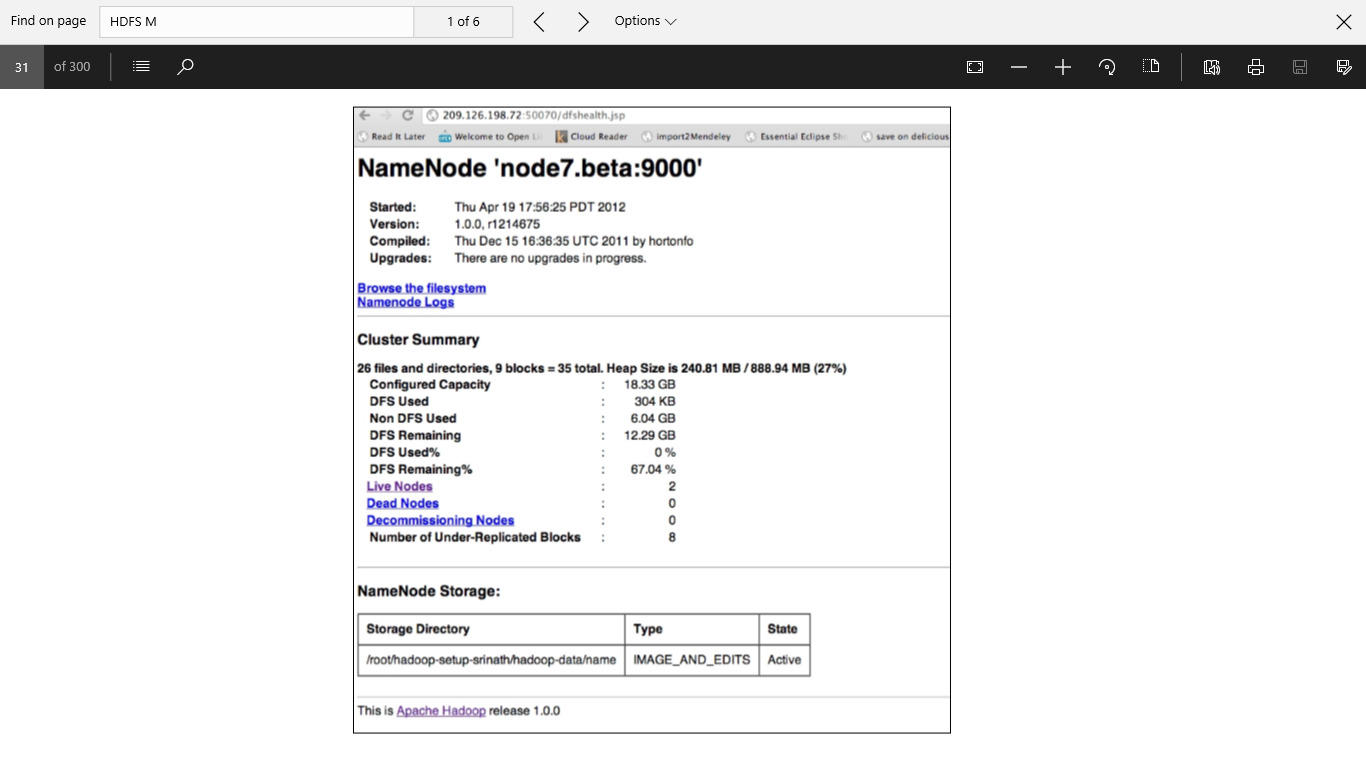
|  |  |
| --- | --- |
| 27 | context.write(key, new IntWritable(sum)); |
| 28 | } |

|  |  |
| --- | --- |
| 29 | } |

**Program-4: Using HDFS monitoring UI**

Let us access the HDFS web console.

1. Access the link http://MASTER\_NODE:50070/ using your browser, and verify that you can see the HDFS startup page. Here, replace MASTER\_NODE with the IP address of the master node running the HDFS NameNode.
2. The following screenshot shows the current status of the HDFS installation including the number of nodes, total storage, storage taken by each node. It also allows users to browse the HDFS filesystem.



**Program-5: HDFS basic command-line file operations.**

**appendToFile**

Usage: hdfs dfs -appendToFile <localsrc> ... <dst>

Append single src, or multiple srcs from local file system to the destination file system. Also reads input from stdin and appends to destination file system.

* hdfs dfs -appendToFile localfile /user/hadoop/hadoopfile
* hdfs dfs -appendToFile localfile1 localfile2 /user/hadoop/hadoopfile
* hdfs dfs -appendToFile localfile hdfs://nn.example.com/hadoop/hadoopfile
* hdfs dfs -appendToFile - hdfs://nn.example.com/hadoop/hadoopfile Reads the input from stdin.

Exit Code:

Returns 0 on success and 1 on error.

**cat**

Usage: hdfs dfs -cat URI [URI ...]

Copies source paths to stdout.

Example:

* hdfs dfs -cat hdfs://nn1.example.com/file1 hdfs://nn2.example.com/file2
* hdfs dfs -cat file:///file3 /user/hadoop/file4

Exit Code:

Returns 0 on success and -1 on error.

**chgrp**

Usage: hdfs dfs -chgrp [-R] GROUP URI [URI ...]

Change group association of files. The user must be the owner of files, or else a super-user. Additional information is in the [Permissions Guide](https://archive.cloudera.com/cdh5/cdh/5/hadoop/hadoop-project-dist/hadoop-hdfs/HdfsPermissionsGuide.html).

Options

* The -R option will make the change recursively through the directory structure.

**chmod**

Usage: hdfs dfs -chmod [-R] <MODE[,MODE]... | OCTALMODE> URI [URI ...]

Change the permissions of files. With -R, make the change recursively through the directory structure. The user must be the owner of the file, or else a super-user. Additional information is in the [Permissions Guide](https://archive.cloudera.com/cdh5/cdh/5/hadoop/hadoop-project-dist/hadoop-hdfs/HdfsPermissionsGuide.html).

Options

* The -R option will make the change recursively through the directory structure.

**chown**

Usage: hdfs dfs -chown [-R] [OWNER][:[GROUP]] URI [URI ]

Change the owner of files. The user must be a super-user. Additional information is in the [Permissions Guide](https://archive.cloudera.com/cdh5/cdh/5/hadoop/hadoop-project-dist/hadoop-hdfs/HdfsPermissionsGuide.html).

Options

* The -R option will make the change recursively through the directory structure.

**copyFromLocal**

Usage: hdfs dfs -copyFromLocal <localsrc> URI

Similar to put command, except that the source is restricted to a local file reference.

Options:

* The -f option will overwrite the destination if it already exists.

**copyToLocal**

Usage: hdfs dfs -copyToLocal [-ignorecrc] [-crc] URI <localdst>

Similar to get command, except that the destination is restricted to a local file reference.

**count**

Usage: hdfs dfs -count [-q] [-h] [-v] [-x] <paths>

Count the number of directories, files and bytes under the paths that match the specified file pattern. The output columns with -count are: DIR\_COUNT, FILE\_COUNT, CONTENT\_SIZE PATHNAME

The output columns with -count -q are: QUOTA, REMAINING\_QUATA, SPACE\_QUOTA, REMAINING\_SPACE\_QUOTA, DIR\_COUNT, FILE\_COUNT, CONTENT\_SIZE, PATHNAME

The -h option shows sizes in human readable format.

The -v option displays a header line.

The -x option excludes snapshots from the result calculation. Without the -x option (default), the result is always calculated from all INodes, including all snapshots under the given path. The -x option is ignored if -u or -q option is given.

Example:

* hdfs dfs -count hdfs://nn1.example.com/file1 hdfs://nn2.example.com/file2
* hdfs dfs -count -q hdfs://nn1.example.com/file1
* hdfs dfs -count -q -h hdfs://nn1.example.com/file1
* hdfs dfs -count -q -h -v hdfs://nn1.example.com/file1

Exit Code:

Returns 0 on success and -1 on error.

**cp**

Usage: hdfs dfs -cp [-f] [-p | -p[topax]] URI [URI ...] <dest>

Copy files from source to destination. This command allows multiple sources as well in which case the destination must be a directory.

'raw.\*' namespace extended attributes are preserved if (1) the source and destination filesystems support them (HDFS only), and (2) all source and destination pathnames are in the /.reserved/raw hierarchy. Determination of whether raw.\* namespace xattrs are preserved is independent of the -p (preserve) flag.

Options:

* The -f option will overwrite the destination if it already exists.
* The -p option will preserve file attributes [topx] (timestamps, ownership, permission, ACL, XAttr). If -p is specified with no *arg*, then preserves timestamps, ownership, permission. If -pa is specified, then preserves permission also because ACL is a super-set of permission. Determination of whether raw namespace extended attributes are preserved is independent of the -p flag.

Example:

* hdfs dfs -cp /user/hadoop/file1 /user/hadoop/file2
* hdfs dfs -cp /user/hadoop/file1 /user/hadoop/file2 /user/hadoop/dir

Exit Code:

Returns 0 on success and -1 on error.

**du**

Usage: hdfs dfs -du [-s] [-h] [-x] URI [URI ...]

Displays sizes of files and directories contained in the given directory or the length of a file in case its just a file.

Options:

* The -s option will result in an aggregate summary of file lengths being displayed, rather than the individual files. Without the -s option, calculation is done by going 1-level deep from the given path.
* The -h option will format file sizes in a "human-readable" fashion (e.g 64.0m instead of 67108864)
* The -x option will exclude snapshots from the result calculation. Without the -x option (default), the result is always calculated from all INodes, including all snapshots under the given path.

Example:

* hdfs dfs -du /user/hadoop/dir1 /user/hadoop/file1 hdfs://nn.example.com/user/hadoop/dir1

Exit Code: Returns 0 on success and -1 on error.

**dus**

Usage: hdfs dfs -dus <args>

Displays a summary of file lengths.

**Note:** This command is deprecated. Instead use hdfs dfs -du -s.

**expunge**

Usage: hdfs dfs -expunge

Empty the Trash. Refer to the [HDFS Architecture Guide](https://archive.cloudera.com/cdh5/cdh/5/hadoop/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html) for more information on the Trash feature.

**find**

Usage: hdfs dfs -find <path> ... <expression> ...

Finds all files that match the specified expression and applies selected actions to them. If no *path* is specified then defaults to the current working directory. If no expression is specified then defaults to -print.

The following primary expressions are recognised:

* -name pattern   
  -iname pattern

Evaluates as true if the basename of the file matches the pattern using standard file system globbing. If -iname is used then the match is case insensitive.

* -print   
  -print0

Always evaluates to true. Causes the current pathname to be written to standard output. If the -print0 expression is used then an ASCII NULL character is appended.

The following operators are recognised:

* expression -a expression   
  expression -and expression   
  expression expression

Logical AND operator for joining two expressions. Returns true if both child expressions return true. Implied by the juxtaposition of two expressions and so does not need to be explicitly specified. The second expression will not be applied if the first fails.

Example:

hdfs dfs -find / -name test -print

Exit Code:

Returns 0 on success and -1 on error.

**get**

Usage: hdfs dfs -get [-ignorecrc] [-crc] <src> <localdst>

Copy files to the local file system. Files that fail the CRC check may be copied with the -ignorecrc option. Files and CRCs may be copied using the -crc option.

Example:

* hdfs dfs -get /user/hadoop/file localfile
* hdfs dfs -get hdfs://nn.example.com/user/hadoop/file localfile

Exit Code:

Returns 0 on success and -1 on error.

**getfacl**

Usage: hdfs dfs -getfacl [-R] <path>

Displays the Access Control Lists (ACLs) of files and directories. If a directory has a default ACL, then getfacl also displays the default ACL.

Options:

* -R: List the ACLs of all files and directories recursively.
* *path*: File or directory to list.

Examples:

* hdfs dfs -getfacl /file
* hdfs dfs -getfacl -R /dir

Exit Code:

Returns 0 on success and non-zero on error.

**getfattr**

Usage: hdfs dfs -getfattr [-R] -n name | -d [-e en] <path>

Displays the extended attribute names and values (if any) for a file or directory.

Options:

* -R: Recursively list the attributes for all files and directories.
* -n name: Dump the named extended attribute value.
* -d: Dump all extended attribute values associated with pathname.
* -e *encoding*: Encode values after retrieving them. Valid encodings are "text", "hex", and "base64". Values encoded as text strings are enclosed in double quotes ("), and values encoded as hexadecimal and base64 are prefixed with 0x and 0s, respectively.
* *path*: The file or directory.

Examples:

* hdfs dfs -getfattr -d /file
* hdfs dfs -getfattr -R -n user.myAttr /dir

Exit Code:

Returns 0 on success and non-zero on error.

**getmerge**

Usage: hdfs dfs -getmerge <src> <localdst> [addnl]

Takes a source directory and a destination file as input and concatenates files in src into the destination local file. Optionally addnl can be set to enable adding a newline character at the end of each file.

**ls**

Usage: hdfs dfs -ls [-R] <args>

Options:

* The -R option will return stat recursively through the directory structure.

For a file returns stat on the file with the following format:

permissions number\_of\_replicas userid groupid filesize modification\_date modification\_time filename

For a directory it returns list of its direct children as in Unix. A directory is listed as:

permissions userid groupid modification\_date modification\_time dirname

Example:

* hdfs dfs -ls /user/hadoop/file1

Exit Code:

Returns 0 on success and -1 on error.

**lsr**

Usage: hdfs dfs -lsr <args>

Recursive version of ls.

**Note:** This command is deprecated. Instead use hdfs dfs -ls -R

**mkdir**

Usage: hdfs dfs -mkdir [-p] <paths>

Takes path uri's as argument and creates directories.

Options:

* The -p option behavior is much like Unix mkdir -p, creating parent directories along the path.

Example:

* hdfs dfs -mkdir /user/hadoop/dir1 /user/hadoop/dir2
* hdfs dfs -mkdir hdfs://nn1.example.com/user/hadoop/dir hdfs://nn2.example.com/user/hadoop/dir

Exit Code:

Returns 0 on success and -1 on error.

**moveFromLocal**

Usage: hdfs dfs -moveFromLocal <localsrc> <dst>

Similar to put command, except that the source localsrc is deleted after it's copied.

**moveToLocal**

Usage: hdfs dfs -moveToLocal [-crc] <src> <dst>

Displays a "Not implemented yet" message.

**mv**

Usage: hdfs dfs -mv URI [URI ...] <dest>

Moves files from source to destination. This command allows multiple sources as well in which case the destination needs to be a directory. Moving files across file systems is not permitted.

Example:

* hdfs dfs -mv /user/hadoop/file1 /user/hadoop/file2
* hdfs dfs -mv hdfs://nn.example.com/file1 hdfs://nn.example.com/file2 hdfs://nn.example.com/file3 hdfs://nn.example.com/dir1

Exit Code:

Returns 0 on success and -1 on error.

**put**

Usage: hdfs dfs -put <localsrc> ... <dst>

Copy single src, or multiple srcs from local file system to the destination file system. Also reads input from stdin and writes to destination file system.

* hdfs dfs -put localfile /user/hadoop/hadoopfile
* hdfs dfs -put localfile1 localfile2 /user/hadoop/hadoopdir
* hdfs dfs -put localfile hdfs://nn.example.com/hadoop/hadoopfile
* hdfs dfs -put - hdfs://nn.example.com/hadoop/hadoopfile Reads the input from stdin.

Exit Code:

Returns 0 on success and -1 on error.

**rm**

Usage: hdfs dfs -rm [-f] [-r|-R] [-skipTrash] URI [URI ...]

Delete files specified as args.

Options:

* The -f option will not display a diagnostic message or modify the exit status to reflect an error if the file does not exist.
* The -R option deletes the directory and any content under it recursively.
* The -r option is equivalent to -R.
* The -skipTrash option will bypass trash, if enabled, and delete the specified file(s) immediately. This can be useful when it is necessary to delete files from an over-quota directory.

Example:

* hdfs dfs -rm hdfs://nn.example.com/file /user/hadoop/emptydir

Exit Code:

Returns 0 on success and -1 on error.

**rmr**

Usage: hdfs dfs -rmr [-skipTrash] URI [URI ...]

Recursive version of delete.

**Note:** This command is deprecated. Instead use hdfs dfs -rm -r

**setfacl**

Usage: hdfs dfs -setfacl [-R] [-b|-k -m|-x <acl\_spec> <path>]|[--set <acl\_spec> <path>]

Sets Access Control Lists (ACLs) of files and directories.

Options:

* -b: Remove all but the base ACL entries. The entries for user, group and others are retained for compatibility with permission bits.
* -k: Remove the default ACL.
* -R: Apply operations to all files and directories recursively.
* -m: Modify ACL. New entries are added to the ACL, and existing entries are retained.
* -x: Remove specified ACL entries. Other ACL entries are retained.
* --set: Fully replace the ACL, discarding all existing entries. The *acl\_spec* must include entries for user, group, and others for compatibility with permission bits.
* *acl\_spec*: Comma separated list of ACL entries.
* *path*: File or directory to modify.

Examples:

* hdfs dfs -setfacl -m user:hadoop:rw- /file
* hdfs dfs -setfacl -x user:hadoop /file
* hdfs dfs -setfacl -b /file
* hdfs dfs -setfacl -k /dir
* hdfs dfs -setfacl --set user::rw-,user:hadoop:rw-,group::r--,other::r-- /file
* hdfs dfs -setfacl -R -m user:hadoop:r-x /dir
* hdfs dfs -setfacl -m default:user:hadoop:r-x /dir

Exit Code:

Returns 0 on success and non-zero on error.

**setfattr**

Usage: hdfs dfs -setfattr -n name [-v value] | -x name <path>

Sets an extended attribute name and value for a file or directory.

Options:

* -b: Remove all but the base ACL entries. The entries for user, group and others are retained for compatibility with permission bits.
* -n name: The extended attribute name.
* -v value: The extended attribute value. There are three different encoding methods for the value. If the argument is enclosed in double quotes, then the value is the string inside the quotes. If the argument is prefixed with 0x or 0X, then it is taken as a hexadecimal number. If the argument begins with 0s or 0S, then it is taken as a base64 encoding.
* -x name: Remove the extended attribute.
* *path*: The file or directory.

Examples:

* hdfs dfs -setfattr -n user.myAttr -v myValue /file
* hdfs dfs -setfattr -n user.noValue /file
* hdfs dfs -setfattr -x user.myAttr /file

Exit Code:

Returns 0 on success and non-zero on error.

**setrep**

Usage: hdfs dfs -setrep [-R] [-w] <numReplicas> <path>

Changes the replication factor of a file. If *path* is a directory then the command recursively changes the replication factor of all files under the directory tree rooted at *path*.

Options:

* The -w flag requests that the command wait for the replication to complete. This can potentially take a very long time.
* The -R flag is accepted for backwards compatibility. It has no effect.

Example:

* hdfs dfs -setrep -w 3 /user/hadoop/dir1

Exit Code:

Returns 0 on success and -1 on error.

**stat**

Usage: hdfs dfs -stat URI [URI ...]

Returns the stat information on the path.

Example:

* hdfs dfs -stat path

Exit Code: Returns 0 on success and -1 on error.

**tail**

Usage: hdfs dfs -tail [-f] URI

Displays last kilobyte of the file to stdout.

Options:

* The -f option will output appended data as the file grows, as in Unix.

Example:

* hdfs dfs -tail pathname

Exit Code: Returns 0 on success and -1 on error.

**test**

Usage: hdfs dfs -test -[ezd] URI

Options:

* The -e option will check to see if the file exists, returning 0 if true.
* The -z option will check to see if the file is zero length, returning 0 if true.
* The -d option will check to see if the path is directory, returning 0 if true.

Example:

* hdfs dfs -test -e filename

**text**

Usage: hdfs dfs -text <src>

Takes a source file and outputs the file in text format. The allowed formats are zip and TextRecordInputStream.

**touchz**

Usage: hdfs dfs -touchz URI [URI ...]

Create a file of zero length.

Example:

* hdfs dfs -touchz pathname

Exit Code: Returns 0 on success and -1 on error.

Exercise 1

* 1. Load a text file .
  2. Create a text file using program (synthetic dataset)
  3. Create a folder in the hdfs.
  4. Copy a file into your folder in hdfs.
  5. Copy a file from one folder to other folder within the HDFS.
  6. Copy the files from HDFS to local file system.

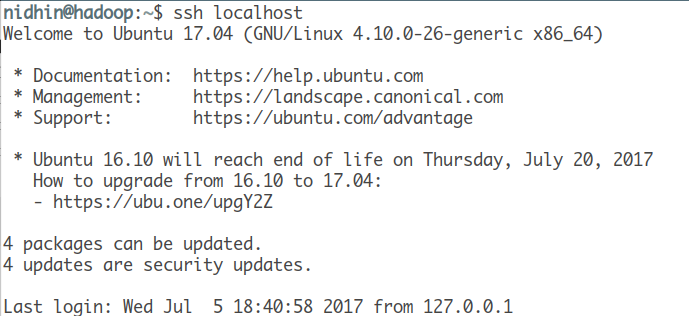
**Program-6: Setting Hadoop in a Distributed Cluster Environment.**

**Pseudo-Distributed mode.**

prerequisite

Now we will setup Hadoop in Pseudo-Distributed setup. This is same as running Hadoop on a real cluster. At least 20% of free memory in hard disk must be free to run Hadoop properly. Also, SSH (Secure Shell) must be installed. We can check it by using

**Step-1:** $ ssh localhost



Master node communicate with slave node very freequently over SSH protocol. in Pseudo-Distributed mode, only one node exist (your machine) and master slave interaction is simulated by JVM. Since communication is very frequent, ssh should be password less.

**Step-2:** Authentication needs to be done using Public key. Above command may not work in your machine if ssh is not installed on your machine, use this command to install ssh

$ sudo apt-get install openssh-server

**Step-3:** to disable password authentication

$ nano /etc/ssh/sshd\_config

**Step-4:** edit the following line to “no” as shown Password Authentication no restart ssh to apply the settings

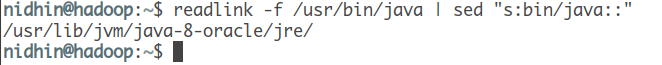
$ /etc/init.d/sshd restart

**Step-5:** Configuration: note that most of the files are in etc/hadoop.

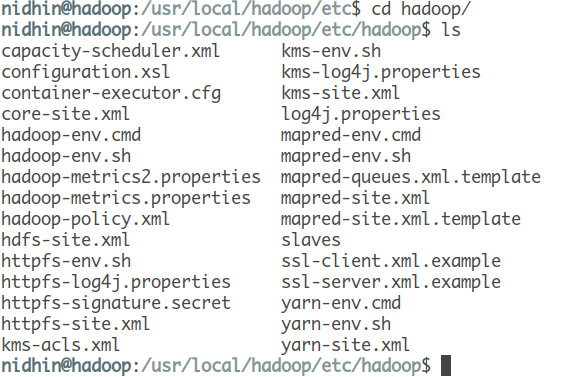
***1. hadoop-env.sh***

Hadoop needs to know where Java is installed in your machine before it can function. To find the java path on your machine:

$ readlink -f /usr/bin/java | sed "s:bin/java::"



We will set the JAVA\_HOME to point to our Java install directory and other environment variable for Hadoop. go to the directory /usr/local/hadoop/etc/hadoop

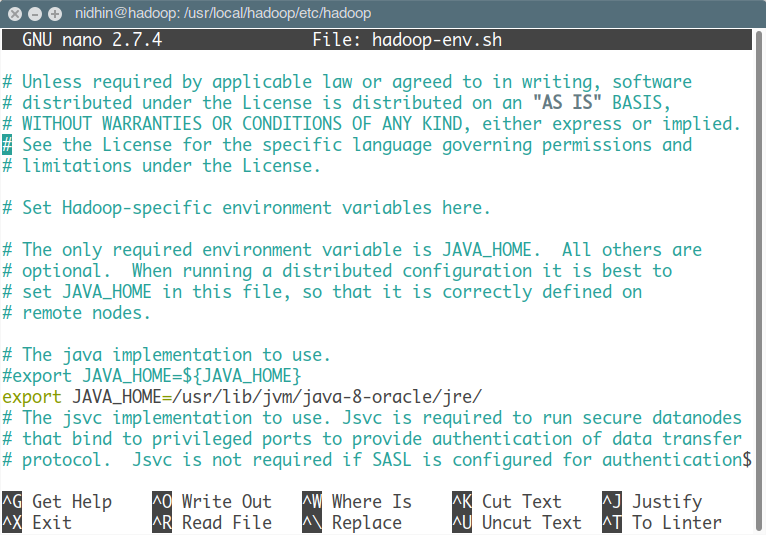


**Step-6:** open hadoop-env.sh using nano editor in terminal

$ nano hadoop-env.sh

Add this environment variable to the file

export JAVA\_HOME=/<the path you found from above command>/



**Step-7:** Also add Hadoop prefix after Java home. This variable is required by Hadoop to startup and run in this mode.

export HADOOP\_PREFIX=/usr/local/hadoop/

now save and exit nano using Ctrl+X and then enter Y to save the buffer with existing name.

***Core-site.xml***

Now we will edit core-site.xml in the same directory. In the terminal

nano core-site.xml

add the following inside the <configuration> </configuration> tag

<property>  
 <name>fs.defaultFS</name>  
 <value>hdfs://localhost:9000</value>  
</property>

***hdfs-site.xml***

Now we will cofigure HDFS using the file hdfs-site.xml. Paste the following inside the <configuration> </configuration> tag

<property>  
 <name>dfs.replication</name>  
 <value>1</value>  
</property>

any data you store on HDFS is replicated to 1 another node as a backup

***mapred-site.xml***

we create a new file in the same directory using the command

nano mapred-site.xml

paste the following lines to mapred-site.xml

<configuration>  
 <property>  
 <name>mapreduce.framework.name</name>  
 <value>yarn</value>  
 </property>  
</configuration>

in this property, we configured the resource negotiator as YARN.

***yarn-site.xml***

At last, we configure the resource negotiator YARN.

nano yarn-site.xml

Paste the following inside the <configuration> </configuration> tag

<property>  
 <name>yarn.nodemanager.aux-services</name>  
 <value>mapreduce\_shuffle</value>  
</property>

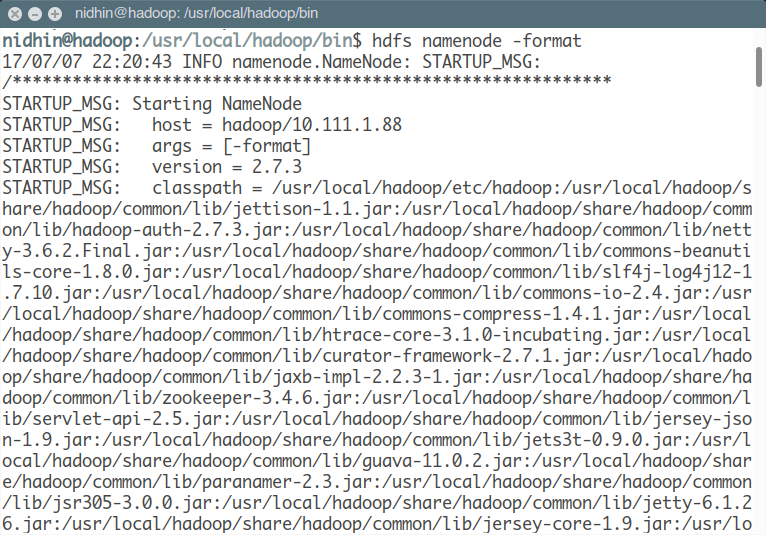
Now we are ready to get started.

***Format NameNode***

Move to bin directory and format the existing namenode using the command. NameNode is the master node in any cluster and keeps track of all the other nodes in the cluster where the processes run.

**Step-13:** We format this to have a fresh start. A NameNode to a cluster is like table of content to a book!

$ hdfs namenode -format

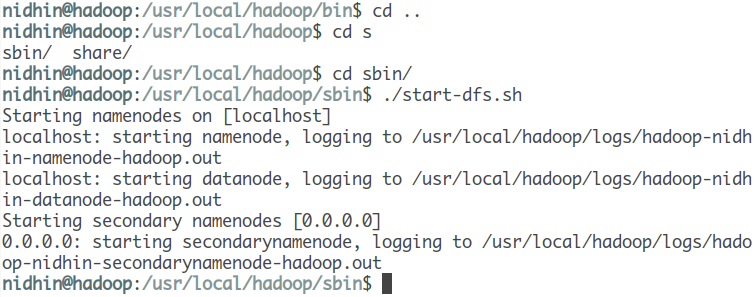


If there are any error in message while executing the command, read through it fix and run the command again.

***Start master and slave node***

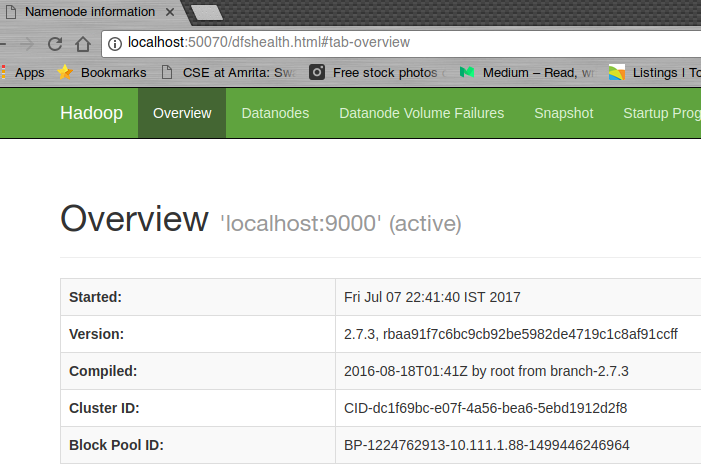
**Step-8:** Once the above step is successful, move to usr/local/hadoop/sbin directory and fire following commands

$ ./start-dfs.sh



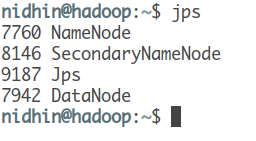
you can check if the NameNode is running on your browser under

*localhost:50070*



**Step-9:** jps command gives the list of running process in java. This is used to check if HDFS is running.

$ jps

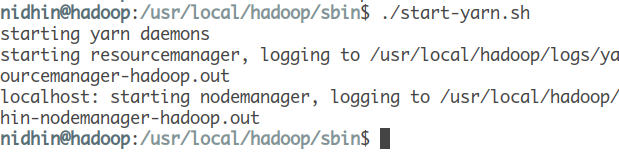


NameNode, SecondaryNameNode and Datanode are the processes associated with HDFS

**Step-10:** Start YARN

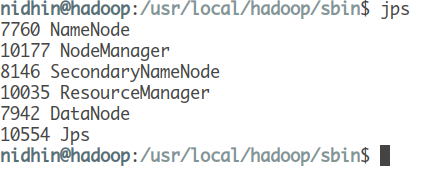
YARN is our resource negotiator here. We can start it by running the command on the same sbin directory

$ start-yarn.sh



you can recheck if YARN is running properly by running jps command again

$ jps



NodeManager and ResourceManager are two new process associated with managing node.

**Program-7: Running the WordCount program in a distributed cluster environment.**

|  |
| --- |
| package org.myorg; |
|  |
| import java.io.\*; |
| import java.util.\*; |
|  |
| import org.apache.hadoop.fs.Path; |
| import org.apache.hadoop.filecache.DistributedCache; |
| import org.apache.hadoop.conf.\*; |
| import org.apache.hadoop.io.\*; |
| import org.apache.hadoop.mapred.\*; |
| import org.apache.hadoop.util.\*; |
|  |
| public class WordCount extends Configured implements Tool { |
|  |
| public static class Map extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable> { |
|  |
| static enum Counters { INPUT\_WORDS } |
|  |
| private final static IntWritable one = new IntWritable(1); |
| private Text word = new Text(); |
|  |
| private boolean caseSensitive = true; |
| private Set<String> patternsToSkip = new HashSet<String>(); |
|  |
| private long numRecords = 0; |
| private String inputFile; |
|  |
| public void configure(JobConf job) { |
| caseSensitive = job.getBoolean("wordcount.case.sensitive", true); |
| inputFile = job.get("map.input.file"); |
|  |
| if (job.getBoolean("wordcount.skip.patterns", false)) { |
| Path[] patternsFiles = new Path[0]; |
| try { |
| patternsFiles = DistributedCache.getLocalCacheFiles(job); |
| } catch (IOException ioe) { |
| System.err.println("Caught exception while getting cached files: " + StringUtils.stringifyException(ioe)); |
| } |
| for (Path patternsFile : patternsFiles) { |
| parseSkipFile(patternsFile); |
| } |
| } |
| } |
|  |
| private void parseSkipFile(Path patternsFile) { |
| try { |
| BufferedReader fis = new BufferedReader(new FileReader(patternsFile.toString())); |
| String pattern = null; |
| while ((pattern = fis.readLine()) != null) { |
| patternsToSkip.add(pattern); |
| } |
| } catch (IOException ioe) { |
| System.err.println("Caught exception while parsing the cached file '" + patternsFile + "' : " + StringUtils.stringifyException(ioe)); |
| } |
| } |
|  |
| public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException { |
| String line = (caseSensitive) ? value.toString() : value.toString().toLowerCase(); |
|  |
| for (String pattern : patternsToSkip) { |
| line = line.replaceAll(pattern, ""); |
| } |
|  |
| StringTokenizer tokenizer = new StringTokenizer(line); |
| while (tokenizer.hasMoreTokens()) { |
| word.set(tokenizer.nextToken()); |
| output.collect(word, one); |
| reporter.incrCounter(Counters.INPUT\_WORDS, 1); |
| } |
|  |
| if ((++numRecords % 100) == 0) { |
| reporter.setStatus("Finished processing " + numRecords + " records " + "from the input file: " + inputFile); |
| } |
| } |
| } |
|  |
| public static class Reduce extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable> { |
| public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException { |
| int sum = 0; |
| while (values.hasNext()) { |
| sum += values.next().get(); |
| } |
| output.collect(key, new IntWritable(sum)); |
| } |
| } |
|  |
| public int run(String[] args) throws Exception { |
| JobConf conf = new JobConf(getConf(), WordCount.class); |
| conf.setJobName("wordcount"); |
|  |
| conf.setOutputKeyClass(Text.class); |
| conf.setOutputValueClass(IntWritable.class); |
|  |
| conf.setMapperClass(Map.class); |
| conf.setCombinerClass(Reduce.class); |
| conf.setReducerClass(Reduce.class); |
|  |
| conf.setInputFormat(TextInputFormat.class); |
| conf.setOutputFormat(TextOutputFormat.class); |
|  |
| List<String> other\_args = new ArrayList<String>(); |
| for (int i=0; i < args.length; ++i) { |
| if ("-skip".equals(args[i])) { |
| DistributedCache.addCacheFile(new Path(args[++i]).toUri(), conf); |
| conf.setBoolean("wordcount.skip.patterns", true); |
| } else { |
| other\_args.add(args[i]); |
| } |
| } |
|  |
| FileInputFormat.setInputPaths(conf, new Path(other\_args.get(0))); |
| FileOutputFormat.setOutputPath(conf, new Path(other\_args.get(1))); |
|  |
| JobClient.runJob(conf); |
| return 0; |
| } |
|  |
| public static void main(String[] args) throws Exception { |
| int res = ToolRunner.run(new Configuration(), new WordCount(), args); |
| System.exit(res); |
| } |
| } |
|  |

**Sample Runs**

Sample text-files as input:

$ bin/hadoop dfs -ls /usr/joe/wordcount/input/   
/usr/joe/wordcount/input/file01   
/usr/joe/wordcount/input/file02   
$ bin/hadoop dfs -cat /usr/joe/wordcount/input/file01   
Hello World, Bye World!   
$ bin/hadoop dfs -cat /usr/joe/wordcount/input/file02   
Hello Hadoop, Goodbye to hadoop.

Run the application:

$ bin/hadoop jar /usr/joe/wordcount.jar org.myorg.WordCount /usr/joe/wordcount/input /usr/joe/wordcount/output

Output:

$ bin/hadoop dfs -cat /usr/joe/wordcount/output/part-00000   
Bye 1   
Goodbye 1   
Hadoop, 1   
Hello 2   
World! 1   
World, 1   
hadoop. 1   
to 1

Notice that the inputs differ from the first version we looked at, and how they affect the outputs.

Now, lets plug-in a pattern-file which lists the word-patterns to be ignored, via the DistributedCache.

$ hadoop dfs -cat /user/joe/wordcount/patterns.txt   
\.   
\,   
\!   
to

Run it again, this time with more options:

$ bin/hadoop jar /usr/joe/wordcount.jar org.myorg.WordCount -Dwordcount.case.sensitive=true /usr/joe/wordcount/input /usr/joe/wordcount/output -skip /user/joe/wordcount/patterns.txt

As expected, the output:

$ bin/hadoop dfs -cat /usr/joe/wordcount/output/part-00000   
Bye 1   
Goodbye 1   
Hadoop 1   
Hello 2   
World 2   
hadoop 1

Run it once more, this time switch-off case-sensitivity:

$ bin/hadoop jar /usr/joe/wordcount.jar org.myorg.WordCount -Dwordcount.case.sensitive=false /usr/joe/wordcount/input /usr/joe/wordcount/output -skip /user/joe/wordcount/patterns.txt

Sure enough, the output:

$ bin/hadoop dfs -cat /usr/joe/wordcount/output/part-00000   
bye 1   
goodbye 1   
hadoop 2   
hello 2   
world 2

**Program-8. Practice on Map Reduce Monitoring User Interface**

**Mapper:**

package com.sreejith.loganalyzer;

import java.io.IOException;

import java.text.ParseException;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.sreejith.loganalyzer.ParseLog;

public class LogMapper extends

     Mapper&lt;LongWritable, Text, IntWritable, IntWritable&gt; {

    private static Logger logger = LoggerFactory.getLogger(LogMapper.class);

    private IntWritable hour = new IntWritable();

    private final static IntWritable one = new IntWritable(1);

    private static Pattern logPattern = Pattern

         .compile("([^ ]\*) ([^ ]\*) ([^ ]\*) \\[([^]]\*)\\]"

                 + " \"([^\"]\*)\""

                 + " ([^ ]\*) ([^ ]\*).\*");

     public void map(LongWritable key, Text value, Context context)

         throws InterruptedException, IOException {

     logger.info("Mapper started");

     String line = ((Text) value).toString();

     Matcher matcher = logPattern.matcher(line);

     if (matcher.matches()) {

         String timestamp = matcher.group(4);

         try {

             hour.set(ParseLog.getHour(timestamp));

         } catch (ParseException e) {

             logger.warn("Exception", e);

         }

         context.write(hour, one);

     }

     logger.info("Mapper Completed");

    }

}

**Reducer:**

package com.sreejith.loganalyzer;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.mapreduce.Reducer;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class LogReducer extends

     Reducer&lt;IntWritable, IntWritable, IntWritable, IntWritable&gt; {

    private static Logger logger = LoggerFactory.getLogger(LogReducer.class);

    public void reduce(IntWritable key, Iterable&lt;IntWritable&gt; values,

         Context context) throws IOException, InterruptedException {

     logger.info("Reducer started");

     int sum = 0;

     for (IntWritable value : values) {

         sum = sum + value.get();

     }

     context.write(key, new IntWritable(sum));

     logger.info("Reducer completed");

    }

}

**Driver:**

package com.sreejith.loganalyzer;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

 public class LogDriver {

    private static Logger logger = LoggerFactory.getLogger(LogDriver.class);

     public static void main(String[] args) throws Exception {

     logger.info("Code started");

     Job job = new Job();

     job.setJarByClass(LogDriver.class);

     job.setJobName("Log Analyzer");

     job.setMapperClass(LogMapper.class);

     job.setReducerClass(LogReducer.class);

     job.setOutputKeyClass(IntWritable.class);

     job.setOutputValueClass(IntWritable.class);

     FileInputFormat.addInputPath(job, new Path(args[0]));

     FileOutputFormat.setOutputPath(job, new Path(args[1]));

     job.waitForCompletion(true);

     logger.info("Code ended");

    }

 }

**Program-9. Sort operation using MapReduce**

import java.nio.ByteBuffer;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.io.WritableComparator;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class SortValues {

public static void main(String[] args) throws Exception {

Path inputPath = new Path(“hdfs://localhost:54310/home/sortinput”);

Path outputDir = new Path(“hdfs://localhost:54310/home/sortoutput”);

// Create configuration

Configuration conf = new Configuration();

// Create job

Job job = new Job(conf, “Sort the Numbers”);

job.setJarByClass(SortValues .class);

// Setup MapReduce

job.setMapperClass(MapTask.class);

job.setReducerClass(ReduceTask.class);

job.setNumReduceTasks(1);

// Specify key / value

job.setMapOutputKeyClass(IntWritable.class);

job.setMapOutputValueClass(IntWritable.class);

job.setOutputKeyClass(IntWritable.class);

job.setOutputValueClass(IntWritable.class);

//job.setSortComparatorClass(IntComparator.class);

// Input

FileInputFormat.addInputPath(job, inputPath);

job.setInputFormatClass(TextInputFormat.class);

// Output

FileOutputFormat.setOutputPath(job, outputDir);

job.setOutputFormatClass(TextOutputFormat.class);

// Execute job

int code = job.waitForCompletion(true) ? 0 : 1;

System.exit(code);

}

public static class **MapTask**extends

Mapper<LongWritable, Text, IntWritable, IntWritable> {

public void map(LongWritable key, Text value, Context context)

throws java.io.IOException, InterruptedException {

String line = value.toString();

String[] tokens = line.split(“,”); // This is the delimiter between

int keypart = Integer.parseInt(tokens[0]);

int valuePart = Integer.parseInt(tokens[1]);

context.write(new IntWritable(valuePart), new IntWritable(keypart));

}

}

public static class **ReduceTask**extends

Reducer<IntWritable, IntWritable, IntWritable, IntWritable> {

public void reduce(IntWritable key, Iterable<IntWritable> list, Context context)

throws java.io.IOException, InterruptedException {

for (IntWritable value : list) {

context.write(value, key);

}

}

}

}

**Program-10: Simple Analytics using Map Reduce.**

The following steps describe how to use MapReduce to calculate simple analytics about the weblog dataset:

* 1. Download the weblog dataset from [ftp://ita.ee.lbl.gov/traces/NASA\_ access\_log\_Jul95.gz](ftp://ita.ee.lbl.gov/traces/NASA_%20access_log_Jul95.gz) and unzip it. We call the extracted folder as DATA\_DIR.
  2. Upload the data to HDFS by running the following commands from HADOOP\_HOME.
  3. If /data is already there, clean it up: >bin/hadoopdfs -mkdir /data > bin/hadoopdfs -mkdir /data/input1 > bin/hadoopdfs -put <DATA\_DIR>/NASA\_access\_log\_Jul95 /data/input1
  4. Unzip the source code of this chapter (chapter6.zip). We will call that folder CHAPTER\_6\_SRC.
  5. Change the hadoop.home property in the CHAPTER\_6\_SRC/build.xmlÀOHWRSRLQW to your Hadoop installation folder.
  6. Compile the source by running the ant build command from the CHAPTER\_6\_SRC folder.
  7. Copy the build/lib/hadoop-cookbook-chapter6.jar to your HADOOP\_HOME.
  8. Run the MapReduce job through the following command from HADOOP\_HOME: >bin/hadoop jar hadoop-cookbook-chapter6.jar chapter6. WebLogMessageSizeAggregator/data/input1 /data/output1
  9. Read the results by running the following command: $bin/hadoopdfs -cat /data/output1/\* You will see that it will print the results as following:

Mean 1150

Max 6823936

Min 0

private final static IntWritable one = new IntWritable(1);

public void map(Object key, Text value, Context context) throws IOException, InterruptedException

{

Matcher matcher = httplogPattern.matcher(value. toString());

if (matcher.matches())

{ int size = Integer.parseInt(matcher.group(5));

context.write(new Text("msgSize"),one);

}

}

public static class AReducer extends Reducer<Text, IntWritable, Text, IntWritable>

{

public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException,InterruptedException

{

double tot = 0;

int count = 0;

Chapter 6

139

int min = Integer.MAX\_VALUE;

int max = 0;

Iterator<IntWritable> iterator = values.iterator();

while (iterator.hasNext())

{

int value = iterator.next().get();

tot = tot + value;

count++;

if (value < min)

{

min = value;

}

if (value > max)

{

max = value;

}

}

context.write(new Text("Mean"), new IntWritable((int) tot / count)); context.write(new Text("Max"), new IntWritable(max));

context.write(new Text("Min"), new IntWritable(min));

}

}

**Program-11: Creation of Database using hive.**

import java.sql.SQLException;

import java.sql.Connection;

import java.sql.ResultSet;

import java.sql.Statement;

import java.sql.DriverManager;

public class HiveCreateDb {

private static String driverName = "org.apache.hadoop.hive.jdbc.HiveDriver";

public static void main(String[] args) throws SQLException {

// Register driver and create driver instance

Class.forName(driverName);

// get connection

Connection con = DriverManager.getConnection("jdbc:hive://localhost:10000/default", "", "");

Statement stmt = con.createStatement();

stmt.executeQuery("CREATE DATABASE userdb");

System.out.println(“Database userdb created successfully.”);

con.close();

}

}

Save the program in a file named HiveCreateDb.java. The following commands are used to compile and execute this program.

$ javac HiveCreateDb.java

$ java HiveCreateDb

**Output:**

Database userdb created successfully.

**Program-12:**

#unzip, efficiently remove the header , and then add to hdfs.

unzip hourly\_TEMP\_2014.zip

tail -n +2 hourly\_TEMP\_2014.csv > hourly\_TEMP\_2014.csv.tmp && mv -f hourly\_TEMP\_2014.csv.tmp hourly\_TEMP\_2014.csv

gzip hourly\_TEMP\_2014.csv

hadoop fs -copyFromLocal hourly\_TEMP\_2014.csv.gz /tmp

#text - load some initial data (via beeline or Hue)

create table temps\_txt (statecode string, countrycode string, sitenum string, paramcode string, poc string, latitude string, longitude string, datum string, param string, datelocal string, timelocal string, dategmt string, timegmt string, degrees double, uom string, mdl string, uncert string, qual string, method string, methodname string, state string, county string, dateoflastchange string) row format delimited fields terminated by ',';

load data inpath '/tmp/hourly\_TEMP\_2014.csv.gz' into table temps\_txt;

Notice how I’m loading a gziped file.  This is a nice feature of the “load data” command.  [Here is some light reading on compression loads](https://cwiki.apache.org/confluence/display/Hive/CompressedStorage).

#orc - create and load some initial data (via beeline or Hue)

create table temps\_orc (statecode string, countrycode string, sitenum string, paramcode string, poc string, latitude string, longitude string, datum string, param string, datelocal string, timelocal string, dategmt string, timegmt string, degrees double, uom string, mdl string, uncert string, qual string, method string, methodname string, state string, county string, dateoflastchange string) stored as orc;

insert into table temps\_orc select \* from temps\_txt;

#avro - create and load some initial data (via beeline or Hue)

create table temps\_avr

stored as avro

tblproperties ('avro.schema.literal'='{

"name": "temps",

"type": "record",

"fields": [

{"name":"statecode", "type":"string"},

{"name":"countrycode", "type":"string"},

{"name":"sitenum", "type":"string"},

{"name":"paramcode", "type":"string"},

{"name":"poc", "type":"string"},

{"name":"latitude", "type":"string"},

{"name":"longitude", "type":"string"},

{"name":"datum", "type":"string"},

{"name":"param", "type":"string"},

{"name":"datelocal", "type":"string"},

{"name":"timelocal", "type":"string"},

{"name":"dategmt", "type":"string"},

{"name":"timegmt", "type":"string"},

{"name":"degrees", "type":"double"},

{"name":"uom", "type":"string"},

{"name":"mdl", "type":"string"},

{"name":"uncert", "type":"string"},

{"name":"qual", "type":"string"},

{"name":"method", "type":"string"},

{"name":"methodname", "type":"string"},

{"name":"state", "type":"string"},

{"name":"county", "type":"string"},

{"name":"dateoflastchange", "type":"string"}

]}');

insert into table temps\_avr select \* from temps\_txt;

#aggregates via hive (beeline or Hue)

select \* from temps\_orc where state == '"California"' and dategmt == '"2014-09-24"' and timegmt == '"08:00"';

select \* from temps\_avr where state == '"California"' and dategmt == '"2014-09-24"' and timegmt == '"08:00"';

#aggregates via impala(impala-shell)

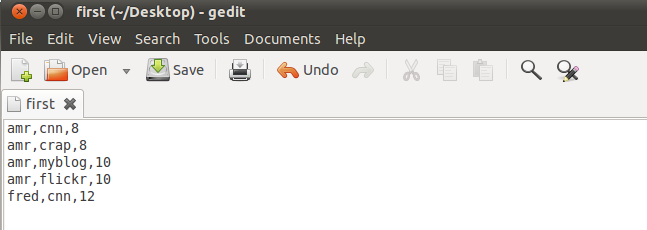
invalidate metadata;

select max(degrees), state, county from temps\_avr group by state, county

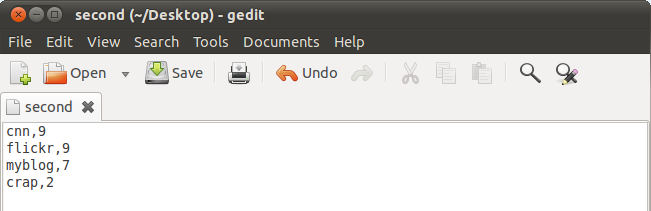
**Program-13. Basic operations in pig**

The Apache Pig Operators is a high-level procedural language for querying large data sets using Hadoop and the Map Reduce Platform. A Pig Latin statement is an operator that takes a relation as input and produces another relation as output. These operators are the main tools for Pig Latin provides to operate on the data. They allow you to transform it by sorting, grouping, joining, projecting, and filtering. Let’s create two files to run the commands:

We have two files with name ‘first’ and ‘second.’ The first file contain three fields: user, url & id.

[](https://d1jnx9ba8s6j9r.cloudfront.net/blog/wp-content/uploads/2014/01/operator3.png)

The second file contain two fields: url & rating. These two files are CSV files.

[](https://d1jnx9ba8s6j9r.cloudfront.net/blog/wp-content/uploads/2014/01/operator4.png)

The Apache Pig operators can be classified as: *Relational and Diagnostic.*

### Relational Operators:

Relational operators are the main tools Pig Latin provides to operate on the data. It allows you to transform the data by sorting, grouping, joining, projecting and filtering. This section covers the basic relational operators.

#### LOAD:

LOAD operator is used to load data from the file system or HDFS storage into a Pig relation.

*In this example,* the Load operator loads data from file ‘first’ to form relation ‘loading1’. The field names are user, url, id.

[Operators in Apache Pig - 5](https://d1jnx9ba8s6j9r.cloudfront.net/blog/wp-content/uploads/2014/01/operator5.png)

[Operators in Apache Pig - 6](https://d1jnx9ba8s6j9r.cloudfront.net/blog/wp-content/uploads/2014/01/operator6.png)

#### FOREACH:

This operator generates data transformations based on columns of data. It is used to add or remove fields from a relation. Use FOREACH-GENERATE operation to work with columns of data.

[Operators in Apache Pig - 7](https://d1jnx9ba8s6j9r.cloudfront.net/blog/wp-content/uploads/2014/01/operator7.png)

#### FOREACH Result:

[](https://d1jnx9ba8s6j9r.cloudfront.net/blog/wp-content/uploads/2014/01/operator8.png)

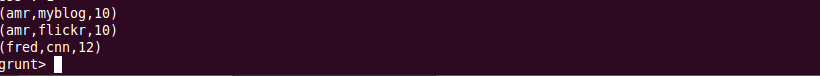
#### FILTER:

This operator selects tuples from a relation based on a condition.

*In this example,* we are filtering the record from ‘loading1’ when the condition ‘id’ is greater than 8.

[Operators in Apache Pig - 9](https://d1jnx9ba8s6j9r.cloudfront.net/blog/wp-content/uploads/2014/01/operator9.png)

#### FILTER Result:

[](https://d1jnx9ba8s6j9r.cloudfront.net/blog/wp-content/uploads/2014/01/operator10.png)

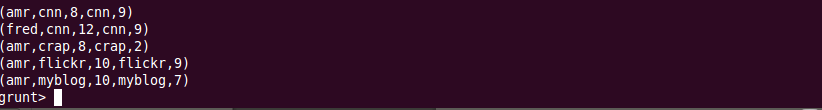
#### JOIN:

JOIN operator is used to perform an inner, equijoin join of two or more relations based on common field values. The JOIN operator always performs an inner join. Inner joins ignore null keys, so it makes sense to filter them out before the join.

*In this example,* join the two relations based on the column ‘url’ from ‘loading1’ and ‘loading2’.

[Operators in Apache Pig - 11](https://d1jnx9ba8s6j9r.cloudfront.net/blog/wp-content/uploads/2014/01/operator11.png)

#### JOIN Result:

[](https://d1jnx9ba8s6j9r.cloudfront.net/blog/wp-content/uploads/2014/01/operator12.png)

#### ORDER BY:

Order By is used to sort a relation based on one or more fields. You can do sorting in ascending or descending order using ASC and DESC keywords.

In below example, we are sorting data in loading2 in ascending order on ratings field.

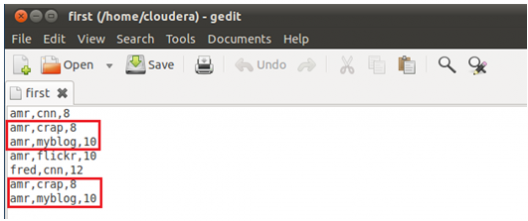
orderby1

#### ORDER BY Result:



#### DISTINCT:

Distinct removes duplicate tuples in a relation.Lets take an input file as below, which has **amr,crap,8** and **amr,myblog,10**twice in the file. When we apply distinct on the data in this file, duplicate entries are removed.



distinct2

#### DISTINCT Result:



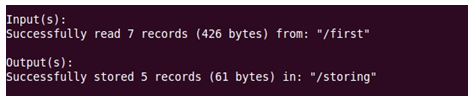
#### STORE:

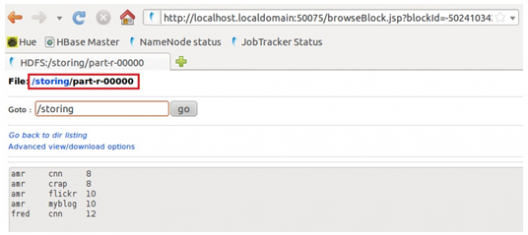
Store is used to save results to the file system.

Here we are saving**loading3** data into a file named **storing** on HDFS.

store1

#### STORE Result:





#### GROUP:

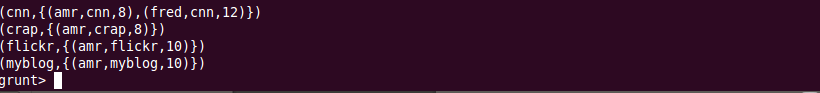
The GROUP operator groups together the tuples with the same group key (key field). The key field will be a tuple if the group key has more than one field, otherwise it will be the same type as that of the group key. The result of a GROUP operation is a relation that includes one tuple per group.

*In this example,* group th

Operators in Apache Pig - 13

e relation ‘loading1’ by column url.

#### GROUP Result:

[](https://d1jnx9ba8s6j9r.cloudfront.net/blog/wp-content/uploads/2014/01/operator14.png)

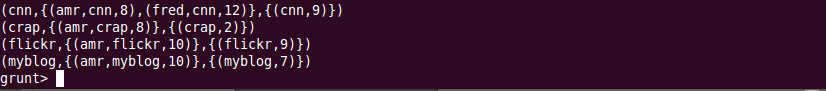
#### COGROUP:

COGROUP is same as GROUP operator. For readability, programmers usually use GROUP when only one relation is involved and COGROUP when multiple relations re involved.

In this example group the ‘loading1’ and ‘loading2’ by url field in both relations.

[Operators in Apache Pig - 15](https://d1jnx9ba8s6j9r.cloudfront.net/blog/wp-content/uploads/2014/01/operator15.png)

#### COGROUP Result:

[](https://d1jnx9ba8s6j9r.cloudfront.net/blog/wp-content/uploads/2014/01/operator16.png)

#### CROSS:

The CROSS operator is used to compute the cross product (Cartesian product) of two or more relations.

Applying cross product on loading1 and loading2.

[Operators in Apache Pig - 17](https://d1jnx9ba8s6j9r.cloudfront.net/blog/wp-content/uploads/2014/01/operator17.png)

#### CROSS Result:

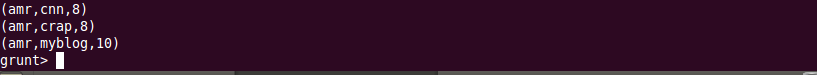
[](https://d1jnx9ba8s6j9r.cloudfront.net/blog/wp-content/uploads/2014/01/operator18.png)

#### LIMIT:

LIMIT operator is used to limit the number of output tuples. If the specified number of output tuples is equal to or exceeds the number of tuples in the relation, the output will include all tuples in the relation.

[Operators in Apache Pig - 19](https://d1jnx9ba8s6j9r.cloudfront.net/blog/wp-content/uploads/2014/01/operator19.png)

#### LIMIT Result:

[](https://d1jnx9ba8s6j9r.cloudfront.net/blog/wp-content/uploads/2014/01/operator20.png)

#### SPLIT:

SPLIT operator is used to partition the contents of a relation into two or more relations based on some expression. Depending on the conditions stated in the expression.

Split the loading2 into two relations x and y. x relation created by loading2 contain the fields that the rating is greater than 8 and y relation contain fields that rating is less than or equal to 8.

[Operators in Apache Pig - 21](https://d1jnx9ba8s6j9r.cloudfront.net/blog/wp-content/uploads/2014/01/operator21.png)

[Operators in Apache Pig - 22](https://d1jnx9ba8s6j9r.cloudfront.net/blog/wp-content/uploads/2014/01/operator22.png)

[Operators in Apache Pig - 23](https://d1jnx9ba8s6j9r.cloudfront.net/blog/wp-content/uploads/2014/01/operator23.png)

**Program-14: Implementation of Word Count using Pig.**

lines = LOAD '/user/hadoop/HDFS\_File.txt' AS (line:chararray);

words = FOREACH lines GENERATE FLATTEN(TOKENIZE(line)) as word;

grouped = GROUP words BY word;

wordcount = FOREACH grouped GENERATE group, COUNT(words);

DUMP wordcount;

**Program-15: Simple programs using Spark.**

**(i). Word Count**

JavaRDD<String> textFile = sc.textFile("hdfs://...");

JavaPairRDD<String, Integer> counts = textFile

.flatMap(s -> Arrays.asList(s.split(" ")).iterator())

.mapToPair(word -> **new** Tuple2<>(word, 1))

.reduceByKey((a, b) -> a + b);

counts.saveAsTextFile("hdfs://...");

**(ii). Pi Estimation**

List<Integer> l = **new** ArrayList<>(NUM\_SAMPLES);

**for** (int i = 0; i < NUM\_SAMPLES; i++) {

l.add(i);

}

long count = sc.parallelize(l).filter(i -> {

double x = Math.random();

double y = Math.random();

**return** x\*x + y\*y < 1;

}).count();

System.out.println("Pi is roughly " + 4.0 \* count / NUM\_SAMPLES);

**Program-16: Implementation of WordCount using Spark**

|  |
| --- |
|  |
| Package org.apache.spark.examples; |
|  |
| import scala.Tuple2; |
|  |
| import org.apache.spark.api.java.JavaPairRDD; |
| import org.apache.spark.api.java.JavaRDD; |
| import org.apache.spark.sql.SparkSession; |
|  |
| import java.util.Arrays; |
| import java.util.List; |
| import java.util.regex.Pattern; |
|  |
| public final class JavaWordCount { |
| private static final Pattern SPACE = Pattern.compile(" "); |
|  |
| public static void main(String[] args) throws Exception { |
|  |
| if (args.length < 1) { |
| System.err.println("Usage: JavaWordCount <file>"); |
| System.exit(1); |
| } |
|  |
| SparkSession spark = SparkSession |
| .builder() |
| .appName("JavaWordCount") |
| .getOrCreate(); |
|  |
| JavaRDD<String> lines = spark.read().textFile(args[0]).javaRDD(); |
|  |
| JavaRDD<String> words = lines.flatMap(s -> Arrays.asList(SPACE.split(s)).iterator()); |
|  |
| JavaPairRDD<String, Integer> ones = words.mapToPair(s -> new Tuple2<>(s, 1)); |
|  |
| JavaPairRDD<String, Integer> counts = ones.reduceByKey((i1, i2) -> i1 + i2); |
|  |
| List<Tuple2<String, Integer>> output = counts.collect(); |
| for (Tuple2<?,?> tuple : output) { |
| System.out.println(tuple.\_1() + ": " + tuple.\_2()); |
| } |
| spark.stop(); |
| } |
| } |

Program – 16: RDD operations transformations and actions.

Filtering using python

lines = sc.textFile("README.md")

>>> pythonLines = lines.filter(**lambda** line: "Python" **in** line)

>>> pythonLines.first()

u'## Interactive Python Shell'

To print error count using python

inputRDD = sc.textFile("log.txt")

errorsRDD = inputRDD.filter(**lambda x: "error" in x)**

errorsRDD = inputRDD.filter(**lambda x: "error" in x)**

warningsRDD = inputRDD.filter(**lambda x: "warning" in x)**

badLinesRDD = errorsRDD.union(warningsRDD

**print** "Input had " + badLinesRDD.count() + " concerning lines"

**print** "Here are 10 examples:"

**for** line **in** badLinesRDD.take(10):

**print** line

Implementation of WordCount

* rdd = sc.textFile("s3://...")
* words = rdd.flatMap(**lambda x: x.split(" "))**
* result = words.map(**lambda x: (x, 1)).reduceByKey(lambda x, y: x + y)**
* **result.collect()**

To get squares of the numbers

nums = sc.parallelize([1, 2, 3, 4])

squared = nums.map(**lambda x: x \* x).collect()**

**for num in squared:**

**print "%i " % (num)**