

CENSUS DATA ANALYSIS

USING HADOOP FRAMEWORK



NIIT

BY: BINDU BHARGAVI GOLLANAPALLI
student id: S170020200571
Mail id- bbhargavi1012@gmail.com
Date-20/02/2017

Table of Contents:

1.ABSTRACT.....	1
2. BIG DATA:.....	1
3. HADOOP:.....	1
3.1 HISTORY OF HADOOP:.....	1
3.2) MODULES IN HADOOP:.....	2
4. HDFS (HADOOP DISTRIBUTED FILE SYSTEM):	2
4.1) Features of HDFS	2
4.2) HDFS ARCHITECTURE:	3
Namenode(MASTER):	3
Datanode(SLAVE):	3
Block:.....	3
5. PROJECT IMPLEMENTATION:.....	4
5.1) Assumptions:	4
5.2) Prerequisites:	4
6. HIVE:.....	4
6.1) Steps for Conversion	4
6.2) TASK 1:	5
Task 1.a) Average income of persons whose age>25 and are not citizens of United states.....	5
Task 1.b) Total number of senior citizens(age>60) and are citizens of United States.....	6
Task 1.c) Married to divorce ratio:.....	7
7. PIG:.....	8
7.1) TASK 2:	8
Task 2.a) Total Number of children whose parents are not in universe.....	9
Task 2.b) Total number of people who are working but not filing tax.	9
Task 2.c) male and female ratio:.....	10
8. MAPREDUCE:	12
8.1) What Mappers does?.....	12
8.2) What Reducer does?.....	12
8.3) TASK 3	13
Task 3.a) Total number of people born in United-States:	13
Task 3.b) total number of females widowed working	15
task 3.c) Average age of divorced people	19
9. CONCLUSIONS:.....	20
10. REFERENCES:	21
11. ACKNOWLEDGEMENT:.....	21

1.ABSTRACT

The project is focussed on analysis of sample census data. Using Hadoop, an open source framework for distributed computing, the raw data of the census is processed and analysed.

2. BIG DATA:

Due to the advent of new technologies, devices, and communication means like social networking sites, the amount of data produced by mankind is growing rapidly every year. Big Data it is a collection of large datasets that cannot be processed using traditional computing techniques. Some of the fields that come under the Big Data.

- Black Box Data
- Social Media Data
- Stock Exchange Data
- Transport Data
- Search Engine Data

Thus, Big Data includes 5Vs:

- 1) **Volume:** refers to the vast amount of data generated every second.
- 2) **Velocity:** refers to the speed at which new data is generated and the speed at which data moves around.
- 3) **Variety:** refers to the different types of data we can now use.
- 4) **Veracity:** refers to the messiness or trustworthiness of the data.
- 5) **Value:** refers to our ability to turn our data into value.

The data in it will be of three types:

Structured data: Relational data.

Semi Structured data: XML data.

Unstructured data: Word, PDF, Text, Media Logs.

BIG DATA can be analysed for insights that lead to better decisions and strategic business moves.

2.1) IMPORTANCE OF BIG DATA:

The importance of BIG DATA doesn't only mean how much data we have, but what we do with it. We can take data from any source and analyse it to find answers that enable

- 1) Cost reductions

- 2) Time reduction
- 3) new product development and optimized offerings
- 4) smart decision making.

When we combine BIG DATA with high-powered analytics, we can accomplish business-related tasks such as:

- Determining root causes of failures, issues and defects in near-real time.
- Generating coupons at the point of sale, based on the customer's buying habits.
- Recalculating entire risk portfolios in minutes.
- Detecting fraudulent behaviour before it affects the organization.

3. HADOOP:

Hadoop is an Apache open source framework written in java that allows distributed processing of large datasets across clusters of computers using simple programming models. A Hadoop frame-worked application works in an environment that provides distributed storage and computation across clusters of computers. Hadoop is designed to scale up from single server to thousands of machines, each offering local computation and storage.

3.1 HISTORY OF HADOOP:

The genesis of Hadoop came from the Google File System paper that was published in October 2003. This paper spawned another research paper from Google – MapReduce: Simplified Data Processing on Large Clusters. In 2006, Doug Cutting Joined Yahoo and took with him the Nutch project as well as ideas based on Google's early work with automating distributed data storage and processing. The Nutch project was divided – the web crawler portion remained as Nutch and the distributed computing and processing portion became Hadoop (named after Cutting's son's toy elephant). In 2008, Yahoo released Hadoop as an open-source project. Today, Hadoop's framework and ecosystem of technologies are managed and maintained by the non-profit Apache Software Foundation (ASF), a global community of software developers and contributors.

3.2) MODULES IN HADOOP:

The project uses these modules:

- **Hadoop Common:** The java libraries and utilities that support the other Hadoop modules.
- **Hadoop Distributed File System (HDFS):** A distributed file system that provides high-throughput access to application data.
- **Hadoop YARN:** A framework for job scheduling and cluster resource management.
- **Hadoop MapReduce:** A YARN-based system for parallel processing of large data sets.

Other Hadoop ecosystem products at Apache which are used for the analysis:

Hive: A data warehouse infrastructure that provides data summarization and ad hoc querying.

Pig: A high-level data-flow language and execution framework for parallel computation.

4. HDFS (HADOOP DISTRIBUTED FILE SYSTEM):

Hadoop File System was developed using distributed file system design. It runs on commodity hardware. Unlike other distributed systems, HDFS is highly fault tolerant and designed using low-cost hardware.

HDFS holds very large amount of data and provides easier access. To store such huge data, the files are stored across multiple machines. These files are stored in redundant fashion to rescue the system from possible data losses in case of failure. HDFS also makes applications available to parallel processing.

4.1) Features of HDFS

- It is suitable for the distributed storage and processing.
- Hadoop provides a command interface to interact with HDFS.
- The built-in servers of namenode and datanode help users to easily check the status of cluster.
- Streaming access to file system data.
- HDFS provides file permissions and authentication.

4.2) HDFS ARCHITECTURE:

HDFS follows the master-slave architecture and it has the following elements.

Namenode(MASTER):

NameNode is at the heart of the HDFS file system which manages the metadata i.e. the data of the files is not stored on the NameNode but rather it has the directory tree of all the files present in the HDFS file system on a Hadoop cluster. NameNode uses two files for the namespace-

fsimage file- It keeps track of the latest checkpoint of the namespace.

edits file-It is a log of changes that have been made to the namespace since checkpoint.

Namenode can never be commodity hardware because the entire HDFS rely on it. It is the single point of failure in HDFS. Namenode must be a high-availability machine.

The system having the namenode acts as the master server and it does the following tasks:

- Manages the file system namespace.
- Regulates client's access to files.

- It also executes file system operations such as renaming, closing, and opening files and directories.

Datanode(SLAVE):

It is the slave node which stores the actual data in blocks and retrieve them when they are told to by client or name node. They report back to name node periodically, with list of blocks that they are storing. These nodes manage the data storage of their system.

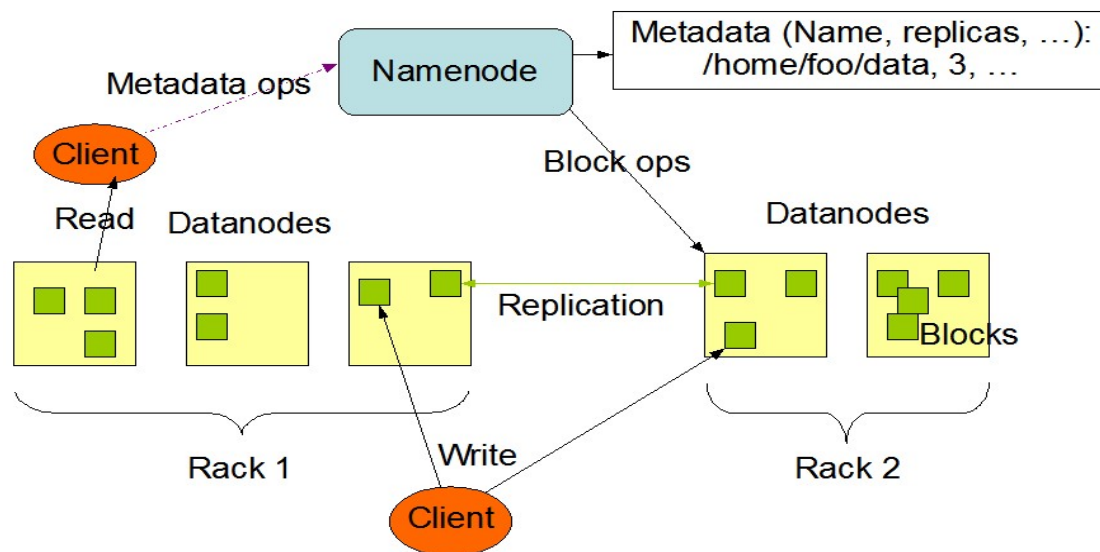
- Datanodes perform read-write operations on the file systems, as per client request.
- They also perform operations such as block creation, deletion, and replication as per the instructions of the namenode.

Block:

The minimum amount of data that can be read or written is generally referred to as a "block" in HDFS.

Generally, the user data is stored in the files of HDFS. The file in a file system will be divided into one or more segments and/or stored in individual data nodes. These file segments are called as blocks. In other words, the minimum amount of data that HDFS can read or write is called a Block. The default block size is 128MB, but it can be increased as per the need to change in HDFS configuration.

HDFS Architecture



5. PROJECT IMPLEMENTATION:

5.1) Assumptions:

- Hadoop Cluster is Running
- Ecosystem Products (Pig, Hive) are installed

- Census data is available on HDFS in JSON Format

5.2) Prerequisites:

The Census data is in JSON format and hence needs to be converted in csv format in Hadoop file system. For that we use HIVE.

6. HIVE:

Apache Hive is a data warehouse infrastructure built on top of Hadoop for providing data summarization, query, and analysis. Hive gives an SQL-like interface to query data stored in various databases and file systems that integrate with Hadoop.

6.1) Steps for Conversion

Step 1: Create Table in hive to read entire record as one json string

```
hive(default)>create database census;
```

```
hive(default)>use census;
```

```
hive (census)> create table census_demo(json string) stored as textfile;
```

```
hive (census)> LOAD DATA LOCAL INPATH '/home/hduser/Downloads/sample.dat' INTO
TABLE census_demo;
```

```
hive(census)>create table censusdata(age INT,education STRING,maritalstatus
STRING,gender STRING,tax STRING,income DOUBLE,parents STRING,country STRING,citizen
STRING,weeks INT)
```

```
> row format delimited
```

```
>fields terminated by ','
```

```
>stored as textfile;
```

Step 2: Create Table to convert and store json string into different fields which will create a file on Hadoop filesystem in csv format. get_json_object is hive built-in function.

```
hive (census)> insert overwrite table censusdata select
```

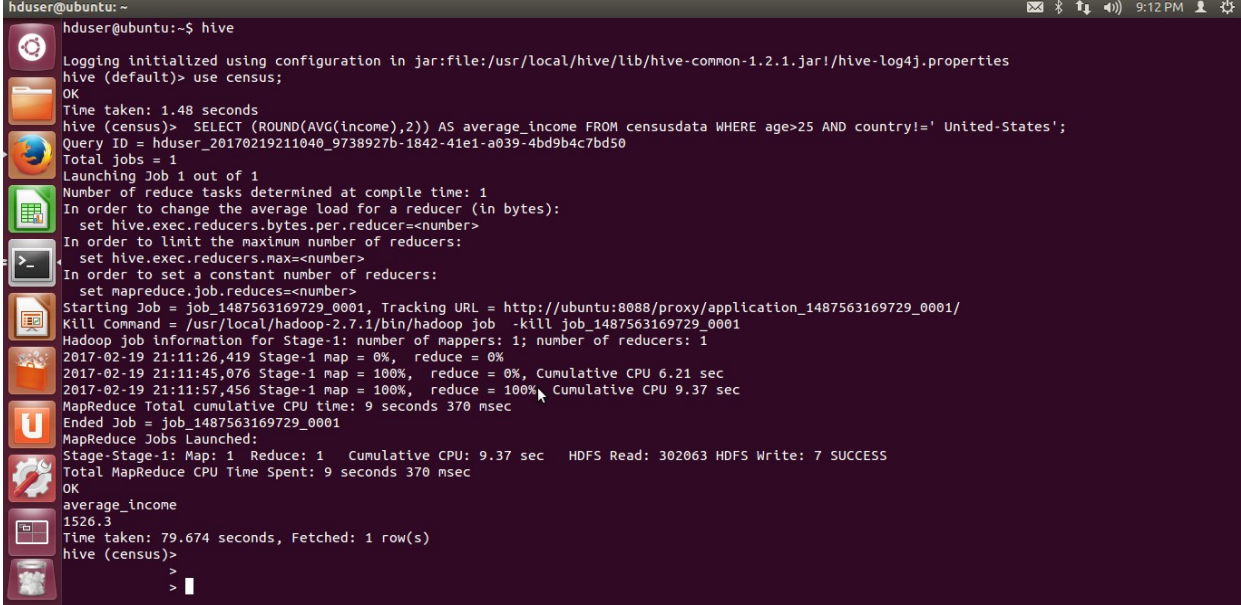
```
get_json_object(json,"$.Age"),get_json_object(json,"$.Education"),get_json_object(json,"$.
MaritalStatus"),get_json_object(json,"$.Gender"),get_json_object(json,"$.TaxFilerStatus"),g
et_json_object(json,"$.Income"),get_json_object(json,"$.Parents"),get_json_object(json,"$.
CountryOfBirth"),get_json_object(json,"$.Citizenship"),get_json_object(json,"$.WeeksWork
ed") from census_demo;
```

6.2) TASK 1:

Task 1.a) Average income of persons whose age>25 and are not citizens of United states.

```
hive (census)> SELECT (ROUND(AVG(income),2)) AS average_income FROM censusdata  
WHERE age>25 AND country!=' United-States';
```

output: 1526.30



```
hduser@ubuntu:~  
hduser@ubuntu:~$ hive  
Logging initialized using configuration in jar:file:/usr/local/hive/lib/hive-common-1.2.1.jar!/hive-log4j.properties  
hive (default)> use census;  
OK  
Time taken: 1.48 seconds  
hive (census)> SELECT (ROUND(AVG(income),2)) AS average_income FROM censusdata WHERE age>25 AND country!=' United-States';  
Query ID = hduser_20170219211040_9738927b-1842-41e1-a039-4bd9b4c7bd50  
Total jobs = 1  
Launching Job 1 out of 1  
Number of reduce tasks determined at compile time: 1  
In order to change the average load for a reducer (in bytes):  
  set hive.exec.reducers.bytes.per.reducer=<number>  
In order to limit the maximum number of reducers:  
  set hive.exec.reducers.max=<number>  
In order to set a constant number of reducers:  
  set mapreduce.job.reduces=<number>  
Starting Job = job_1487563169729_0001, Tracking URL = http://ubuntu:8088/proxy/application_1487563169729_0001/  
Kill Command = /usr/local/hadoop-2.7.1/bin/hadoop job -kill job_1487563169729_0001  
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1  
2017-02-19 21:11:26,419 Stage-1 map = 0%, reduce = 0%  
2017-02-19 21:11:45,076 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 6.21 sec  
2017-02-19 21:11:57,456 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 9.37 sec  
MapReduce Total cumulative CPU time: 9 seconds 370 msec  
Ended Job = job_1487563169729_0001  
MapReduce Jobs Launched:  
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 9.37 sec HDFS Read: 302063 HDFS Write: 7 SUCCESS  
Total MapReduce CPU Time Spent: 9 seconds 370 msec  
OK  
average_income  
1526.3  
Time taken: 79.674 seconds, Fetched: 1 row(s)  
hive (census)>  
>  
>
```

Task 1.b) Total number of senior citizens(age>60) and are citizens of United States.

```
hive (census)> SELECT count(*) from censusdata a,agegroup b where a.age=b.age and  
b.age>60 and a.citizen=' Native- Born in the United States';
```

output: 236


```
hduser@ubuntu: ~  
cable  
Execution log at: /tmp/hduser/hduser_20170222111416_2bc36801-b116-4425-b74f-90bff8a879aa.log  
2017-02-22 11:14:29 Starting to launch local task to process map join; maximum memory = 518979584  
2017-02-22 11:14:31 Dump the side-table for tag: 1 with group count: 40 into file: file:/usr/local/hive/iotmp/50c6dd2a-a6f8-4a9f-838c-59dc37d9c835/hive_2017-02-22_11-14-16_852_2528696834517799294-1/-local-10004/HashTable-Stage-2/MapJoin-mapfile01--.hashtable  
2017-02-22 11:14:31 Uploaded 1 File to: file:/usr/local/hive/iotmp/50c6dd2a-a6f8-4a9f-838c-59dc37d9c835/hive_2017-02-22_11-14-16_852_2528696834517799294-1/-local-10004/HashTable-Stage-2/MapJoin-mapfile01--.hashtable (983 bytes)  
2017-02-22 11:14:31 End of local task; Time Taken: 2.453 sec.  
Execution completed successfully  
MapredLocal task succeeded  
Launching Job 1 out of 1  
Number of reduce tasks determined at compile time: 1  
In order to change the average load for a reducer (in bytes):  
set hive.exec.reducers.bytes.per.reducer=<number>  
In order to limit the maximum number of reducers:  
set hive.exec.reducers.max=<number>  
In order to set a constant number of reducers:  
set mapreduce.job.reduces=<number>  
Starting Job = job_1487778074648_0008, Tracking URL = http://ubuntu:8088/proxy/application_1487778074648_0008/  
Kill Command = /usr/local/hadoop-2.7.1/bin/hadoop job -kill job_1487778074648_0008  
Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1  
2017-02-22 11:14:51,706 Stage-2 map = 0%, reduce = 0%  
2017-02-22 11:15:04,840 Stage-2 map = 100%, reduce = 0%, Cumulative CPU 4.47 sec  
2017-02-22 11:15:15,678 Stage-2 map = 100%, reduce = 100%, Cumulative CPU 6.53 sec  
MapReduce Total cumulative CPU time: 6 seconds 530 msec  
Ended Job = job_1487778074648_0008  
MapReduce Jobs Launched:  
Stage-Stage-2: Map: 1 Reduce: 1 Cumulative CPU: 6.53 sec HDFS Read: 306074 HDFS Write: 4 SUCCESS  
Total MapReduce CPU Time Spent: 6 seconds 530 msec  
OK  
_c0  
236  
Time taken: 60.101 seconds, Fetched: 1 row(s)  
hive (census)>
```

Task 1.c) Married to divorce ratio:

For this we must know all the marital status categories. For that we need to run

```
hive>SELECT maritalstatus,count(*) FROM censusdata GROUP BY marital status;
```

```
hduser@ubuntu: ~  
FAILED: ParseException line 1:54 extraneous input 'maritalstatus' expecting EOF near '<EOF>'  
hive (census)> select maritalstatus,count(*) from censusdata group by maritalstatus;  
Query ID = hduser_20170222105357_79fd4a63-d5fd-4e96-8dbf-5e2fda5b8290  
Total jobs = 1  
Launching Job 1 out of 1  
Number of reduce tasks not specified. Estimated from input data size: 1  
In order to change the average load for a reducer (in bytes):  
  set hive.exec.reducers.bytes.per.reducer=<number>  
In order to limit the maximum number of reducers:  
  set hive.exec.reducers.max=<number>  
In order to set a constant number of reducers:  
  set mapreduce.job.reduces=<number>  
Starting Job = job_1487778074648_0007, Tracking URL = http://ubuntu:8088/proxy/application_1487778074648_0007/  
Kill Command = /usr/local/hadoop-2.7.1/bin/hadoop job -kill job_1487778074648_0007  
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1  
2017-02-22 10:54:14,571 Stage-1 map = 0%, reduce = 0%  
2017-02-22 10:54:35,693 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 7.83 sec  
2017-02-22 10:54:45,713 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 9.82 sec  
MapReduce Total cumulative CPU time: 9 seconds 820 msec  
Ended Job = job_1487778074648_0007  
MapReduce Jobs Launched:  
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 9.82 sec HDFS Read: 300759 HDFS Write: 152 SUCCESS  
Total MapReduce CPU Time Spent: 9 seconds 820 msec  
OK  
maritalstatus _c1  
Divorced 122  
Married-A F spouse present 8  
Married-civilian spouse present 870  
Married-spouse absent 17  
Never married 868  
Separated 26  
Widowed 89  
Time taken: 50.542 seconds, Fetched: 7 row(s)  
hive (census)>
```

hive (census)>select Round(sum(case when maritalstatus like ' Married-A F spouse present'
or maritalstatus like ' Married-civilian spouse present' or maritalstatus like ' Married-spouse
absent' then 1 else 0 end)/count(*),2) as Married_Ratio,
> Round(sum(case when maritalstatus like ' Divorced' then 1 else 0 end)/count(*),2) as
Divorced_Ratio
> from censusdata WHERE age >=18 ;

Output:

Married_Ratio	Divorced_Ratio
0.61	0.08

```
hduser@ubuntu: ~  
_c0  
0  
Time taken: 34.046 seconds, Fetched: 1 row(s)  
hive (census)> select Round(sum(case when maritalstatus like ' Married-A F spouse present' or maritalstatus like ' Married-civilian spouse pr  
esent' or maritalstatus like ' Married-spouse absent' then 1 else 0 end)/count(*),2) as Married_Ratio,  
      > Round(sum(case when maritalstatus like ' Divorced' then 1 else 0 end)/count(*),2) as Divorced_Ratio  
      > from censusdata WHERE age >=18 ;  
Query ID = hduser_20170222104229_205d7fbf-5e31-4091-b1bf-747919583a01  
Total jobs = 1  
Launching Job 1 out of 1  
Number of reduce tasks determined at compile time: 1  
In order to change the average load for a reducer (in bytes):  
  set hive.exec.reducers.bytes.per.reducer=<number>  
In order to limit the maximum number of reducers:  
  set hive.exec.reducers.max=<number>  
In order to set a constant number of reducers:  
  set mapreduce.job.reduces=<number>  
Starting Job = job_1487778074648_0006, Tracking URL = http://ubuntu:8088/proxy/application_1487778074648_0006/  
Kill Command = /usr/local/hadoop-2.7.1/bin/hadoop job -kill job_1487778074648_0006  
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1  
2017-02-22 10:42:45,007 Stage-1 map = 0%, reduce = 0%  
2017-02-22 10:43:30,569 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 19.84 sec  
2017-02-22 10:43:43,202 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 22.82 sec  
MapReduce Total cumulative CPU time: 22 seconds 820 msec  
Ended Job = job_1487778074648_0006  
MapReduce Jobs Launched:  
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 22.82 sec HDFS Read: 305663 HDFS Write: 10 SUCCESS  
Total MapReduce CPU Time Spent: 22 seconds 820 msec  
OK  
married_ratio   divorced_ratio  
0.61    0.08  
Time taken: 77.224 seconds, Fetched: 1 row(s)  
hive (census)>
```

For the next tasks, we will be using PIG.

7. PIG:

Apache Pig is a platform for analysing large data sets that consists of a high-level language for expressing data analysis programs, coupled with infrastructure for evaluating these programs. The salient property of Pig programs is that their structure is amenable to substantial parallelization, which in turns enables them to handle very large data sets. Pig's language layer currently consists of a textual language called Pig Latin, which has the following key properties:

- **Ease of programming.** It is trivial to achieve parallel execution of simple, "embarrassingly parallel" data analysis tasks. Complex tasks comprised of multiple interrelated data transformations are explicitly encoded as data flow sequences, making them easy to write, understand, and maintain.
- **Optimization opportunities.** The way in which tasks are encoded permits the system to optimize their execution automatically, allowing the user to focus on semantics rather than efficiency.
- **Extensibility.** Users can create their own functions to do special-purpose processing.

7.1) TASK 2:

Task 2.a) Total Number of children(age<17) whose parents are not in universe

```
grunt> censusdata = LOAD '/home/hduser/Documents/censusdata/census' USING
PigStorage(',') AS
(age:INT,education:CHARARRAY,maritalstatus:CHARARRAY,gender:CHARARRAY,taxfilestatus
:CHARARRAY,income:DOUBLE,parents:CHARARRAY,country:CHARARRAY,citizen:CHARARRA
Y,weeksworked:INT);
```

```
grunt> filterdata = FILTER censusdata BY age<17;
```

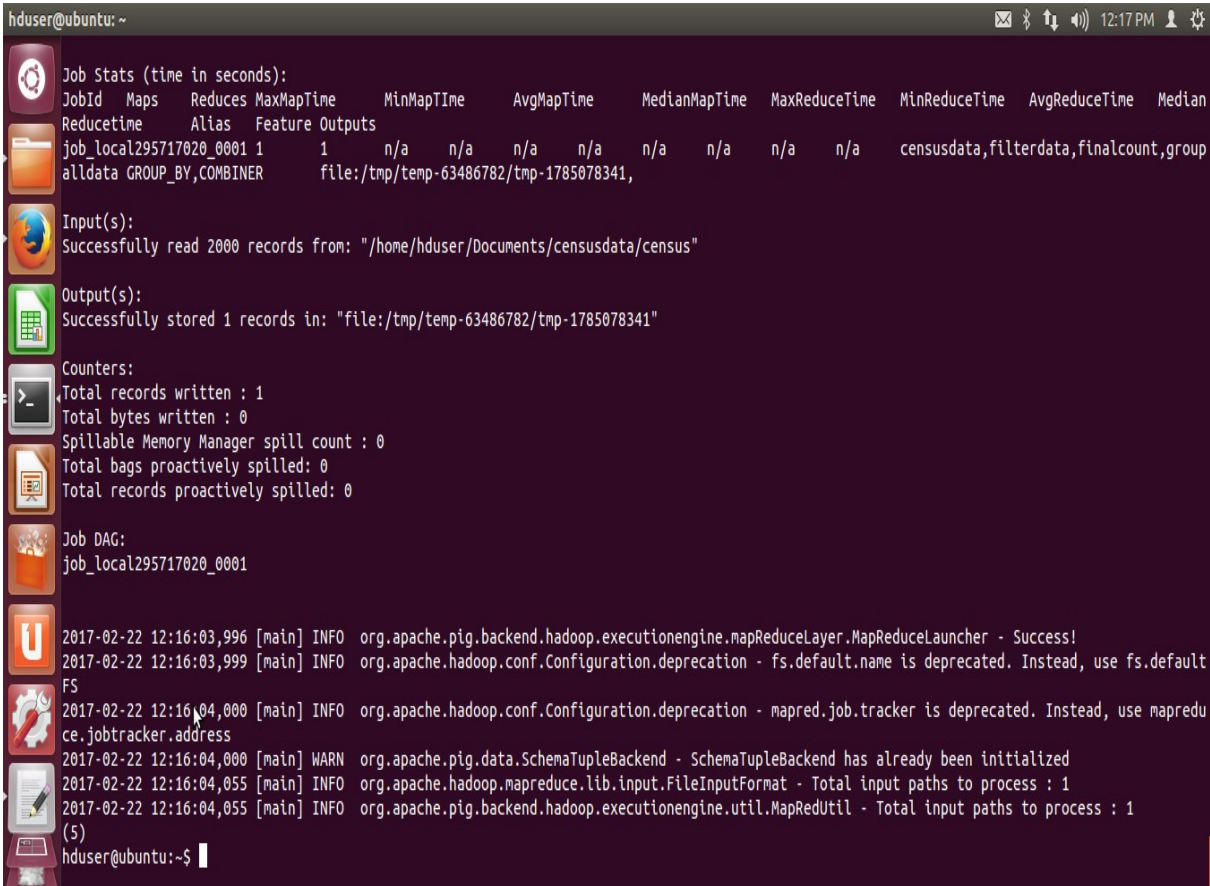
```
grunt> step3 = FILTER filterdata BY parents == ' Not in universe';
```

```
grunt> groupalldata = GROUP step3 ALL;
```

```
grunt> finalcount = FOREACH groupalldata GENERATE COUNT(step3.age);
```

```
grunt> DUMP finalcount;
```

OUTPUT: 5



The screenshot shows a terminal window with the following output:

```
hduser@ubuntu: ~  
Job Stats (time in seconds):  
JobId  Maps  Reduces MaxMapTime  MinMapTime  AvgMapTime  MedianMapTime  MaxReduceTime  MinReduceTime  AvgReduceTime  Median  
ReduceTime  Alias  Feature Outputs  
job_local295717020_0001 1  n/a  n/a  n/a  n/a  n/a  n/a  censusdata,filterdata,finalcount,group  
alldata GROUP_BY,COMBINER  file:/tmp/temp-63486782/tmp-1785078341,  
Input(s):  
Successfully read 2000 records from: "/home/hduser/Documents/censusdata/census"  
Output(s):  
Successfully stored 1 records in: "file:/tmp/temp-63486782/tmp-1785078341"  
Counters:  
Total records written : 1  
Total bytes written : 0  
Spillable Memory Manager spill count : 0  
Total bags proactively spilled: 0  
Total records proactively spilled: 0  
Job DAG:  
job_local295717020_0001  
2017-02-22 12:16:03,996 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!  
2017-02-22 12:16:03,999 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.default  
FS  
2017-02-22 12:16:04,000 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapredu  
ce.jobtracker.address  
2017-02-22 12:16:04,000 [main] WARN org.apache.pig.data.SchemaTupleBackend - SchemaTupleBackend has already been initialized  
2017-02-22 12:16:04,055 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1  
2017-02-22 12:16:04,055 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1  
(s)  
hduser@ubuntu:~$
```

Task 2.b) Total number of people who are working but not filing tax.


```

grunt> censusdata = LOAD '/home/hduser/Documents/censusdata/census' USING
PigStorage(',') AS
(age:INT,education:CHARARRAY,maritalstatus:CHARARRAY,gender:CHARARRAY,taxfilestatus
:CHARARRAY,income:DOUBLE,

parents:CHARARRAY,country:CHARARRAY,citizen:CHARARRAY,weeksworked:INT);

grunt> filterstep1 = FILTER censusdata BY weeksworked>0;

grunt> filterstep2 = FILTER filterstep1 BY taxfilestatus == ' Nonfiler';

grunt> groupallstep = GROUP filterstep2 ALL;

grunt> finalcount = FOREACH groupallstep GENERATE COUNT(filterstep2.taxfilestatus);

grunt> dump finalcount;

```

OUTPUT : 33

```

hduser@ubuntu: ~
Job Stats (time in seconds):
JobId  Maps  Reduces  MaxMapTime  MinMapTime  AvgMapTime  MedianMapTime  MaxReduceTime  MinReduceTime  AvgReduceTime  Median
ReduceTime  Alias  Feature  Outputs
job_local1523103324_0001  1  1  n/a  n/a  n/a  n/a  n/a  n/a  n/a  censusdata,filterstep1,finalco
unt,groupallstep  GROUP_BY,COMBINER  file:/tmp/temp575090749/tmp-684977433,

Input(s):
Successfully read 2000 records from: "/home/hduser/Documents/censusdata/census"

Output(s):
Successfully stored 1 records in: "file:/tmp/temp575090749/tmp-684977433"

Counters:
Total records written : 1
Total bytes written : 0
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0

Job DAG:
job_local1523103324_0001

2017-02-22 12:19:38,615 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2017-02-22 12:19:38,629 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.default
FS
2017-02-22 12:19:38,630 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapredue
ce.jobtracker.address
2017-02-22 12:19:38,630 [main] WARN org.apache.pig.data.SchemaTupleBackend - SchemaTupleBackend has already been initialized
2017-02-22 12:19:38,665 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2017-02-22 12:19:38,665 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(33)
hduser@ubuntu:~$

```

Task 2.c) male and female ratio:

```

grunt>censusdata = LOAD '/home/hduser/Documents/censusdata/census' USING
PigStorage(',') AS
(age:INT,education:CHARARRAY,maritalstatus:CHARARRAY,gender:CHARARRAY,taxfilestatus

```

```
:CHARARRAY,income:DOUBLE,parents:CHARARRAY,country:CHARARRAY,citizen:CHARARRA  
Y,weeksworked:INT);
```

```
grunt>filtermale = FILTER censusdata BY gender == ' Male';
```

```
grunt>filterfemale = FILTER censusdata BY gender == ' Female';
```

```
grunt>groupallmale = GROUP filtermale ALL;
```

```
grunt>count_groupallmale = FOREACH groupallmale GENERATE COUNT(filtermale) as  
male_count;
```

```
grunt>groupallfemale = GROUP filterfemale ALL;
```

```
grunt>count_groupallfemale = FOREACH groupallfemale GENERATE COUNT(filterfemale) as  
female_count;
```

```
ratio = FOREACH groupallmale GENERATE  
CONCAT('MALE:',(chararray)(count_groupallmale.male_count)),  
CONCAT(' FEMALE:',(chararray)(count_groupallfemale.female_count));
```

```
dump ratio;
```

OUTPUT:

```
( Male:939)( Female:1061)
```

```
hduser@ubuntu: ~
groupallmale GROUP_BY, MULTI_QUERY
job_local156442236_0003 1 0 n/a n/a n/a n/a 0 0 0 ratio MAP_ONLY file:/tmp/temp-909153438/tmp266143956,
Input(s):
Successfully read 2000 records from: "/home/hduser/Documents/censusdata/census"
Output(s):
Successfully stored 1 records in: "file:/tmp/temp-909153438/tmp266143956"
Counters:
Total records written : 1
Total bytes written : 0
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0
Job DAG:
job_local105227306_0001 -> job_local1193956786_0002,
job_local1193956786_0002 -> job_local156442236_0003,
job_local156442236_0003
2017-02-22 12:06:55,267 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2017-02-22 12:06:55,271 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.default
FS
2017-02-22 12:06:55,273 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapredu
ce.jobtracker.address
2017-02-22 12:06:55,277 [main] WARN org.apache.pig.data.SchemaTupleBackend - SchemaTupleBackend has already been initialized
2017-02-22 12:06:55,303 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2017-02-22 12:06:55,304 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(MALE :939, FEMALE :1061)
hduser@ubuntu:~$ nano mfratio.pig
hduser@ubuntu:~$
```

For next task, we will be using map reduce programs

8. MAPREDUCE:

MapReduce is a framework designed for writing programs that process large volume of structured and unstructured data in parallel fashion across a cluster, in a reliable and fault-tolerant manner. MapReduce is all about Key-Value pair. The java.util.Map interface is used for key-value in Java.

A MapReduce program usually consists of the following 3 parts:

1. Mapper
2. Reducer
3. Driver

As the name, itself states Map and Reduce, the code is divided basically into two phases one is Map and second is Reduce. Both phase has an input and output as key-value pairs. Programmer has been given the liberty to choose the data model for the input and output for Map and Reduce both. Depending upon the business problem we need to use the appropriate data model.

8.1) What Mappers does?

- The Map function reads the input files as key/value pairs, processes each, and generates zero or more output key/value pairs.
- The Map class extends Mapper class which is a subclass of java.lang.Object:org.apache.hadoop.mapreduce.Mapper
- The input and output types of the map can be (and often are) different from each other.
- For example, if the application is doing a word count, the map function would break the line into words and output a key/value pair for each word. Each output pair would contain the word as the key and the number of instances of that word in the line as the value.
- The Map function is also a good place to filter any unwanted fields/ data from input file, we take the data only we are interested to remove unnecessary workload.

8.2) What Reducer does?

- The Reducer code reads the outputs generated by the different mappers as pairs and emits key value pairs.
- Reducer reduces a set of intermediate values which share a key to a smaller set of values.
- The Reduce class extends Reducer class which is a subclass of java.lang.Object : org.apache.hadoop.mapreduce.Reducer.
- Reducer has 3 primary phases: shuffle, sort and reduce.
- Each reduce function processes the intermediate values for a particular key generated by the map function. There exists a one-one mapping between keys and reducers.
- Multiple reducers run in parallel, as they are independent of one another. The number of reducers for a job is decided by the programmer. By default, the number of reducers is 1.
- The output of the reduce task is typically written to the FileSystem via OutputCollector.collect(WritableComparable, Writable)

8.3) TASK 3

Task 3.a) Total number of people born in United-States:

```
package census;
import java.io.IOException;
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.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.output.FileOutputFormat;
public class totalpeople {
public static class MapperClass extends Mapper<LongWritable, Text, Text, IntWritable>{
private final static IntWritable one = new IntWritable(1);
public void map(LongWritable key, Text censusdata, Context con) throws IOException,
InterruptedException
    {
        String[] word = censusdata.toString().split(",");
        String country=word[7];
        try {
            if(country.contentEquals(" United-States")){
                con.write(new Text("Total no of people born in US"),one);
            }
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
}

public static class ReducerClass extends Reducer<Text, IntWritable, Text, IntWritable> {
private IntWritable result = new IntWritable();
public void reduce(Text key, Iterable<IntWritable> valueList,Context con) throws
IOException, InterruptedException {
    try {
        int sum=0;
        for (IntWritable var : valueList) {
            var.get();
            sum++;
        }
        result.set(sum);
        con.write(key,result);
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}
}

```

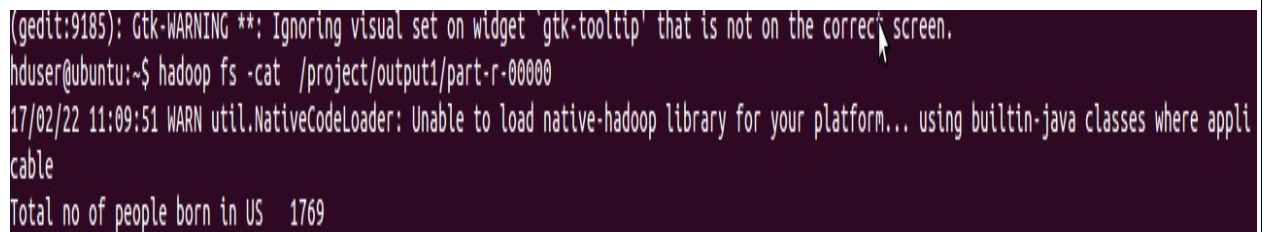
```

    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "total people");
        job.setJarByClass(totalpeople.class);
        job.setMapperClass(MapperClass.class);
        // job.setCombinerClass(IntSumReducer.class);
        //job.setNumReduceTasks(0);
        job.setReducerClass(ReducerClass.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}

```

output: Total number of people born in US 1769



A terminal window screenshot with a dark background. The first line shows a GTK warning: (gedit:9185): Gtk-WARNING **: Ignoring visual set on widget 'gtk-tooltip' that is not on the correct screen. The second line shows a Hadoop command: hduser@ubuntu:~\$ hadoop fs -cat /project/output1/part-r-00000. The third line shows a warning: 17/02/22 11:09:51 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable. The fourth line shows the output: Total no of people born in US 1769.

```

(gedit:9185): Gtk-WARNING **: Ignoring visual set on widget 'gtk-tooltip' that is not on the correct screen.
hduser@ubuntu:~$ hadoop fs -cat /project/output1/part-r-00000
17/02/22 11:09:51 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Total no of people born in US 1769

```

Task 3.b) total number of females widowed working

//total no of widowed working females

```

package census;

import java.io.IOException;

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.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.output.FileOutputFormat;


public class totalfemales {

    public static class MapperClass extends

        Mapper<LongWritable, Text, Text, IntWritable>{

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

        public void map(LongWritable key, Text censusdata, Context con)

            throws IOException, InterruptedException {

            String[] word = censusdata.toString().split(",");

            int weeksworked = Integer.parseInt(word[9]);

            String gender=word[3];

            String maritalstatus=word[2];

            try {

                if(maritalstatus.contentEquals(" Widowed") && weeksworked>0 &&

gender.contentEquals(" Female") ){

                    con.write(new Text("Widowedfemales"),one);

                }

            }

            catch (Exception e) {

                e.printStackTrace();

            }

        }

    }
}

```

```

        public static class ReducerClass extends Reducer<Text, IntWritable, Text,
IntWritable> {

            private IntWritable result = new IntWritable();

            public void reduce(Text key, Iterable<IntWritable> valueList, Context con)
throws IOException, InterruptedException {

                try {

                    int sum=0;

                    //int count = 0;

                    for (IntWritable var : valueList) {

                        var.get();

                        sum++;

                    }

                    result.set(sum);

                    con.write(key,result);

                }

                catch (Exception e) {

                    e.printStackTrace();

                }

            }

        }

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

        Configuration conf = new Configuration();

        Job job = Job.getInstance(conf, "total no.of widowed females");

        job.setJarByClass(totalfemales.class);

        job.setMapperClass(MapperClass.class);

        // job.setCombinerClass(IntSumReducer.class);

        //job.setNumReduceTasks(0);

```

```

    job.setReducerClass(ReducerClass.class);

    job.setOutputKeyClass(Text.class);

    job.setOutputValueClass(IntWritable.class);

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

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

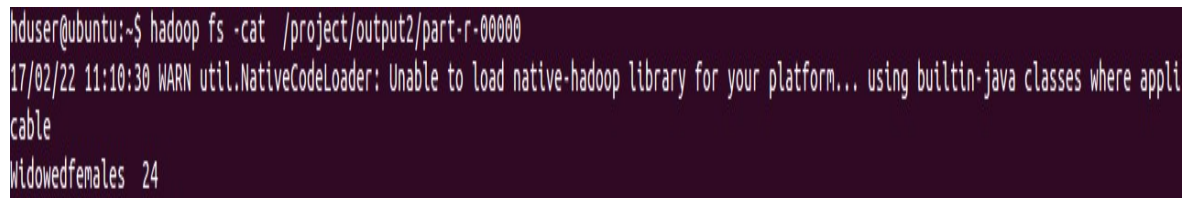
    System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}

```

output: Widowed females 24



A terminal window with a dark background. The first line shows a user running the command 'hadoop fs -cat /project/output2/part-r-00000'. The second line shows a warning message: '17/02/22 11:10:30 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable'. The third line shows the output of the command: 'Widowedfemales 24'.

task 3.c) Average age of divorced people

```

package census;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

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.output.FileOutputFormat;

public class CensusAvg {

    public static class MapperClass extends

```

```

Mapper<LongWritable, Text, Text, LongWritable> {

public void map(LongWritable key, Text censusdata, Context con)

throws IOException, InterruptedException {

String[] word = censusdata.toString().split(",");

String maritalstatus = word[2];

try {

        if(maritalstatus.contentEquals(" Divorced")){

Long age = Long.parseLong(word[0]);

con.write(new Text("average age"), new LongWritable(age));

}

}

catch (Exception e) {

e.printStackTrace();

}

}

}

public static class ReducerClass extends

Reducer<Text, LongWritable, Text,LongWritable> {

public void reduce(Text key, Iterable<LongWritable> valueList,

Context con) throws IOException, InterruptedException {

try {

long total=0;

long count=0;

for (LongWritable var : valueList) {

total+= var.get();

count+=1;

}

}

}

}

```

```

    }

    Long avg = total / count;

    con.write(key, new LongWritable(avg));

    }

    catch (Exception e) {

        e.printStackTrace();

    }

    }

    }

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

        Configuration conf = new Configuration();

        Job job = Job.getInstance(conf, "census avg");

        job.setJarByClass(CensusAvg.class);

        job.setMapperClass(MapperClass.class);

        // job.setCombinerClass(IntSumReducer.class);

        // job.setNumReduceTasks(0);

        job.setReducerClass(ReducerClass.class);

        job.setOutputKeyClass(Text.class);

        job.setOutputValueClass(LongWritable.class);

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

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

        System.exit(job.waitForCompletion(true) ? 0 : 1);

    }

    }

```

output: average age 44

```
nduser@ubuntu:~$ hadoop fs -cat /project/output3/part-r-00000
17/02/22 11:10:52 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
table
average age      44
nduser@ubuntu:~$
```

9. CONCLUSIONS:

Following is the conclusion that we can draw based on the tasks performed by us:

- As the census data is given in JSON format, by using Hive JSON Data can be easily converted to csv format.
- For group by, join and filter based data retrieval Pig is very efficient.
- MapReduce code written in Java makes the complex analysis quite easy.
- From the census data and after running all the tasks we can conclude that
- This project provides the of population summary of United States. The total population of people born in US is 1769 and the male: female ratio is 939:1061.
- we also found that there are children whose parents are not in universe and senior citizens, Nontax filers etc.
- Though MapReduce code written in Java makes the complex analysis quite easy, Hive and Pig are easier to retrieve data efficiently and lines of code is also very less compared to MapReduce programs.

10. REFERENCES:

<http://thornydev.blogspot.in/2013/07/querying-json-records-via-hive.html>

<http://hadoop.apache.org/>

https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html

<http://hortonworks.com/hadoop-tutorial/how-to-use-basic-pig-commands/>

<http://hortonworks.com/hadoop-tutorial/how-to-process-data-with-apache-hive/>

https://www.sas.com/en_us/insights/big-data/hadoop.html

11. ACKNOWLEDGEMENT:

I would like to thank NIIT from whom I have learned and will continue to learn. In particular, I would like to thank Mr. Annu Sharma sir and Mr. Sandeep Agarwal sir for their support and for providing so many case studies. I would also thank my tech mentor Mr. Shakun Tyagi for timely support and providing helpful suggestion.