**HADOOP: Setting up a Single Node Cluster.**

## Purpose

This document describes how to set up and configure a single-node Hadoop installation so that you can quickly perform simple operations using Hadoop MapReduce and the Hadoop Distributed File System (HDFS).

## Prerequisites

**Supported Platform**

* GNU/Linux is supported as a development and production platform. Hadoop has been demonstrated on GNU/Linux clusters with 2000 nodes.
* Windows is also a supported platform but the followings steps are for Linux only. **Required Software**

Required software for Linux include:

1. Java™ must be installed. Recommended Java versions are described at HadoopJavaVersions.
2. ssh must be installed and sshd must be running to use the Hadoop scripts that manage remote Hadoop daemons.

**Installing softwares**

If your cluster doesn’t have the requisite software you will need to install it.For example on Ubuntu Linux:

$ sudo apt-get install ssh

$ sudo apt-get install rsync

## Download: To get a Hadoop distribution, download a recent stable release from one of the [Apache Download Mirrors](http://www.apache.org/dyn/closer.cgi/hadoop/common/).

## **Prepare to Start the Hadoop Cluster:**Unpack the downloaded Hadoop distribution. In the distribution, edit the file etc/hadoop/hadoop-env.sh to define some parameters as follows:

# set to the root of your Java installation

export JAVA\_HOME=/usr/java/latest

Try the following command:

$ bin/hadoop

This will display the usage documentation for the hadoop script.Now you are ready to start your Hadoop cluster in one of the three supported modes:

* [Local (Standalone) Mode](https://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-common/SingleCluster.html#Standalone_Operation)
* [Pseudo-Distributed Mode](https://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-common/SingleCluster.html#Pseudo-Distributed_Operation)
* [Fully-Distributed Mode](https://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-common/SingleCluster.html#Fully-Distributed_Operation)

## **Standalone Operation:**By default, Hadoop is configured to run in a non-distributed mode, as a single Java process. This is useful for debugging.

The following example copies the unpacked conf directory to use as input and then finds and displays every match of the given regular expression. Output is written to the given output directory.

$ mkdir input

$ cp etc/hadoop/\*.xml input

$ bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.2.jar grep input output 'dfs[a-z.]+'

$ cat output/\*

## Pseudo-Distributed Operation

Hadoop can also be run on a single-node in a pseudo-distributed mode where each Hadoop daemon runs in a separate Java process.

**Configuration**

etc/hadoop/core-site.xml:

<configuration>

<property>

<name>fs.defaultFS</name>

<value>hdfs://localhost:9000</value>

</property>

</configuration>

etc/hadoop/hdfs-site.xml:

<configuration>

<property>

<name>dfs.replication</name>

<value>1</value>

</property>

</configuration>

**Setup Passphraseless ssh**

Now check that you can ssh to the localhost without a passphrase:

$ ssh localhost

If you cannot ssh to localhost without a passphrase, execute the following commands:

$ ssh-keygen -t dsa -P '' -f ~/.ssh/id\_dsa

$ cat ~/.ssh/id\_dsa.pub >> ~/.ssh/authorized\_keys

$ chmod 0600 ~/.ssh/authorized\_keys

**Execution**

The following instructions are to run a MapReduce job locally. If you want to execute a job on YARN, see [YARN on Single Node](https://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-common/SingleCluster.html#YARN_on_Single_Node).

1. Format the filesystem:
2. $ bin/hdfs namenode -format
3. Start NameNode daemon and DataNode daemon:
4. $ sbin/start-dfs.sh

The hadoop daemon log output is written to the $HADOOP\_LOG\_DIR directory (defaults to $HADOOP\_HOME/logs).

1. Browse the web interface for the NameNode; by default it is available at:
   * NameNode - http://localhost:50070/
2. Make the HDFS directories required to execute MapReduce jobs:
3. $ bin/hdfs dfs -mkdir /user
4. $ bin/hdfs dfs -mkdir /user/<username>
5. Copy the input files into the distributed filesystem:
6. $ bin/hdfs dfs -put etc/hadoop input
7. Run some of the examples provided:
8. $ bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.2.jar grep input output 'dfs[a-z.]+'
9. Examine the output files: Copy the output files from the distributed filesystem to the local filesystem and examine them:
10. $ bin/hdfs dfs -get output output
11. $ cat output/\*
12. When you’re done, stop the daemons with:
13. $ sbin/stop-dfs.sh

**YARN on a Single Node**

You can run a MapReduce job on YARN in a pseudo-distributed mode by setting a few parameters and running ResourceManager daemon and NodeManager daemon in addition.The following instructions assume that 1. ~ 4. steps of [the above instructions](https://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-common/SingleCluster.html#Execution) are already executed.

1. Configure parameters as follows:etc/hadoop/mapred-site.xml:
2. <configuration>
3. <property>
4. <name>mapreduce.framework.name</name>
5. <value>yarn</value>
6. </property>
7. </configuration>

etc/hadoop/yarn-site.xml:

<configuration>

<property>

<name>yarn.nodemanager.aux-services</name>

<value>mapreduce\_shuffle</value>

</property>

</configuration>

1. Start ResourceManager daemon and NodeManager daemon:
2. $ sbin/start-yarn.sh
3. Browse the web interface for the ResourceManager; by default it is available at:
   * ResourceManager - http://localhost:8088/
4. Run a MapReduce job.
5. When you’re done, stop the daemons with:
6. $ sbin/stop-yarn.sh

.

**BASIC DATA ORGANISATION AND TYPES IN PIG: -**

Basic data structure of the pig system .

Pig has a very limited set of data types. Pig data types are classified into two types. They are:

* Primitive
* Complex

Primitive Data Types: The primitive datatypes are also called as simple datatypes. The simple data types that pig supports are: 

* int : It is signed 32 bit integer. This is similar to the Integer in java.
* long : It is a 64 bit signed integer. This is similar to the Long in java.
* float : It is a 32 bit floating point. This data type is similar to the Float in java.
* double : It is a 63 bit floating pint. This data type is similar to the Double in java.
* chararray : It is character array in unicode UTF-8 format. This corresponds to java's String object.
* bytearray : Used to represent bytes. It is the default data type. If you don't specify a data type for a filed, then bytearray datatype is assigned for the field.
* boolean : to represent true/false values.

Complex Types: Pig supports three complex data types. They are listed below:

* Tuple : An ordered set of fields. Tuple is represented by braces. Example: (1,2)
* Bag : A set of tuples is called a bag. Bag is represented by flower or curly braces. Example: {(1,2),(3,4)}
* Map : A set of key value pairs. Map is represented in a square bracket. Example: [key#value] . The # is used to separate key and value.

Pig allows nesting of complex data structures. Example: You can nest a tuple inside a tuple, bag and a Map   
**Null**: Null is not a datatype. Null is an undefined value or corrupted value. Example: Let say you have declared a field as int type. However, that field contains character values. When reading data from this field, pig converts those character values(corrupted) values into Nulls. Any operation with Null results in Null. The Null in pig is similar to the Null in SQL.

**STEPS IN THE PROJECT**

1. **DATA FILE PREPARATION AND PREVIEW : -**

Pig being a query based system, a basic understanding of the data file is important. Before moving forward let us go through a sample of the data used for the trend analysis.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| PMC | 12-08-2016 | 531.94 | 19.7 | 32212569 | 18.554975 | 73.797235 |
| Pune Cantonment | 01-08-2016 | 13.3 | 0.54 | 505770.9 | 18.5022438 | 73.8785237 |
| PCMC | 07-08-2016 | 239.32 | 9.58 | 2212569 | 18.6245536 | 73.8064108 |

The basic scheme in the above sample and the data file used in this project is:

**(Administrative region), (Date), (Capacity), (Flow Rate), (Population), (Latitude), (Longitude).**

**Note: -**

Pig can work on semi structured, structured and non-structured files in the above case we have a semi structured file. Before loading a semi-structured file into pig {using PigStorage ()} it is important to prepare the file so that it has a rough structure for example in the above case each field is separated by a space. Thus when we are loading the file we can specify that the fields are separated by spaces,in some other files the separators may be “&”, “&&”, “,” etc. these tell pig to separate the values of one field from the next when it encounters them. These symbols are called delimiting symbols.

Sometimes the file may need to be prepared before loading for example the file may have a combination of two delimiting characters then one of the characters may be needed to be replaced by the other in order for the semi structured file to be correctly loaded.

For example: Data before any changes(it has two delimiters , and &)

-Field1, field2 & field3

: After changes

-Field1 & field2 & field3

Now a single delimiter is present in the file. This can be achieved by using the **‘sed’** utility in the Linux system.

1. **LOADING THE FILE INTO THE HDFS: -**

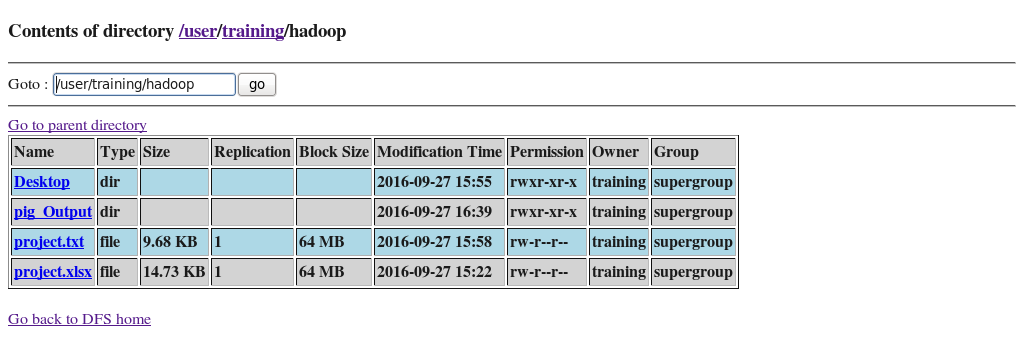
Since hive in its map reduce mode works in the Hadoop file system we need to move the data file into the HDFS partition. This can be achieved by running the put command. For the project we have put the file into a directory named training so first we shall make the desired directory.

**hadoop fs -mkdir /user/training/hadoop**

next we shall load the data file named “project.xlsx” into the directory.

**hadoop fs -put /home/training/Desktop/**project.xlsx **/user/training/hadoop**

This command has two parts first is the path name (data/ distribution\_data.csv) of the file to be put in the Hadoop file system second is the path (/user/training/) of the location where the file is to be loaded.



SNAPSHOT OF HADOOP FILE SYSTEM SHOWING TRAINING DIRECTORY

1. **LOADING THE FILE INTO PIG:**

The below mentioned command will load the file into the pig thus enabling us to do further work on the data.The **load** statement will simply load the data into the specified relation in Apache Pig. In the given e.g. the relation is PigStorage().The **PigStorage()** function loads and stores data as structured text files. It takes a delimiter using which each entity of a tuple is separated as a parameter. By default, it takes **‘\t’** as a parameter.

**Data = LOAD '/user/training/hadoop/**project.txt**' Using PigStorage('\t') AS (Administrative\_region:chararray,date:chararray,Capacity:float,Flowrate:float,Population:int,lat:float,long:float);**

This line loads the information from HDFS into a **PIG collection named Data**. Pig expects data to be tab-delimited by default and as such our file is by default tab (space) delimited. Let’s say our file was semi colon “;” delimited we could have told the system that semicolons are field separators by providing colon as the delimiter.

**NOTE:**

Pig Latin commands are terminated with semicolons. If you press Return on a line without terminating it, you’ll get the >> characters, indicating that the command has been continued.

The **Load** and **Store** functions in Apache Pig are used to determine how the data goes ad comes out of Pig. These functions are used with the load and store operators. Given below is the list of load and store functions available in Pig.

|  |  |
| --- | --- |
| **S.N.** | **Function & Description** |
| 1 | [**PigStorage()**](https://www.tutorialspoint.com/apache_pig/apache_pig_pigstore.htm)  To load and store structured files. |
| 2 | [**TextLoader()**](https://www.tutorialspoint.com/apache_pig/apache_pig_textloader.htm)  To load unstructured data into Pig. |
| 3 | [**BinStorage()**](https://www.tutorialspoint.com/apache_pig/apache_pig_binstorage.htm)  To load and store data into Pig using machine readable format. |
| 4 | [**Handling Compression**](https://www.tutorialspoint.com/apache_pig/apache_pig_handling_compression.htm)  We can load and store compressed data in Apache Pig using the functions BinStorage() and TextLoader() |



SNAPSHOT SHOWING THE BACKEND WORKING OF MAP REDUCE WHILE LOADING DATA INTO PIG

1. **DATA PROCESSING**

Now that the data has been loaded we can perform operations on the data to get the the required data from the data i.e. the data is ready for analysis so that useful conclusions can be drawn out of the data.

Following are the various conclusions that were drawn from the data along with the way in which the steps were carried out.

**A) CATEGORY COUNT: -**

The following script is used to generate a count of all the various categories and the number of videos uploaded in each category over the time period of data in the file.

Note: “--“represents a comment;

**Script: -**

--extract the categories from the data

**group21 = foreach Data generate flatten(TOKENIZE(Administrative\_region)) as t:chararray;**

--group similar types of category tokens together into groups

**g2 =GROUP group21 by t;**

--count the number of entries in each group

**cnt= FOREACH g2 generate group, COUNT(group1);**

--now save the output file to the HDFS

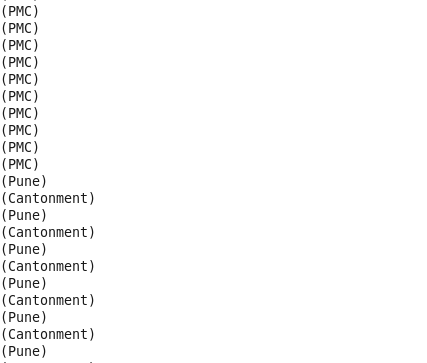
**STORE cnt into 'hdfs://localhost:8020/user/training/category\_count';**

**Command 1:**

The **TOKENIZE()** function of Pig is used to split a string (which contains a group of words) in a single tuple and returns a bag which contains the output of the split operation. In this case it will return bags containing the type of category.

The **FLATTEN**is an operator that changes the structure of tuples and bags. Flatten un-nests tuples as well as bags. The working of flatten is complex and varies from case to case but in this case it converts the bag into tuple. Thus **group1** has tuples of category type corresponding to each entry in the data relation.

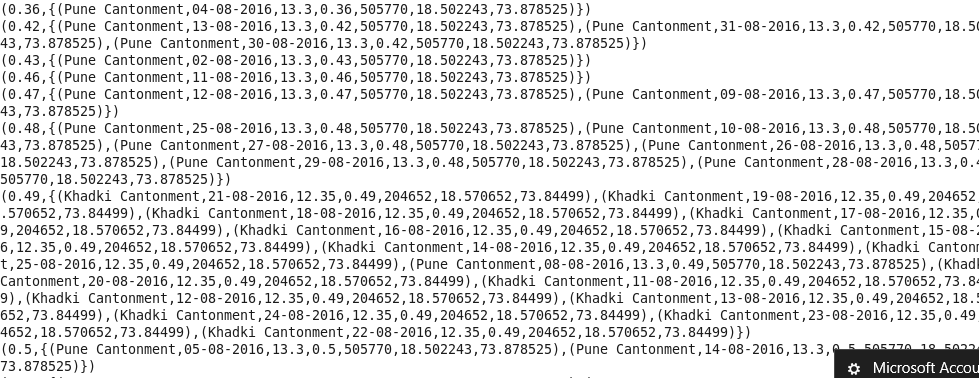
SNAPSHOT OF DATA AFTER TOKENIZE



**Group command**:The GROUP operator groups together tuples that have 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 case each tuple will have a collection of one kind of category. For example, one tuple will have all the entries of the category “Entertainment”.

**COUNT():** This function counts all values ignoring the null values. In the above case it basically counts the number of occurrences of each category in their respective tuple.

****

**B)FINDING THE MINIMUM AND MAXIMUM OF VARIOUS FIELDS**

The following code will find the most viewed video in the data.

--to find the highest rated video

**B = GROUP Data ALL;**

**C = FOREACH B {**

**ord = ORDER Data BY view DESC;**

**top = LIMIT ord 1;**

**GENERATE FLATTEN(top);**

**};**

**STORE C into 'hdfs://localhost:8020/user/training/most\_viewed';**

**Group all**: Use ALL if you want all tuples to go to a single group; for example, when doing aggregates across entire relations.

**Foreach:**The FOREACH operator is used to generate specified data transformations based on the column data.

**Order:** The ORDER BY operator is used to display the contents of a relation in a sorted order based on one or more fields.

**Limit:** The LIMIT operator is used to get a limited number of tuples from a relation. In this case we specify that we need only the top entry.

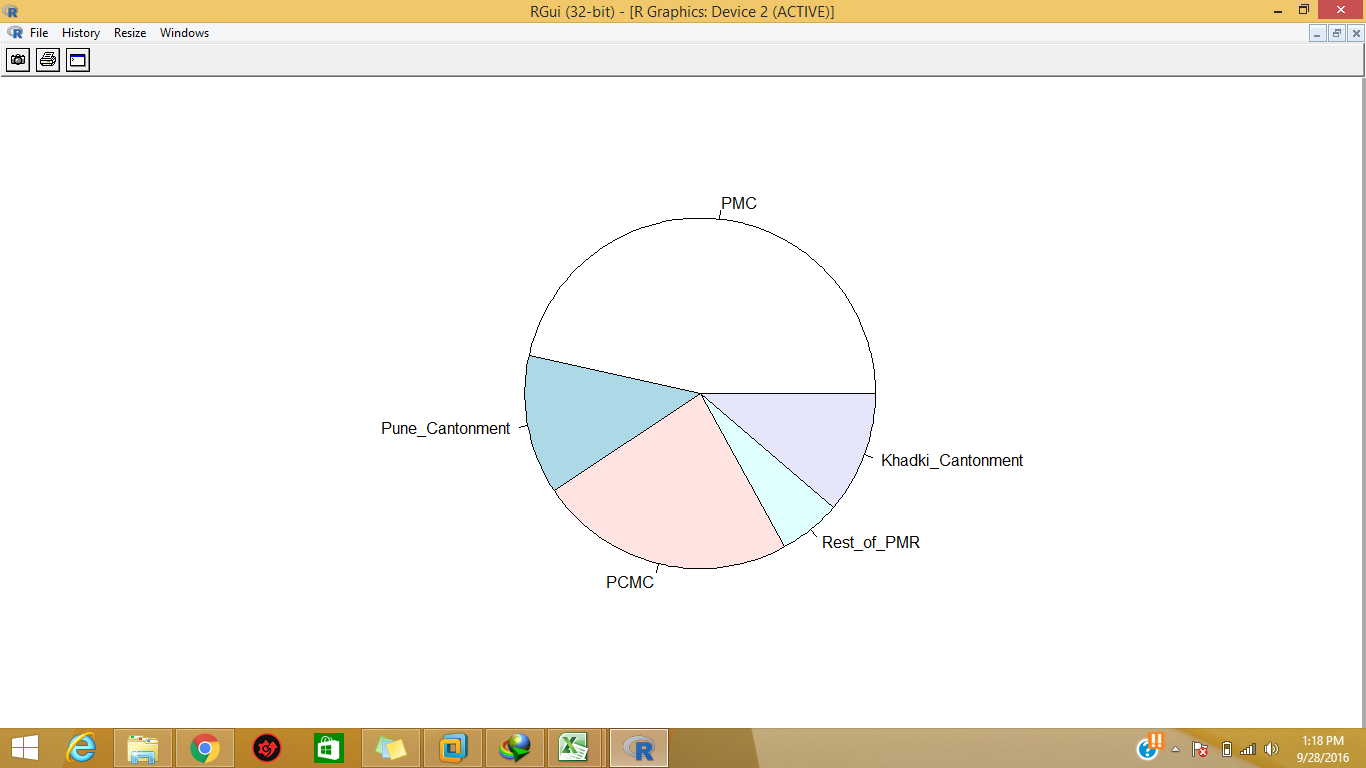
Similarly, we can run similar steps of commands multiple times to find the max and min of various fields like the most comments on a video. This will give us an insight to how many people watch a video and how many actually comment on a video which is the most watched category also these outputs help us to understand the output of other processes.

**C) Finding Average Water Usage**

**data<-read.csv(file.choose(),header=T)**

**attach(data)**

**pie(Flowrate\_avg,Administrative\_region)**

****

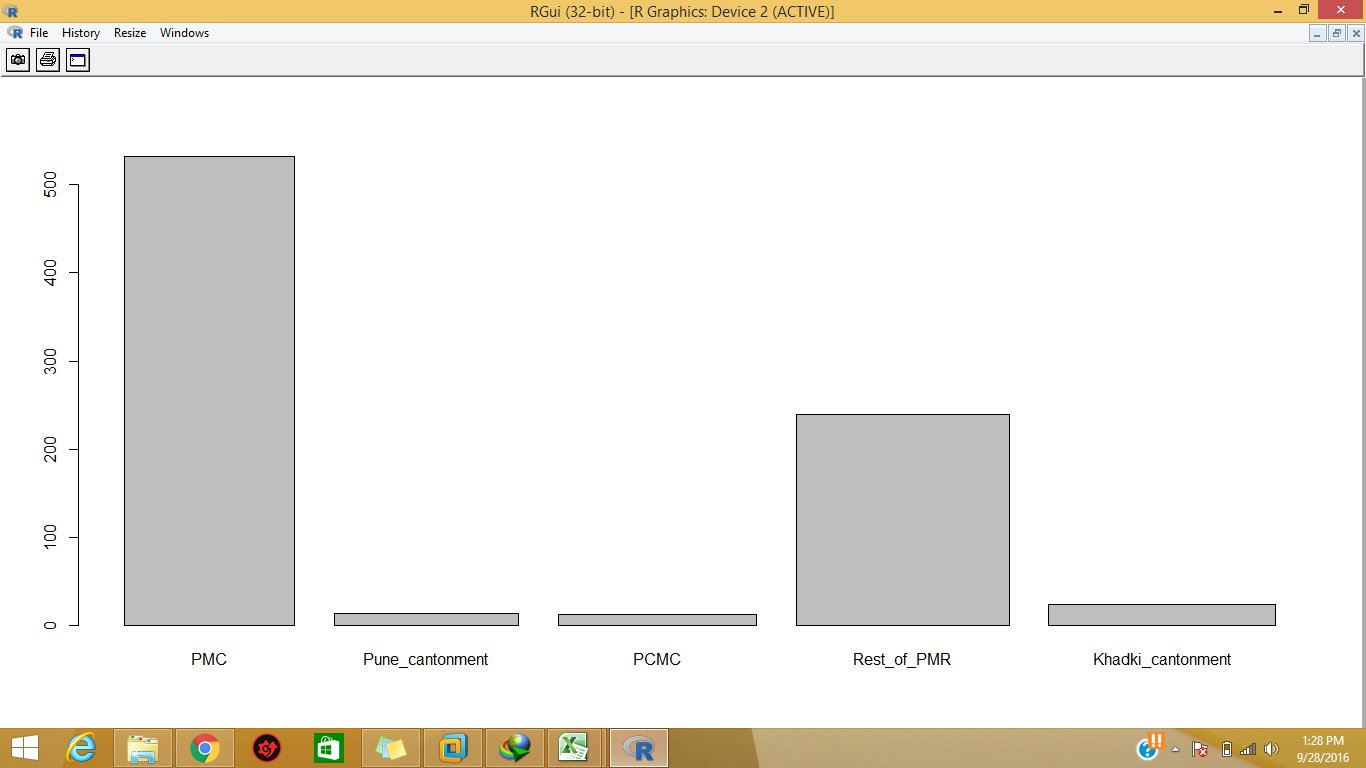
SNAPSHOT OF PIE CHART SHOWING AVERAGE WATER USAGE

**D) Area wise water consumption**

**data2<-read.csv(file.choose(),header=T)**

**attach(data2)**

**barplot(Usage,ad,ylab="(MLD)",names.arg=c("PMC","Pune\_cantonment","PCMC","Rest\_of\_PMR","Khadki\_cantonment"))**

****

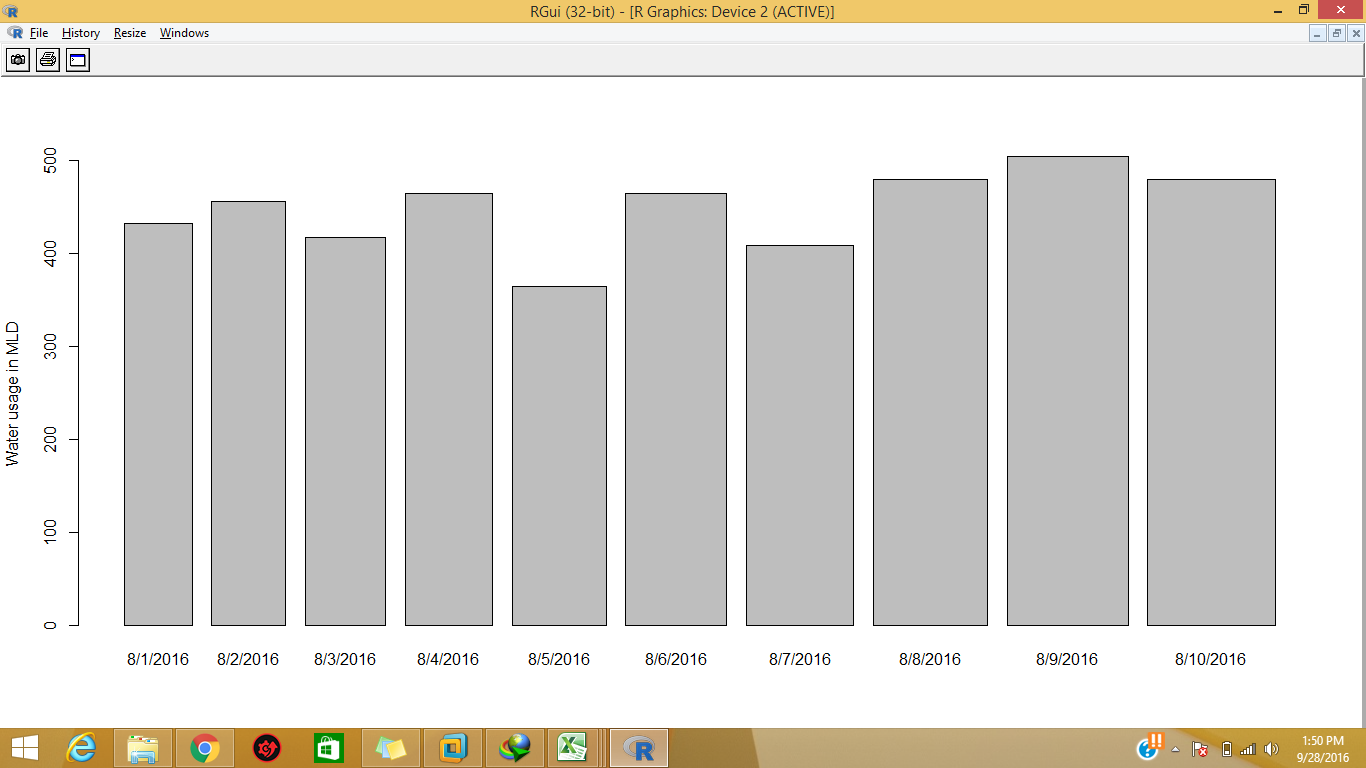
SNAPSHOT OF OUTPUT OF TOTAL NUMBER OF VIDEOS

**F) Day wise Statistics of water uses**

**data3<-read.csv(file.choose(),header=T)**

**attach(data3)**

**barplot(Usage,Date,names.arg=c("8/1/2016","8/2/2016","8/3/2016","8/4/2016","8/5/2016","8/6/2016","8/7/2016","8/8/2016","8/9/2016","8/10/2016"))**



**E) TO FIND THE AVERAGE OF DIFFERENT FIELDS**

Pig has some predefined utilities like the AVG() command that can find the average of a given set of data quantities. This reduces the load on the programmer and on provides a quick and efficient alternative.

The following script will find the average views a video receives in general.

**Script: -**

**Data = LOAD '/user/training/hadoop/project.txt' Using PigStorage('\t') AS (Administrative\_region:chararray,date:chararray,Capacity:float,Flowrate:float,Population:int,lat:float,long:float);**

**group1 = Group Data All;**

**Flowrateavg = Foreach group1 generate (group1.Administrative\_region, group1.flowrate), AVG(group1.flowrate);**

**Group all**: Use ALL if you want all tuples to go to a single group; for example, when doing aggregates across entire relations.

**Foreach:**The FOREACH operator is used to generate specified data transformations based on the column data.

**AVG ():** The Pig-Latin AVG() function is used to compute the average of the numerical values within a bag. While calculating the average value, the AVG() function ignores the NULL values**.**Here we ask the average function to calculate the sum of view field in the data relation and the same is represented by “data.view”

* To get the global average value, we need to perform aGroup All operation, and calculate the average value using the AVG() function.
* To get the average value of a group, we need to group it using the Group By operator and proceed with the average function**.**

By running similar set of commands but different parameter we can find the average of different fields though very simple the above outputs are very important as it gives a clear image of how the users consume the content on YouTube. For example, we can tell on an average how many views a video receives, we can calculate the average comments on a video.

**USAGE (imaginary case):**

We can find the ratio of views to comments for a video. This can tell us how many people actually like to comment if this number is considerable new features can be added to the comment feature on the YouTube.

**5 ) STORING THE FILE FROM PIG TO HDFS:**

Once the data has been processed it must be moved to the hadoop file system this is achieved by using the STORE() command. As seen above after each and every time after the data has been processed and a suitable result has been reached the output relation in stored. In each case the script ends with the STORE command/query given below is a brief summary of how the data store utility works.

Stores or saves results to the file system.

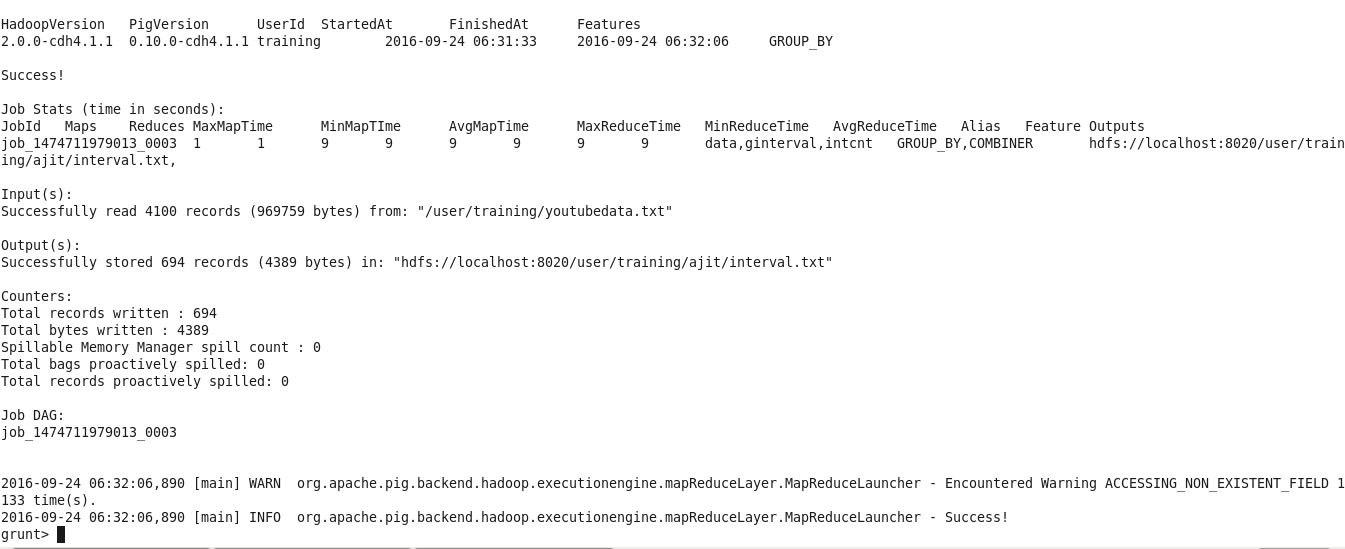
**Syntax: -**

STORE alias INTO 'directory' [USING function];

Terms

1. **Alias**: The name of a relation to be stored.
2. **INTO**: Required keyword.
3. **'directory**': The name of the storage directory, in quotes. If the directory already exists, the STORE operation will fail.The output data files, named part-nnnnn, are written to this directory.
4. **USING:** Keyword. Use this clause to name the store function**.**If the USING clause is omitted, the default store function PigStorage is used.
5. **Function:** The store function. You can use a built in function.
   * **HBaseStorage:**Loads and stores data from an HBase table.
   * **PigStorage:** Loads and stores data as structured text files**.**
   * **PigDump:** Stores data in UTF-8 format.
   * **JsonLoader, JsonStorage:** Load or store JSON data.
   * **BinStorage:** Loads and stores data in machine-readable format.

PigStorage is the default store function and does not need to be specified (simply omit the USING clause). You can write your own store function if your data is in a format that cannot be processed by the built in functions (see User Defined Functions).



SCHEMATIC SHOWING STORING OF FILES FROM PIG INTO HDFS FILE SYSTEM