ANALYSIS OF DIGITIZED CENSUS DATA

BY: GURVIN SINGH SURI

STUDENT ID: S170020200027

TABLE OF CONTENTS

1. BIG DATA: INTRODUCTION OF HADOOP

Big Data:

Big Data is very large, loosely structured data set that defies traditional storage. This data can be Human Generated Data or Machine Generated Data.

Human Generated Data is emails, documents, photos, and tweets. We are generating this data faster than ever. Just imagine the number of videos uploaded to You Tube and tweets swirling around. This data can be Big Data too.

Machine Generated Data is a new breed of data. This category consists of sensor data, and logs generated by 'machines' such as email logs, click stream logs, etc. Machine generated data is orders of magnitude larger than Human Generated Data.

Big data comes from multiple sources.

- Web Data
- Social media data : Sites like Facebook, Twitter, LinkedIn generate a large amount of data
- Click stream data : When users navigate a website, the clicks are logged for further analysis (like navigation patterns). Click stream data is important in on line advertising and E-Commerce.
- Sensor data : sensors embedded in roads to monitor traffic and other applications generate a large volume of data.
- Connected Devices : Smart phones are a great example. For example when we use a navigation application like Google Maps or Waze, our phone sends pings back reporting its location and speed (this information is used for calculating traffic hotspots). Just imagine hundreds of millions (or even billions) of devices consuming data and generating data.

Challenges of Big Data:

a. Sheer size of Big Data

Just the size of big data, makes it impossible (or at least cost prohibitive) to store in traditional storage like databases or conventional filers.
We are talking about cost to store gigabytes of data. Using traditional storage filers can cost a lot of money to store Big Data.

b. Big Data is unstructured or semi structured

A lot of Big Data is unstructured.

Lack of structure makes relational databases not well suited to store Big Data.

Plus, few databases can cope with storing billions of rows of data.

c. No point in just storing big data, if we can't process it

Storing Big Data is part of the game. We must process it to mine intelligence out of it. Traditional storage systems are pretty 'dumb' as in they just store bits -- They don't offer any processing power.

The traditional data processing model has data stored in a 'storage cluster', which is copied over to a 'compute cluster' for processing, and the results are written back to the storage cluster.

This model however doesn't quite work for Big Data because copying so much data out to a compute cluster might be too time consuming or impossible.

One solution is to process Big Data 'in place' -- as in a storage cluster doubling as a compute cluster.

Introduction of Hadoop:

Hadoop is an open-source implementation of Google's distributed computing framework (which is proprietary). It consists of two parts: Hadoop Distributed File System (HDFS), which is modeled after Google's GFS, and Hadoop MapReduce, which is modeled after Google's MapReduce.

MapReduce is a programming framework. Its description was published by Google in 2004 Much like other frameworks, such as Spring, Struts, or MFC, the MapReduce framework does some things for you, and provides a place for you to fill in the blanks. What MapReduce does is to organize multiple computers in a cluster to perform the calculations we need. It takes care of distributing the work between computers and of putting together the results of each computer's computation. Just as important, it takes care of hardware and network failures, so that they do not affect the flow of your computation. We in turn, should break our problem into separate pieces which can be processed in parallel by multiple machines, and we provide the code to do the actual calculation.

Hadoop and Big Data

Hadoop is built to run on a cluster of machines

Suppose we need to store lots of photos. We will start with a single disk. When we exceed a single disk, we may use a few disks stacked on a machine. When we max out all the disks on a single machine, we need to get a bunch of machines, each with a bunch of disks.

This is exactly how Hadoop is built. Hadoop is designed to run on a cluster of machines from the get go.

Advantages of Hadoop:

a. Hadoop clusters scale horizontally

More storage and compute power can be achieved by adding more nodes to a Hadoop cluster. This eliminates the need to buy increasingly powerful and expensive hardware.

b. Hadoop can handle unstructured / semi-structured data

Hadoop doesn't enforce a 'schema' on the data it stores. It can handle arbitrary text and binary data. So, Hadoop can 'digest' any unstructured data easily.

c. Hadoop clusters provides storage and computing

We saw how having separate storage and processing clusters is not the best fit for Big Data. Hadoop clusters provide storage and distributed computing all in one.

d. Hadoop provides storage for Big Data at reasonable cost

Storing Big Data using traditional storage can be expensive. Hadoop is built around commodity hardware. Hence it can provide large storage for a reasonable cost. Hadoop has been used in the field at Petabyte scale.

One study by Cloudera suggested that Enterprises usually spend around $25,000 to $50,000 dollars per tera byte per year. With Hadoop, this cost drops to few thousands of dollars per tera byte per year. And hardware gets cheaper and cheaper this cost continues to drop.

e. Hadoop allows to capture new or more data

Sometimes organizations don't capture a type of data, because it was too cost prohibitive to store it. Since Hadoop provides storage at reasonable cost, this type of data can be captured and stored.

One example would be web site click logs. Because the volume of these logs can be very high, very few organizations captured these. Now with Hadoop it is possible to capture and store the logs

f. With Hadoop, you can store data longer

To manage the volume of data stored, companies periodically purge older data. For example, only logs for the last 3 months could be stored and older logs were deleted. With Hadoop, it is possible to store the historical data longer. This allows new analytics to be done on older historical data.
For example, take click logs from a web site. Few years ago, these logs were stored for a brief period to calculate statics like popular pages, etc. Now with Hadoop it is viable to store these click logs for longer period.

g. Hadoop provides scalable analytics

There is no point in storing all the data, if we can't analyze them. Hadoop not only provides distributed storage, but also distributed processing as well. Meaning we can crunch a large volume of data in parallel.

The compute framework of Hadoop is called Map Reduce. Map Reduce has been proven to the scale of peta bytes.

h. Hadoop provides rich analytics

Native Map Reduce supports Java as primary programming language. Other languages like Ruby, Python and R can be used as well.

Of course, writing custom Map Reduce code is not the only way to analyze data in Hadoop. Higher level Map Reduce is available. For example, a tool named Pig takes English like data flow language and translates them into Map Reduce. Another tool Hive, takes SQL queries and runs them using Map Reduce.

2. HADOOP ECOSYSTEM:

The popularity of Hadoop has grown in the last few years, because it meets the needs of many organizations for flexible data analysis capabilities with an unmatched price-performance curve. The flexible data analysis features apply to data in a variety of formats, from unstructured data, such as raw text, to semi-structured data, such as logs, to structured data with a fixed schema.

Hadoop has been particularly useful in environments where massive server farms are used to collect data from a variety of sources. Hadoop can process parallel queries as big, background batch jobs on the same server farm. This saves the user from having to acquire additional hardware for a traditional database system to process the data (assume such a system can scale to the required size). Hadoop also reduces the effort and time required to load data into another system; you can process it directly within Hadoop. This overhead becomes impractical in very large data sets.

The Hadoop ecosystem includes other tools to address needs. Hive is a SQL dialect and Pig is a dataflow language for that hide the tedium of creating MapReduce jobs behind higher-level abstractions more appropriate for user goals. Zookeeper is used for federating services and Oozie is a scheduling system. Avro, Thrift and Protobuf are platform-portable data serialization and description formats.

HDFS - HADOOP DISTRIBUTED FILE SYSTEM

HDFS, or the Hadoop Distributed File System, gives the programmer unlimited storage (fulfilling a cherished dream for programmers). However, here are additional advantages of HDFS.

• Horizontal scalability. Thousands of servers holding petabytes of data. When you need even more storage, you don't switch to more expensive solutions, but add servers instead.

• Commodity hardware. HDFS is designed with relatively cheap commodity hardware in mind. HDFS is self-healing and replicating.

• Fault tolerance. Every member of the Hadoop zoo knows how to deal with hardware failures. If you have 10 thousand servers, then you will see one server fail every day, on average. HDFS foresees that by replicating the data, by default three times, on different data node servers. Thus, if one data node fails, the other two can be used to restore the third one in a different place.

HDFS implementation is modeled after GFS, Google Distributed File system.

YARN-YET ANOTHER RESOURCE NEGOTIATOR

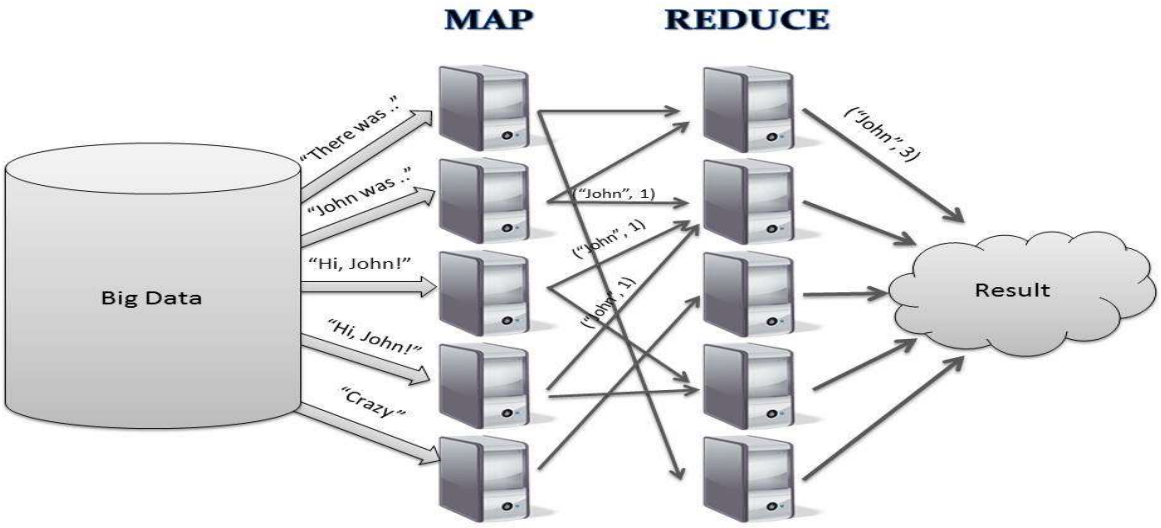
YARN is the architectural center of Hadoop that allows multiple data processing engines such as interactive SQL, real-time streaming, data science and batch processing to handle data stored in a single platform, unlocking an entirely new approach to analytics.

YARN is the foundation of the new generation of Hadoop and is enabling organizations everywhere to realize a modern data architecture.

YARN is the prerequisite for Enterprise Hadoop, providing resource management and a central platform to deliver consistent operations, security, and data governance tools across Hadoop clusters.

MAPREDUCE

MapReduce takes care of distributed computing. It reads the data, usually from its storage, the Hadoop Distributed File System (HDFS), in an optimal way. However, it can read the data from other places too, including mounted local file systems, the web, and databases. It divides the computations between different computers (servers, or nodes). It is also fault-tolerant.



If some of your nodes fail, Hadoop knows how to continue with the computation, by re-assigning the incomplete work to another node and cleaning up after the node that could not complete its task. It also knows how to combine the results of the computation in one place.

HIVE - DATA WAREHOUSING

Hive let us in and out of the HDFS cages, and we can talk SQL to Hive.

We can do a lot of data analysis with Hadoop, but we will also have to write MapReduce tasks. Hive takes that task upon itself. Hive defines a simple SQL-like query language, called QL, that enables users familiar with SQL to query the data.

At the same time, if our Hive program does almost what we need, but not quite, we can call on our MapReduce skill. Hive allows us to write custom mappers and reducers to extend the QL capabilities.

PIG - BIG DATA MANIPULATION

Pig: lets us move HDFS cages around, and we call this component “pig” because it speaks Pig Latin.

Just like Hive is the SQL of Big Data, then Pig Latin is the language of the stored procedures of Big Data. It allows us to manipulate large volumes of information, analyze them, and create new derivative data sets. Internally it creates a sequence of MapReduce jobs, and thus we can use this simple language to solve sophisticated large-scale problems.

HBASE: THE DATABASE FOR BIG DATA

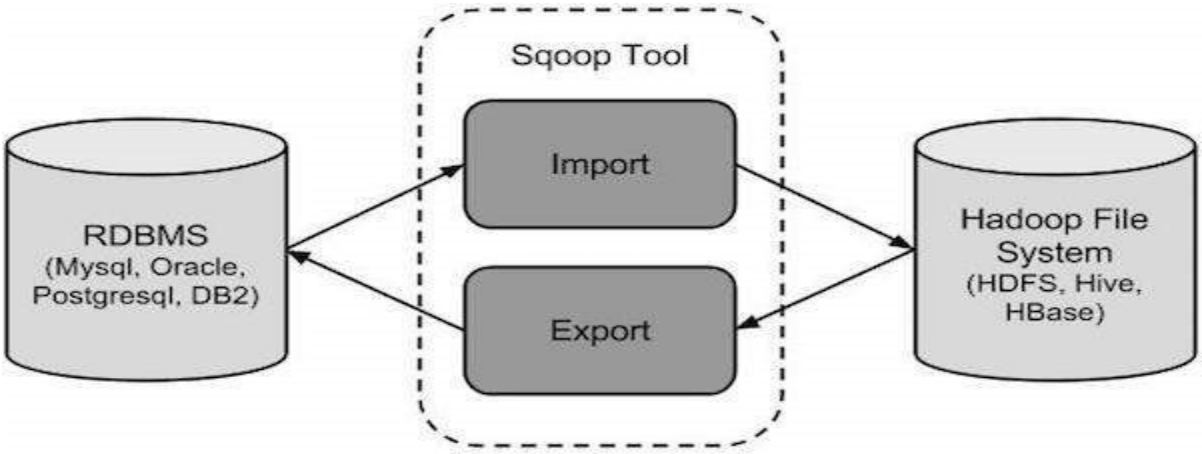
HBase is a database for Big Data, having up to millions of columns and billions of rows.

Another feature of HBase is that it is a key-value database, not a relational database. The key-value databases are considered as more fitting for Big Data since they don't store nulls values.

SQOOP: “SQL TO HADOOP AND HADOOP TO SQL”

Sqoop is a tool designed to transfer data between Hadoop and relational database servers. It is used to import data from relational databases such as MySQL, Oracle to Hadoop HDFS, and export from Hadoop file system to relational databases. The Apache Software Foundation provided it.

The following image describes the workflow of Sqoop.



Sqoop Import

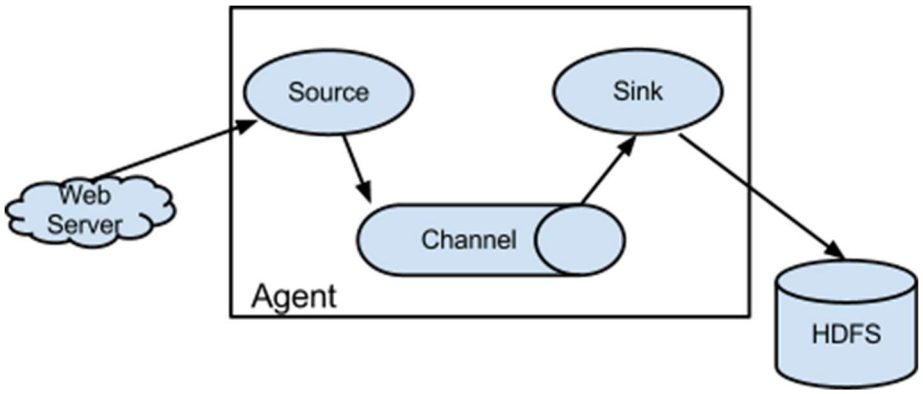
The import tool imports individual tables from RDBMS to HDFS. Each row in a table is treated as a record in HDFS. All records are stored as text data in text files or as binary data in Avro and Sequence files.

Sqoop Export

The export tool exports a set of files from HDFS back to an RDBMS. The files given as input to Sqoop contain records, which are called as rows in table. Those are read and parsed into a set of records and delimited with user-specified delimiter.

FLUME

Flume is a distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of log data. It has a simple and flexible architecture based on streaming data flows. It is robust and fault tolerant with tunable reliability mechanisms and many failover and recovery mechanisms. It uses a simple extensible data model that allows for online analytic application.



OOZIE:

Apache Oozie is a scheduler system to run and **manage Hadoop jobs** in a distributed environment. It allows to combine multiple complex jobs to be run in a sequential order to achieve a bigger task. Within a sequence of task, two or more jobs can also be programmed to run parallel to each other.

One of the main advantages of Oozie is that it is tightly integrated with Hadoop stack supporting various Hadoop jobs like **Hive, Pig, Sqoop** as well as system-specific jobs like **Java and Shell**.

It is responsible for triggering the workflow actions, which in turn uses the Hadoop execution engine to execute the task. Hence, Oozie can leverage the existing Hadoop machinery for load balancing, fail-over, etc.

Oozie detects completion of tasks through callback and polling. When Oozie starts a task, it provides a unique **callback HTTP URL** to the task, and notifies that URL when it is complete. If the task fails to invoke the callback URL, Oozie can poll the task for completion.

Following three types of jobs are common in Oozie −

- **Oozie Workflow Jobs** − These are represented as Directed Acyclic Graphs (DAGs) to specify a sequence of actions to be executed.
- **Oozie Coordinator Jobs** − These consist of workflow jobs triggered by time and data availability.
- **Oozie Bundle** − These can be referred to as a package of multiple coordinator and workflow jobs.

ZOOKEEPER

Zookeeper allows distributed processes to coordinate with each other through a shared hierarchical name space of data registers called registers znodes, much like a file system. Unlike normal file systems ZooKeeper provides its clients with high throughput, low latency, highly available, strictly ordered access to the znodes. The performance aspects of ZooKeeper allow it to be used in large distributed systems. The reliability aspects prevent it from becoming the single point of failure in big systems. Its strict ordering allows sophisticated synchronization primitives to be implemented at the client.

The name space provided by ZooKeeper is much like that of a standard file system. A name is a sequence of path elements separated by a slash (/). Every znode in Zookeeper's name space is identified by a path. And every znode has a parent whose path is a prefix of the znode with one less element; the exception to this rule is root (/) which has no parent. Also, exactly like standard file systems, a znode cannot be deleted if it has any children

3. PROJECT OBJECTIVE:

In our project, we have data of 2000 American citizens of various age groups having different educational qualifications and income. They provide the country their contribution in the form of tax which the government uses for the country benefits.

By analyzing this data, we can predict the citizens who are not paying their tax. We can check total male to female ratio number of adults, middle-aged, senior citizens, and elderly people.

We can sort people on basis of their gender, educational status, marital status, tax filer status and various other data which are available with us.

Based on the data available with us we are going to perform following queries using various technologies provided by the HADOOP ecosystem.

i. Ratio of male to female population.
ii. List of male and female citizens who are senior citizen and are still working.
iii. Total tax paid by working adults.
iv. Average income of persons whose age>25 and are not citizens of us.
v. Ratio of married to unmarried or single people (excluding children).
vi. People paying tax more than the average tax of all citizens working.
vii. Ratio of number of widowed females working to those not working
viii. Persons who are working, but not filing tax.
ix. People paying tax less than the average tax of all citizens working.

4. PROJECT IMPLEMENTATION

Assumptions:

1. Hadoop Cluster is Running
2. Ecosystem Products (Pig, Hive, Sqoop) are installed
3. Census data in JSON format is available with us and the same is loaded into the Hadoop file system for further table creation and analysis.

Prerequisites for All Jobs:

The Census data is in JSON format and hence needs to be converted in csv format in Hadoop file system.

Steps for Conversion

Step 1: In HIVE database 'census' is created and is used.Three tables are created in the following database.

create table jsondata(jsondata string) row format delimited stored as textfile;

create table agegroup(age int,agegroup String) row format delimited fields terminated by '\t' stored as textfile;

create table census_voter(age int,education string,maritalstatus string,gender string,taxfilerstatus string,income double,parents string,countryofbirth string,citizenship string,weekworked int) row format delimited fields terminated by ',' stored as textfile;

Step 2: Data is loaded into the first two tables 'jsondata' and 'age group'

LOAD DATA INPATH '/Project/CensusData/agegroup.dat' into table agegroup;

LOAD DATA INPATH '/Project/CensusData/sample.dat' OVERWRITE INTO TABLE jsondata;

Step 3: The data from the json table is loaded into the main table 'census_voter' and is stored in .csv format in the directory and the same data file is used for further analysis.

insert overwrite table census_voter select get_json_object(jsondata,"$.Age"), get_json_object(jsondata,"$.Education"),get_json_object(jsondata,"$.MaritalStatus"),get_json_object(jsondata,"$.Gender"),get_json_object(jsondata,"$.TaxFilerStatus"),get_json_object(jsondata,"$.Income"),get_json_object(jsondata,"$.Parents"),get_json_object(jsondata,"$.Co

untryOfBirth"),get_json_object(jsondata,"$.Citizenship"),get_json_object(jsondata,"$.Weeks Worked") from jsondata;

INSERT OVERWRITE DIRECTORY '/Project/Data/census_voter' row format delimited fields terminated by ',' select * from census_voter;

Input Table	census_voter
Data Elements	Age, Education, Marital Status, Gender, TaxFiler Status, Income, Parents, Country of Birth, Citizenship, WeeksWorked

5. QUERIES

TASK 1: RATIO OF MALE AND FEMALE POPULATION TO TOTAL POPULATION.

TECHNOLOGY USED: MAPREDUCE JOB USING JAVA

Total = 2000

Male = 939

Female =1061

Ratio Male = Male/Total = 0.4695

Female = Female/Total = 0.5305

Implementation:

Step 1: census_voter data is loaded and the mapper's map function reads the data line by line from the file.

Step 2: Gender is taken as the key and value is initialized as 1 for each key.

Step 3: The key-value pair (i.e. Gender,1) are passed to the reducer for totaling number of counts per Gender.

Step 4: The total number of records value read by the mapper is retrieved from the mapper and value is stored in a variable defining total number of records.

Step 5: The sum of the total of each gender is separately divided by the total number of records giving the ratio for each gender.

SAMPLE OUTPUT

Female Ratio	**0.5305**
Male Ratio	**0.4696**

TASK 2: LIST OF MALE AND FEMALE CITIZENS WHO ARE SENIOR CITIZEN AND ARE STILL WORKING.

TECHNOLOGY USED: MAPREDUCE JOB USING JAVA

Implementation:

Step 1: census_voter data is loaded and the mapper's map function reads the data line by line from the file.

Step 2: Gender is taken as the key and value is initialized as 1 for each key where age is greater than or equal to 60 and weeks worked.>0.

Step 3: The key-value pair (i.e. Gender,1) are passed to the reducer for totaling number of counts per Gender.

Step 4: The sum of the total of each gender is separately stored giving the number of senior citizens working per gender.

SAMPLE OUTPUT

Female	**30**
Male	**33**

TASK 3: TOTAL TAX PAID BY WORKING ADULTS.

TECHNOLOGY USED: MAPREDUCE JOB USING JAVA

Implementation:

Step 1: census_voter data is loaded and the mapper's map function reads the data line by line from the file.

Step 2: Age is taken as the key first and depending on the values the data is sorted in following groups and the key name is changed to group names.

a. Adults – Age: 18-40 years

b. Elderly – Age: Above 80 years

c. Middle Aged – Age: 41-60 yrs

d. Senior Citizens -- Age:61-80 yrs

Step 3: The data is again sorted for records having weeksworked value >0.

Step 4: The income is retrieved from the sorted data and are stored in the form of key-value pair (i.e. Agegroup,income) and are passed to the reducer for totaling the income and generate tax value per Agegroup.

Step 5: The taxable amount value of each Agegroup is passed on as the output by the reducer.

SAMPLE OUTPUT:

Adults	**109837.79**
Elderly	**169.76**
Middle Aged	**65482.85**
Senior Citizens	**9444.51**

TASK 4: AVERAGE INCOME OF PERSONS WHOSE AGE>25 AND ARE NOT CITIZENS OF US.

TECHNOLOGY USED: HIVE

HIVE QUERY

```
select Round(Avg(Income),2)As AVERAGE_INCOME_OF_PERSONS from census_voter
where CountryOfBirth!=' United-States' and Age>25;
```

SAMPLE OUTPUT:

average_income_of_persons	**1526.30**

TASK 5. RATIO OF MARRIED TO UNMARRIED OR SINGLE PEOPLE (EXCLUDING CHILDREN).

TECHNOLOGY USED: HIVE

Total no of citizens (excluding children and teenager below 18 years) = 1465

Married = 1131

Ratio of Married = 0.7720136518771331

Unmarried = 334

Ratio of Unmarried = 0.2279863481228669

HIVE QUERY

```
select Round(sum(case when maritalstatus not like ' Never married' then 1 else 0
end)/count(*),2) as Married_Ratio,

Round(sum(case when maritalstatus like ' Never married' then 1 else 0 end)/count(*),2) as
Unmarried_Ratio

from census_voter WHERE age >=18 ;
```

SAMPLE OUTPUT:

married_ratio	**0.77**
unmarried_ratio	**0.23**

TASK 6: PEOPLE PAYING TAX MORE THAN THE AVERAGE TAX OF ALL CITIZENS WORKING.

TECHNOLOGY USED: HIVE

(NOT POSSIBLE FOR TAX AS THE INCOME IS GIVEN NOT THE TAX PAID. WE APPLY 10% TAX AS PER AMERICA'S TAX POLICY.)

HIVE QUERY

```
create view AVG_Tax as select AVG((Income*10)/100) as Tax from census_voter where taxfilerstatus != ' Nonfiler';

create view AVG_Tax1 as select * from census_voter CV join AVG_Tax where ((Income*10)/100) < AVG_Tax.tax AND taxfilerstatus !=' Nonfiler' order by Income;

select count(*) as No_of_persons working from Avg_Tax1;
```

No_of_persons working paying Tax more than the average Tax Of All Citizens	**593**

TASK 7: RATIO OF NUMBER OF WIDOWED FEMALES WORKING TO THOSE NOT WORKING

TECHNOLOGY USED: PIG

Total number of widows = 89

Widows working =25

Widows not working = 64

Ratio of working widows = 0.28

Ratio of non working widows =0.72

PIG SCRIPT

```
census_voter = LOAD '/Project/Data/census_voter/000000_0' USING PigStorage(',') AS (age,education,maritalstatus,gender,taxfilerstatus,income,parents,countryofbirth,citizenship, weeksWorked);

voter_widowed = FILTER census_voter BY maritalstatus == ' Widowed';

voter_widowed_grp= GROUP voter_widowed all;

count_widows = FOREACH voter_widowed_grp GENERATE COUNT(voter_widowed)as totalwidows;

widow_working = FILTER voter_widowed BY weeksWorked>0;

working_widowed_grp= GROUP widow_working all;

count_working_widows = FOREACH working_widowed_grp GENERATE COUNT(widow_working)as working_widows;

widow_not_working = FILTER voter_widowed BY weeksWorked==0;

nonworking_widowed_grp= GROUP widow_not_working all;
```

```
count_not_working_widows = FOREACH nonworking_widowed_grp GENERATE
COUNT(widow_not_working)as nonworking_widows;

ratio = FOREACH working_widowed_grp GENERATE CONCAT('WORKING_WIDOWS  :
',(chararray)(ROUND_TO(((float)count_working_widows.working_widows/(float)count_wido
ws.totalwidows),2))) as Working_Widows,CONCAT('   NON_WORKING_WIDOWS
:',(chararray)(ROUND_TO(((float)count_not_working_widows.nonworking_widows/(float)cou
nt_widows.totalwidows),2))) as Non_Working_Widows;

dump ratio;
```

SAMPLE OUTPUT

Working Widows	**0.32**
Non-Working Widows	**0.68**

TASK 8: PERSONS WHO ARE WORKING, BUT NOT FILING TAX.

TECHNOLOGY USED: PIG

PIG SCRIPT

```
census_voter = LOAD '/Project/Data/census_voter/000000_0' USING PigStorage(',') AS
(age,education,maritalstatus,gender,taxfilerstatus,income,parents,countryofbirth,citizenship,
weeksWorked);

voter_working_nonfiler = FILTER census_voter BY weeksWorked>0 and taxfilerstatus == '
Nonfiler';

group_filer = GROUP voter_working_nonfiler all;

count_bag = FOREACH group_filer GENERATE COUNT(voter_working_nonfiler);

dump count_bag;
```

SAMPLE OUTPUT

No_of_persons working but not filing Tax	**33**

TASK 9: PEOPLE PAYING TAX LESS THAN THE AVERAGE TAX OF ALL CITIZENS WORKING.

TECHNOLOGY USED: PIG

PIG SCRIPT

```
census_voter = LOAD '/Project/Data/census_voter/000000_0' USING PigStorage(',') AS (age,education,maritalstatus,gender,taxfilerstatus,income,parents,countryofbirth,citizenship, weeksWorked);

Filer = FILTER census_voter BY taxfilerstatus != ' Nonfiler' And weeksWorked>0;

Group_Filer = GROUP Filer all;

income_filer= FOREACH Group_Filer GENERATE AVG(Filer.income) as inc;

income_filer_filter = FILTER Filer BY income < income_filer.inc AND weeksWorked>0;

group_filer = GROUP income_filer_filter all;

-- FOR LIST OF PEOPLE

dump income_filer_filter;

--FOR NUMBER OF PEOPLE

count_bag = FOREACH group_filer GENERATE COUNT(income_filer_filter);

dump count_bag;
```

SAMPLE OUTPUT

No_of_persons working paying Tax less than the average Tax Of All Citizens	**593**

6. CONCLUSION:

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

1. Working with HADOOP is very useful since we get accurate and efficient data in the form of key-value pairs.
2. HIVE is easy to configure and it is very simple to create tables in HIVE. Loading big datasets into HIVE Table is very simple and useful as we don't need to write insert query for each record separately. The whole file is loaded into the table all at once.
3. For normal group by, join and filter based data retrieval Pig is very efficient.
4. SQOOP is useful when we have data in SQL Tables that need to be imported in HADOOP filesystem or to HIVE
5. MapReduce code written in Java makes the complex analysis quite easy. Codes required to be written to collect user inputs and performing complex join operations are handled efficiently using this approach.

ACKNOWLEDGEMENTS

I want to express my sincere and deep sense of gratitude, I am thankful to NIIT for giving me permission to commence this project in the first instance, to do the necessary research work and to use the sample data provided by the institute.

I am deeply indebted to my mentors Mr. Annu Sharma, Mr. Sandeep Agarwal and Mr. Shakun Tyagi whose help, guidance, stimulating suggestions, encouragement, professional advice and experience sharing helped me all the time of study and writing of this report. Furthermore, I want to thank all the IT professionals working in NIIT who have helped me throughout the course training.

GURVIN SINGH SURI

Student Id: S170020200027