**Presented by :- SHREYAS S. PUROHIT**

**NIIT**

**Thane**



**Analysis and Summarization Of H1B Applicants**

**What is Big Data?**

Big data is a term that describes the large volume of data – both structured and unstructured – that inundates a business on a day-to-day basis. It is collection of large datasets that cannot be processed using traditional computing techniques. Big data is not merely a data; rather it has become a complete subject, which involves various tools, techniques and frameworks.

**Examples of Big Data:-**

Following are some the examples of 'Big Data'-

|  |  |
| --- | --- |
| [Introduction to BIG DATA: Types, Characteristics & Benefits](https://cdn.guru99.com/images/Big_Data/061114_0759_WhatIsBigDa2.jpg) | The **New York Stock Exchange** generates about ***one terabyte*** of new trade data per day. |
| Statistic shows that **500+terabytes** of new data gets ingested into the databases of social media site **Facebook**, every day. This data is mainly generated in terms of photo and video uploads, message exchanges, putting comments etc. | [Introduction to BIG DATA: Types, Characteristics & Benefits](https://cdn.guru99.com/images/Big_Data/061114_0759_WhatIsBigDa3.jpg) |
|  |  |
| [Introduction to BIG DATA: Types, Characteristics & Benefits](https://cdn.guru99.com/images/Big_Data/061114_0759_WhatIsBigDa4.jpg) | Single Jet engine can generate **10+terabytes** of data in 30 minutes of a flight time. With many thousand flights per day, generation of data reaches up to many Petabytes. |

**Categories of 'Big Data'**

Big data' could be found in three forms:

1. **Structured**
2. **Unstructured**
3. **Semi-structured**

**1) Structured** :-

Any data that can be stored, accessed and processed in the form of fixed format is termed as a 'structured' data. Over the period of time, talent in computer science have achieved greater success in developing techniques for working with such kind of data (where the format is well known in advance) and also deriving value out of it. However, now days, we are foreseeing issues when size of such data grows to a huge extent, typical sizes are being in the rage of multiple zettabyte.

**Examples Of Structured Data :-**

An 'Employee' table in a database is an example of Structured Data

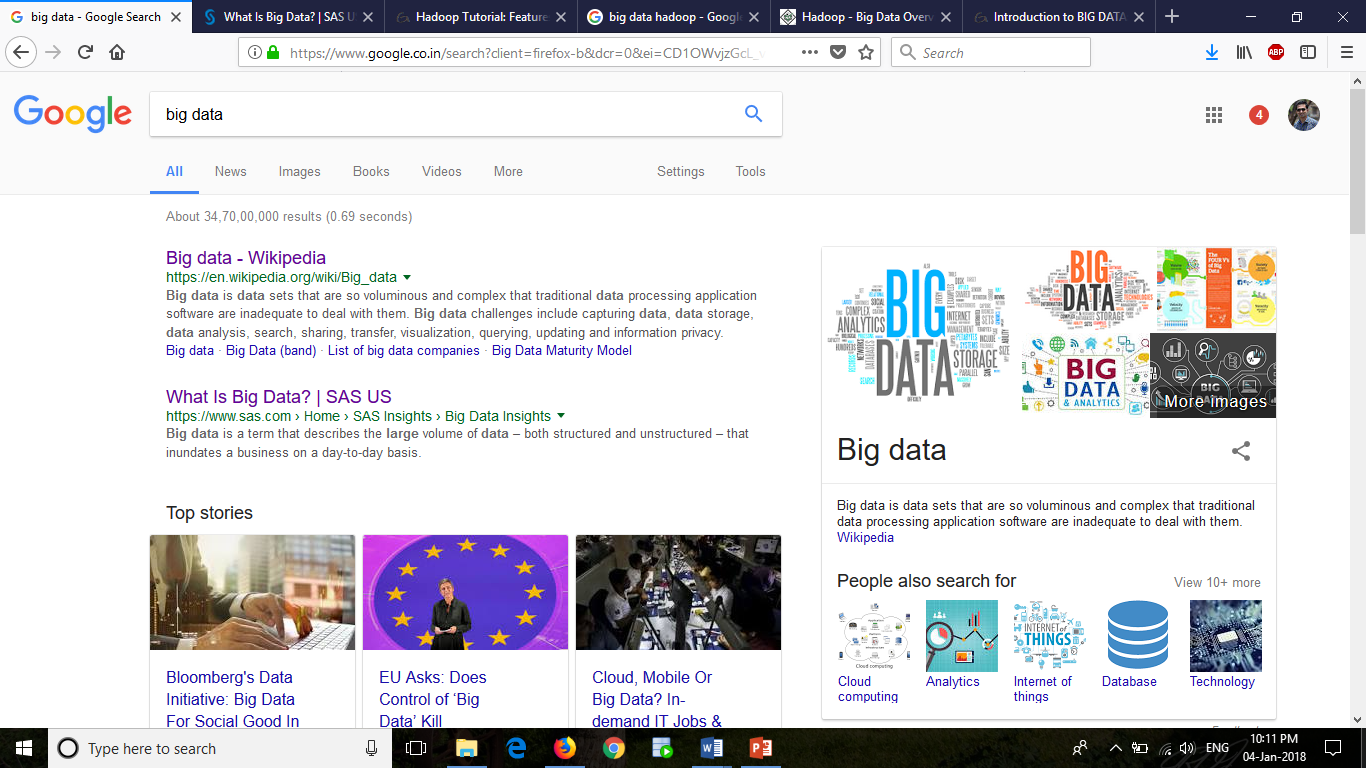
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Employee ID** | **Employee Name** | **Gender** | **Department** | **Salary\_In\_lacs** |
| 2365 | Rajesh Kulkarni | Male | Finance | 650000 |
| 3398 | Pratibha Joshi | Female | Admin | 650000 |
| 7465 | Shushil Roy | Male | Admin | 500000 |
| 7500 | Shubhojit Das | Male | Finance | 500000 |
| 7699 | Priya Sane | Female | Finance | 550000 |

**2) Unstructured:-**

Any data with unknown form or the structure is classified as unstructured data. In addition to the size being huge, un-structured data poses multiple challenges in terms of its processing for deriving value out of it. Typical example of unstructured data is, a heterogeneous data source containing a combination of simple text files, images, videos etc. Now a day organizations have wealth of data available with them but unfortunately, they do not know how to derive value out of it since this data is in its raw form or unstructured format.

**Examples of Un-structured Data**:-

Output returned by 'Google Search'



**3) Semi-structured:-**

Semi-structured data can contain both the forms of data. We can see semi-structured data as a structured in form but it is actually not defined with e.g. a table definition in relational DBMS. Example of semi-structured data is a data represented in XML file.

**Examples Of Semi-structured Data:-**

Personal data stored in a XML file-

<rec><name>Prashant Rao</name><sex>Male</sex><age>35</age></rec>

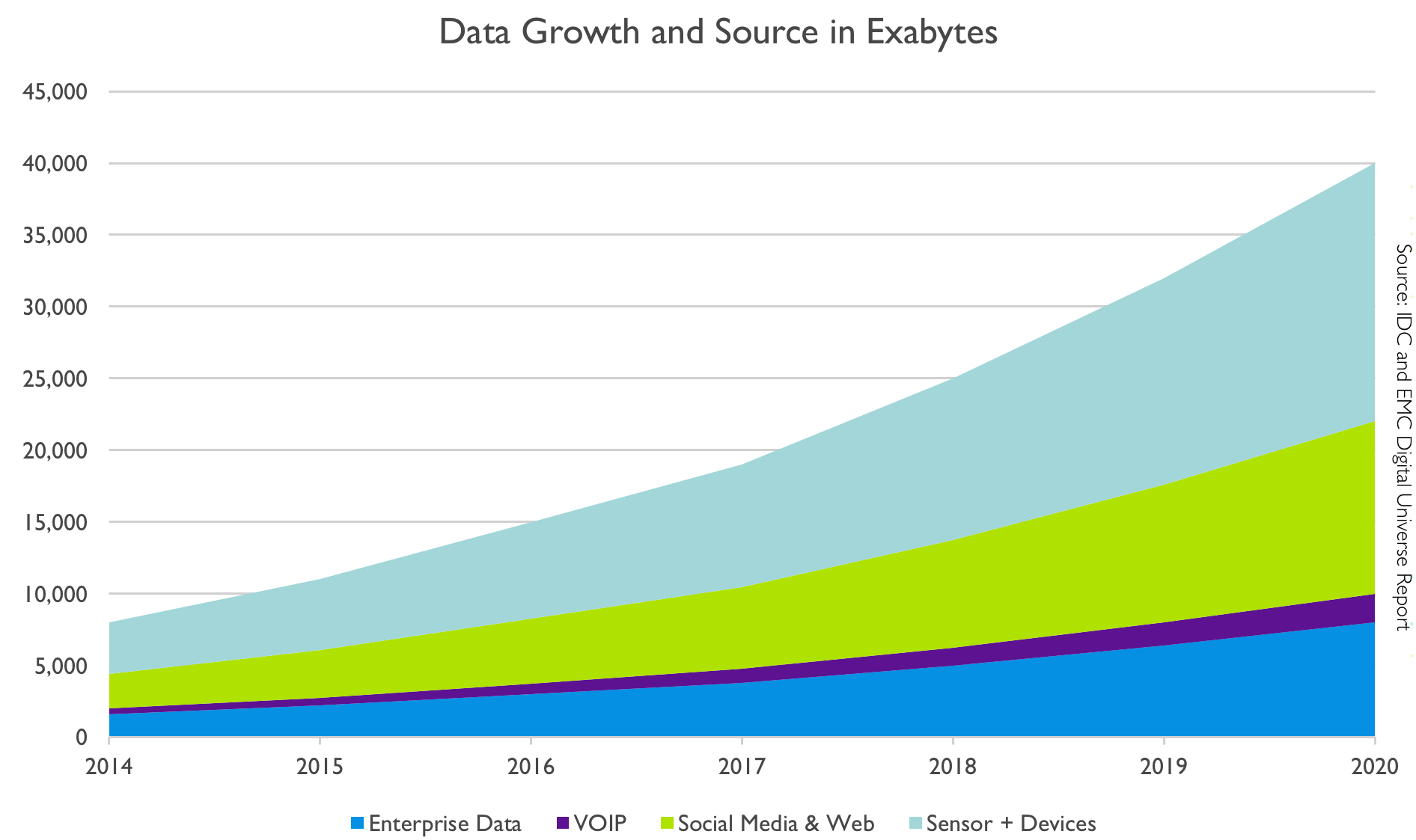
<rec><name>Seema R.</name><sex>Female</sex><age>41</age></rec>

<rec><name>Satish Mane</name><sex>Male</sex><age>29</age></rec>

<rec><name>Subrato Roy</name><sex>Male</sex><age>26</age></rec>

<rec><name>Jeremiah J.</name><sex>Male</sex><age>35</age></rec>

**Data Growth over years:-**



**The Big 5 Vs of Big Data**

**1) Volume *–*** The name 'Big Data' itself is related to a size which is enormous. Size of data plays very crucial role in determining value out of data. Also, whether a particular data can actually be considered as a Big Data or not, is dependent upon volume of data. Hence, **'Volume'** is one characteristic which needs to be considered while dealing with 'Big Data'.

**2) Variety *–*** The next aspect of 'Big Data' is its **variety**. Variety refers to heterogeneous sources and the nature of data, both structured and unstructured. During earlier days, spreadsheets and databases were the only sources of data considered by most of the applications. Now days, data in the form of emails, photos, videos, monitoring devices, PDFs, audio, etc. is also being considered in the analysis applications. This variety of unstructured data poses certain issues for storage, mining and analyzing data.

**3) Velocity *–*** The term **'velocity'** refers to the speed of generation of data. How fast the data is generated and processed to meet the demands, determines real potential in the data. Big Data Velocity deals with the speed at which data flows in from sources like business processes, application logs, networks and social media sites, sensors, mobile devices etc. The flow of data is massive and continuous.

**4) Veracity - Veracity** refers to the messiness or trustworthiness of the data. With many forms of big data, quality and accuracy are less controllable (just think of Twitter posts with hash tags, abbreviations, typos and colloquial speech as well as the reliability and accuracy of content) but big data and analytics technology now allows us to work with these type of data. The volumes often make up for the lack of quality or accuracy.

**5) Value -** It is all well and good having access to big data but unless we can turn it into value it is useless. So you can safely argue that 'value' is the most important V of Big Data. It is important that businesses make a business case for any attempt to collect and leverage big data.

**What Comes Under Big Data?**

Big data involves the data produced by different devices and applications. Given below are some of the fields that come under the umbrella of Big Data.

* **Black Box Data** : It is a component of helicopter, airplanes, and jets, etc. It captures voices of the flight crew, recordings of microphones and earphones, and the performance information of the aircraft.
* **Social Media Data** : Social media such as Facebook and Twitter hold information and the views posted by millions of people across the globe.
* **Stock Exchange Data** : The stock exchange data holds information about the ‘buy’ and ‘sell’ decisions made on a share of different companies made by the customers.
* **Power Grid Data** : The power grid data holds information consumed by a particular node with respect to a base station.
* **Transport Data** : Transport data includes model, capacity, distance and availability of a vehicle.
* **Search Engine Data** : Search engines retrieve lots of data from different databases.



**Benefits of Big Data Processing:-**

Ability to process 'Big Data' brings in multiple benefits, such as-

**1) Businesses can utilize outside intelligence while taking decisions**

Access to social data from search engines and sites like facebook, twitter are enabling organizations to fine tune their business strategies.

**2) Improved customer service**

Traditional customer feedback systems are getting replaced by new systems designed with 'Big Data' technologies. In these new systems, Big Data and natural language processing technologies are being used to read and evaluate consumer responses.

**3) Early identification of risk to the product/services, if any**

**4) Better operational efficiency**

'Big Data' technologies can be used for creating staging area or landing zone for new data before identifying what data should be moved to the data warehouse. In addition, such integration of 'Big Data' technologies and data warehouse helps organization to offload infrequently accessed data.

**5) Cost Savings :**  Some tools of Big Data like Hadoop and Cloud-Based Analytics can bring cost advantages to business when large amounts of data are to be stored and these tools also help in identifying more efficient ways of doing business.

**6)** **Time Reductions :** The high speed of tools like Hadoop and in-memory analytics can easily identify new sources of data which helps businesses analyzing data immediately and make quick decisions based on the learnings.

**7) New Product Development :** By knowing the trends of customer needs and satisfaction through analytics you can create products according to the wants of customers.

**Challenges of Big Data Processing:-**

**Data storage and quality :-** Companies and Organizations are growing at a very fast pace. Moreover, the growth of the companies rapidly increases the amount of data produced. The storage of this data is becoming a challenge for everyone. Options like data lakes/ warehouses are used to collect and store massive quantities of unstructured data in its native format. The problem, however, is when a data lakes/ warehouse try to combine inconsistent data from disparate sources, it encounters errors.  Inconsistent data, duplicates, logic conflicts, and missing data all result in data quality challenges.

**People who understand Big Data Analysis:-** Data Analysis is very important to make the huge amount of data being produced, useful. Therefore, there is a huge need for Big Data analysts and Data Scientists. The storage of quality data scientists has made it a job in great demand. It is important for a data scientist to have skills that are varied as the job is multidisciplinary. This is another challenge faced by companies. The number of data scientists available is very less in comparison to the amount of data being produced.To know more about the job of a data scientist, read our blogs on

**Good quality analysis:-** The companies and organizations use big data produced to make the best decisions possible.  Consequently, the data they are using should be accurate. If the data used to make decisions is not accurate it will result in ill-advised decisions that would ultimately be detrimental to the future success of their business.  This high reliance on data quality makes testing a high priority issue. This requires a lot of resources to ensure the accuracy of the information provided. The process of creating accurate data is very time consuming and requires the use of tools that can be expensive.

**Security and privacy of the data:-** Once, companies and organizations figure out how to use big data, it gives them a varied range of opportunities. However, it also involves big risks when it comes to the security and the privacy of the data. The tools used for analysis, stores, manages, analyses, and utilizes the data from a different variety of sources. This ultimately leads to a risk of exposure of the data, making it highly vulnerable. Therefore, the production of more and more data increases  security and privacy concerns. Thus making it essential for analysts and data scientists to consider these issues and deal with the data in a manner that will not lead to the disruption of privacy.

**Various sources of data :-** Dealing with the volume of data being produced and the velocity at which it is being produced is a challenge. Additionally, it is a challenge to manage the enormous number of sources that are producing this data. The data comes from the company’s internal sources like finance, marketing etc. Moreover, external sources like social media produce a huge amount of data. Therefore, making the data extremely diverse and massive. Any number of tools and Big Data experts will not be enough to manage and utilize this amount of data optimally.

**HADOOP**

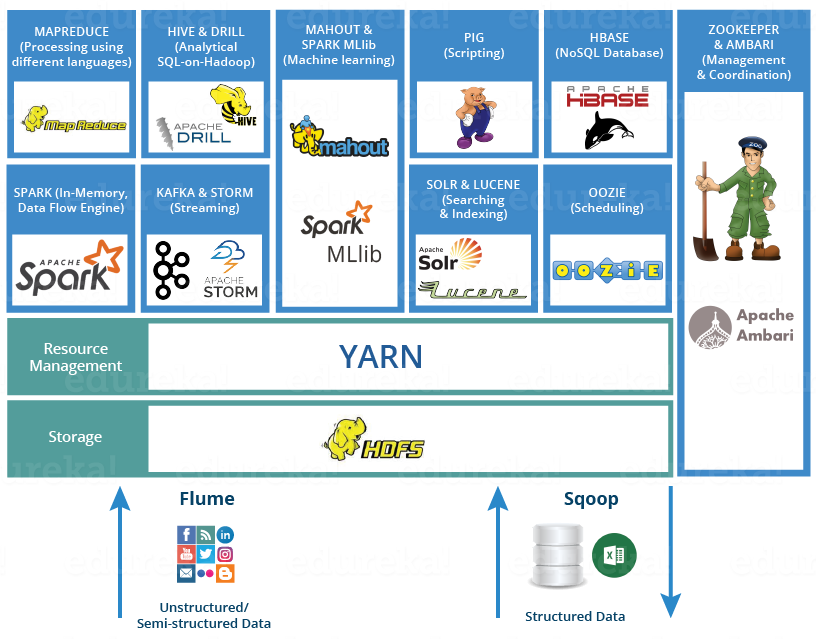
Hadoop is an open-source software framework for storing data and running applications on clusters of commodity hardware. It provides massive storage for any kind of data, enormous processing power and the ability to handle virtually limitless concurrent tasks or jobs. It allows distributed processing 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.

**Importance Of Hadoop:-**



* **Ability to store and process huge amounts of any kind of data, quickly.** With data volumes and varieties constantly increasing, especially from social media and the Internet of Things (IOT), that's a key consideration.
* **Computing power.** Hadoop's distributed computing model processes big data fast. The more computing nodes you use, the more processing power you have.
* **Fault tolerance.** Data and application processing are protected against hardware failure. If a node goes down, jobs are automatically redirected to other nodes to make sure the distributed computing does not fail. Multiple copies of all data are stored automatically.
* **Flexibility.** Unlike traditional relational databases, you don’t have to preprocess data before storing it. You can store as much data as you want and decide how to use it later. That includes unstructured data like text, images and videos.
* **Low cost.** The open-source framework is free and uses commodity hardware to store large quantities of data.
* **Scalability.** You can easily grow your system to handle more data simply by adding nodes. Little administration is required.

**Components Of Hadoop:-**



Apache Hadoop consists of two sub-projects –

1. **Hadoop MapReduce** : MapReduce is a computational model and software framework for writing applications which are run on Hadoop. These MapReduce programs are capable of processing enormous data in parallel on large clusters of computation nodes.
2. **HDFS** (**Hadoop Distributed File System**): HDFS takes care of storage part of Hadoop applications. MapReduce applications consume data from HDFS. HDFS creates multiple replicas of data blocks and distributes them on compute nodes in cluster. This distribution enables reliable and extremely rapid computations.

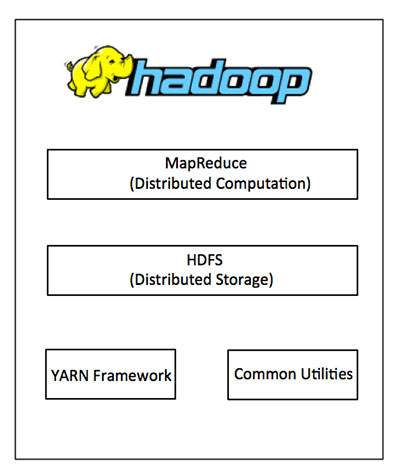
Although Hadoop is best known for MapReduce and its distributed file system- HDFS, the term is also used for a family of related projects that fall under the umbrella of distributed computing and large-scale data processing. Other Hadoop-related projects at [Apache](https://www.guru99.com/apache.html) include are **Hive**, **HBase**, **Mahout**, **Sqoop** , **Flume** and **ZooKeeper**.

**Hadoop Architecture :-**

Hadoop framework includes following four modules:

* **Hadoop Common:** These are Java libraries and utilities required by other Hadoop modules. These libraries provides file system and OS level abstractions and contains the necessary Java files and scripts required to start Hadoop.
* **Hadoop YARN:** This is a framework for job scheduling and cluster resource management.
* **Hadoop Distributed File System (HDFS™):** A distributed file system that provides high-throughput access to application data.
* **Hadoop MapReduce:** This is YARN-based system for parallel processing of large data sets.

We can use following diagram to depict these four components available in Hadoop framework.



**HDFS**

Hadoop comes with a distributed file system called **HDFS**(**HADOOP Distributed File Systems**) HADOOP based applications make use of HDFS. HDFS is designed for storing very large data files, running on clusters of commodity hardware. It is fault tolerant, scalable, and extremely simple to expand.

HDFS cluster primarily consists of a **NameNode** that manages the file system **Metadata** and a **DataNodes** that stores the **actual data**.

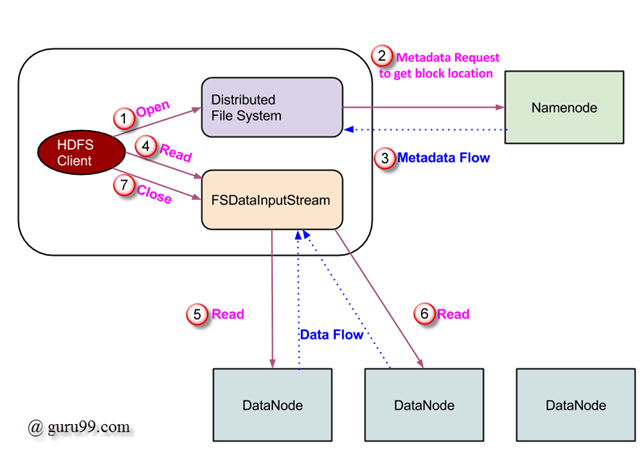
* **NameNode:-**NameNode can be considered as a master of the system. It maintains the file system tree and the metadata for all the files and directories present in the system. Two files **'Namespace image'** and the **'edit log'** are used to store metadata information. Namenode has knowledge of all the datanodes containing data blocks for a given file, however, it does not store block locations persistently. This information is reconstructed every time from datanodes when the system starts.
* **DataNode :-**DataNodes are slaves which reside on each machine in a cluster and provide the actual storage. It is responsible for serving, read and write requests for the clients.

Read/write operations in HDFS operate at a block level. Data files in HDFS are broken into block-sized chunks, which are stored as independent units. Default block-size is 128 MB.

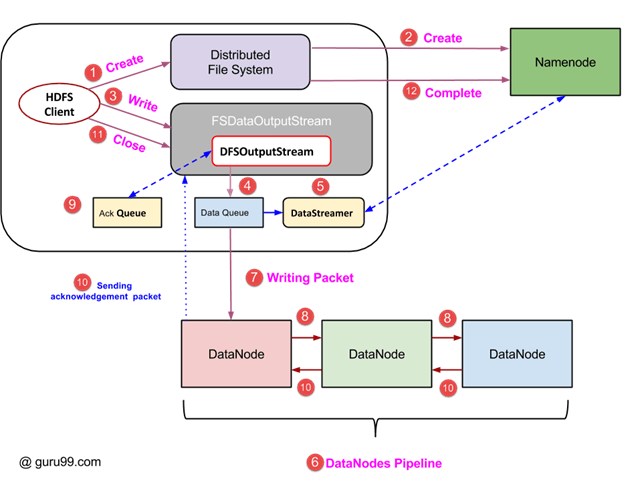
HDFS operates on a concept of data replication wherein multiple replicas of data blocks are created and are distributed on nodes throughout a cluster to enable high availability of data in the event of node failure.

## **Read and Write Operation In HDFS :-**

Data read request is served by HDFS, NameNode and DataNode. Let's call reader as a 'client'. Below diagram depicts file read operation in Hadoop.

[](https://cdn.guru99.com/images/Big_Data/061114_0923_LearnHDFSAB1.png)

**Write Operation:-**



**MAPREDUCE**

**MapReduce** is a programming model suitable for processing of huge data. Hadoop is capable of running MapReduce programs written in various languages: Java, Ruby, Python, and C++. MapReduce programs are parallel in nature, thus are very useful for performing large-scale data analysis using multiple machines in the cluster.

**MapReduce programs work in two phases:**

1. Map phase
2. Reduce phase.

Input to each phase are **key-value** pairs. In addition, every programmer needs to specify two functions: **map function** and **reduce function**.

The whole process goes through following phases of execution namely,

**Input Splits:** Input to a MapReduce job is divided into fixed-size pieces called **input splits**Input split is a chunk of the input that is consumed by a single map

**Mapping** :- This is very first phase in the execution of map-reduce program. In this phase data in each split is passed to a mapping function to produce output values. In our example, job of mapping phase is to count number of occurrences of each word from input splits (more details about input-split is given below) and prepare a list in the form of <word, frequency>

**Shuffling** :- This phase consumes output of Mapping phase. Its task is to consolidate the relevant records from Mapping phase output. In our example, same words are clubbed together along with their respective frequency.

**Reducing** :- In this phase, output values from Shuffling phase are aggregated. This phase combines values from Shuffling phase and returns a single output value. In short, this phase summarizes the complete dataset.

## **How MapReduce works** :-

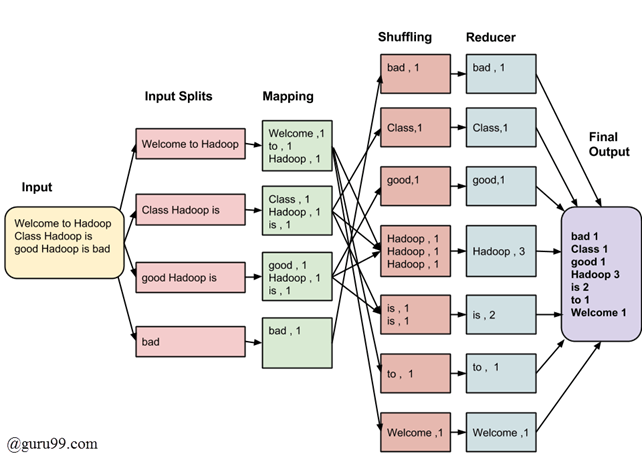
Let’s understand this with an example –

Consider we have following input data for our MapReduce Program

Welcome to Hadoop Class

Hadoop is good

Hadoop is bad

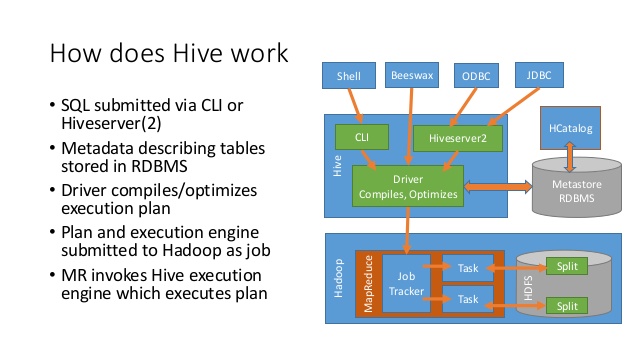
[](https://cdn.guru99.com/images/Big_Data/061114_0930_Introductio1.png)

**HIVE**

Hive is a data warehouse infrastructure tool to process structured data in Hadoop. It resides on top of Hadoop to summarize Big Data, and makes querying and analyzing easy. Apache Hive, allows SQL developers to write Hive Query Language (HQL) statements that are similar to standard SQL statements; now you should be aware that HQL is limited in the commands it understands, but it is still pretty useful. HQL statements are broken down by the Hive service into hive jobs and executed across a Hadoop cluster.

Hive makes job easy for performing operations like

* Data encapsulation
* Ad-hoc queries
* Analysis of huge datasets



## **Important characteristics of Hive :-**

* In Hive, tables and databases are created first and then data is loaded into these tables.
* Hive as data warehouse designed for managing and querying only structured data that is stored in tables.
* While dealing with structured data, Map Reduce doesn't have optimization and usability features like UDFs but Hive framework does. Query optimization refers to an effective way of query execution in terms of performance.
* Hive's SQL-inspired language separates the user from the complexity of Map Reduce programming. It reuses familiar concepts from the relational database world, such as tables, rows, columns and schema, etc. for ease of learning.
* Hadoop's programming works on flat files. So, Hive can use directory structures to "partition" data to improve performance on certain queries.
* A new and important component of Hive i.e. Metastore used for storing schema information. This Metastore typically resides in a relational database. We can interact with Hive using methods like
  + **Web GUI**
  + **Java Database Connectivity (JDBC) interface**
* Most interactions tend to take place over a command line interface (CLI). Hive provides a CLI to write Hive queries using Hive Query Language(HQL)
* Generally, HQL syntax is similar to the SQL syntax that most data analysts are familiar with. The Sample query below display all the records present in mentioned table name.
  + **Sample query** : Select \* from <TableName>
* Hive supports four file formats those are **TEXTFILE, SEQUENCEFILE, ORC and RCFILE** (Record Columnar File).
* For single user metadata storage, Hive uses derby database and for multiple user Metadata or shared Metadata case Hive uses MYSQL.

**Some of the key points about Hive :-**

* The major difference between HQL and SQL is that Hive query executes on Hadoop's infrastructure rather than the traditional database.
* The Hive query execution is going to be like series of automatically generated map reduce Jobs.
* Hive supports partition and buckets concepts for easy retrieval of data when the client executes the query.
* Hive supports custom specific UDF (User Defined Functions) for data cleansing, filtering, etc. According to the requirements of the programmers one can define Hive UDFs.

**ADVANTAGES OF USING APACHE HIVE :-**

1. Fits the low level interface requirement of Hadoop perfectly.

2. Supports external tables which make it possible to process data without actually storing in HDFS.

3. It has a rule based optimizer for optimizing logical plans.

4. Supports partitioning of data at the level of tables to improve performance.

5. Meta store or Metadata store is a big plus in the architecture which makes the lookup easy.

**DISADVANTAGES OF USING APACHE HIVE** :-

1. No support for update and delete.

2. No support for singleton inserts. Data is required to be loaded from a file using LOAD command.

3. No access control implementation.

4. Correlated sub queries are not supported.

**PIG**

In Map Reduce framework, programs need to be translated into a series of Map and Reduce stages. However, this is not a programming model which data analysts are familiar with. So, in order to bridge this gap, an abstraction called Pig was built on top of Hadoop.

Pig is a high level programming language useful for analyzing large data sets. Pig was a result of development effort at Yahoo!

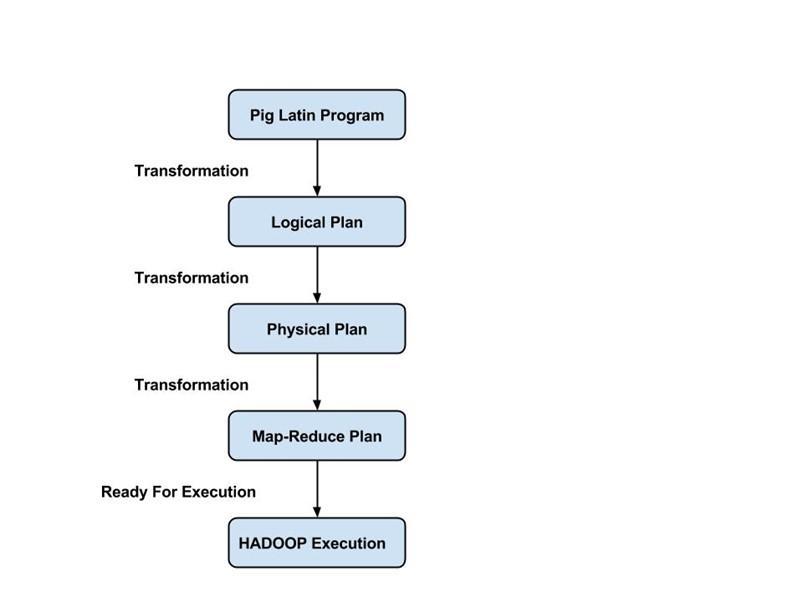
Pig enables people to focus more on **analyzing bulk data sets and to spend less time in writing Map-Reduce programs.**

Pig consists of two components: -

1. **Pig Latin,** which is a language
2. **Runtime environment,** for running PigLatin programs.

A Pig Latin program consist of a series of operations or transformations which are applied to the input data to produce output. These operations describe a data flow which is translated into an executable representation, by Pig execution environment. Underneath, results of these transformations are series of MapReduce jobs which a programmer is unaware of. So, in a way, Pig allows programmer to focus on data rather than the nature of execution.

PigLatin is a relatively stiffened language which uses familiar keywords from data processing e.g., Join, Group and Filter.

[](https://cdn.guru99.com/images/Big_Data/061114_1128_INTRODUCTIO2.jpg)

**Execution modes: -**

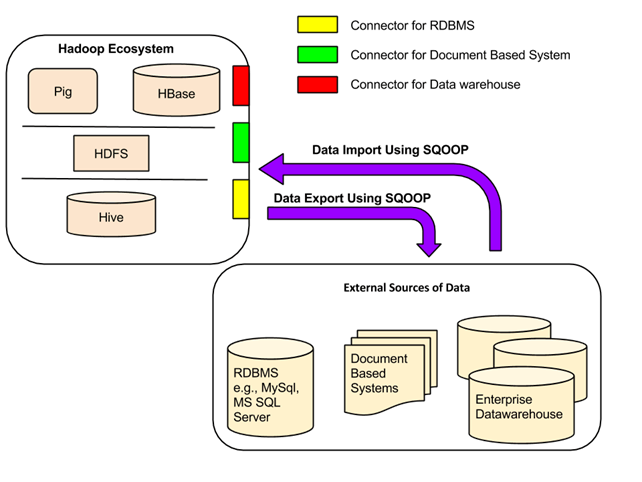
Pig has two execution modes:

1. **Local mode** : In this mode, Pig runs in a single JVM and makes use of local file system. This mode is suitable only for analysis of small data sets using Pig
2. **Map Reduce mode**: In this mode, queries written in Pig Latin are translated into MapReduce jobs and are run on a Hadoop cluster (cluster may be pseudo or fully distributed). MapReduce mode with fully distributed cluster is useful of running Pig on large data sets.

**SQOOP**

Apache **Sqoop** (**SQL-to-Hadoop**) is designed to support bulk import of data into HDFS from structured data stores such as relational databases, enterprise data warehouses, and NoSQL systems. Sqoop is based upon a connector architecture which supports plugins to provide connectivity to new external systems.

An example use case of Sqoop, is an enterprise that runs a nightly Sqoop import to load the day's data from a production transactional RDBMS into a Hive data warehouse for further analysis.



**Sqoop Connectors**

All the existing **Database Management Systems** are designed with SQL standard in mind. However, each DBMS differs with respect to dialect to some extent. So, this difference poses challenges when it comes to data transfers across the systems. Sqoop Connectors are components which help overcome these challenges.

Data transfer between Sqoop and external storage system is made possible with the help of Sqoop's connectors.

Sqoop has connectors for working with a range of popular relational databases, including MySQL, PostgreSQL, Oracle, SQL Server, and DB2. Each of these connectors knows how to interact with its associated DBMS. There is also a generic JDBC connector for connecting to any database that supports Java's JDBC protocol. In addition, Sqoop provides optimized MySQL and PostgreSQL connectors that use database-specific APIs to perform bulk transfers efficiently.

**FLUME**

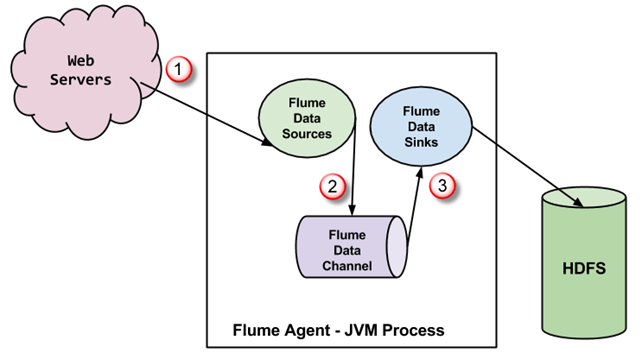
Apache Flume is a system used for moving massive quantities of streaming data into HDFS. Collecting log data present in log files from web servers and aggregating it in HDFS for analysis, is one common example use case of Flume.

Flume supports multiple sources like –

* 'tail' (which pipes data from local file and write into HDFS via Flume, similar to Unix command 'tail')
* System logs
* Apache log4j (enable Java applications to write events to files in HDFS via Flume).

**Data Flow in Flume :-**

* A **Flume agent** is a **JVM** process which has 3 components -**Flume Source**, **Flume Channel** and **Flume Sink**- through which events propagate after initiated at an external source .

[](https://cdn.guru99.com/images/Big_Data/061114_1038_Introductio2.png)

1. In above diagram, the events generated by external source (WebServer) are consumed by Flume Data Source. The external source sends events to Flume source in a format that is recognized by the target source.
2. Flume Source receives an event and stores it into one or more channels. The channel acts as a store which keeps the event until it is consumed by the flume sink. This channel may use local file system in order to store these events.
3. Flume sink removes the event from channel and stores it into an external repository like e.g., HDFS. There could be multiple flume agents, in which case flume sink forwards the event to the flume source of next flume agent in the flow.

## **Some Important features of FLUME :-**

* Flume has flexible design based upon streaming data flows. It is fault tolerant and robust with multiple failover and recovery mechanisms. Flume has different levels of reliability to offer which includes **'best-effort delivery'** and an **'end-to-end delivery'**. **Best-effort delivery** does not tolerate any Flume node failure whereas **'end-to-end delivery'** mode guarantees delivery even in the event of multiple node failures.
* Flume carries data between sources and sinks. This gathering of data can either be scheduled or event driven. Flume has its own query processing engine which makes it easy to transform each new batch of data before it is moved to the intended sink.
* Possible **Flume sinks** include **HDFS** and **Hbase**. Flume can also be used to transport event data including but not limited to network traffic data, data generated by social-media websites and email messages.

**Oozie :-**

Apache Oozie is a server-based worlflow scheduling system to manage Hadoop jobs.

Workflows in Oozie are defined as a collection of control flow and action nodes in a directed acyclic graph. Control flow nodes define the beginning and the end of a workflow (start, end, and failure nodes) as well as a mechanism to control the workflow execution path (decision, fork, and join nodes). Action nodes are the mechanism by which a workflow triggers the execution of a computation/processing task. Oozie provides support for different types of actions including Hadoop Map Reduce, Hadoop distributed file system operations, Pig, SSH, and email.

**Zookeeper :-**

Zookeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. All of these kinds of services are used in some form or another by distributed applications. Each time they are implemented there is a lot of work that goes into fixing the bugs and race conditions that are inevitable. Because of the difficulty of implementing these kinds of services, applications initially usually skimp on them, which make them brittle in the presence of change and difficult to manage. Even when done correctly, different implementations of these services lead to management complexity when the applications are deployed.

**Hbase:-**

HBase is a distributed column-oriented database built on top of the Hadoop file system. It is an open-source project and is horizontally scalable. HBase is a data model that is similar to Google’s big table designed to provide quick random access to huge amounts of structured data. It leverages the fault tolerance provided by the Hadoop File System (HDFS).It is a part of the Hadoop ecosystem that provides random real-time read/write access to data in the Hadoop File System.One can store the data in HDFS either directly or through HBase. Data consumer reads/accesses the data in HDFS randomly using HBase. HBase sits on top of the Hadoop File System and provides read and write access.

**Project Overview**

**Abstract :-**

The H-1B is an employment-based, non-immigrant visa category for temporary foreign workers in the United States. Every year, the US immigration department receives over 200,000 petitions and selects 85,000 applications through a random process. The application data is available for public access to perform in-depth longitudinal research and analysis. This data provides key insights into the prevailing wages for job titles being sponsored by US employers under H1-B visa category. In particular, in my project I utilize the 2011-2016 H-1B petition disclosure data to analyze the employers with the most applications, data science related job positions and relationship between salaries offered and cost of living index.

**Project Objective :-**

The H1B is an employment-based, non-immigrant visa category for temporary foreign workers in the United States. For a foreign national to apply for H1B visa, an US employer must offer a job and petition for H1B visa with the US immigration department. This is the most common visa status applied for and held by international students once they complete college/ higher education (Masters, Ph.D.) and work in a full-time position.

The Office of Foreign Labor Certification (OFLC) generates program data that is useful information about the immigration programs including the H1-B visa.

Various Technologies like **MapReduce**, **Pig** and **Hive** are used to perform operations on these data and the values are yielded.

**H1-B Case Study :-**

We will be performing analysis on the H1B visa applicants between the years 2011-2016. After analyzing the data, we can derive the following facts.

1 a) Is the number of petitions with Data Engineer job title increasing over time?

b) Find top 5 job titles who are having highest avg growth in applications.[ALL]

2 a) Which part of the US has the most Data Engineer jobs for each year?

b) find top 5 locations in the US who have got certified visa for each year.[certified]

3)Which industry(SOC\_NAME) has the most number of Data Scientist positions?

[certified]

4)Which top 5 employers file the most petitions each year? - Case Status - ALL

5) Find the most popular top 10 job positions for H1B visa applications for each year?

a) for all the applications

b) for only certified applications.

6) Find the percentage and the count of each case status on total applications for each year. Create a line graph depicting the pattern of All the cases over the period of time.

7) Create a bar graph to depict the number of applications for each year [All]

8) Find the average Prevailing Wage for each Job for each Year (take part time and full time separate). Arrange the output in descending order - [Certified and Certified Withdrawn.]

9) Which are the employers along with the number of petitions who have the success rate more than 70% in petitions. (total petitions filed 1000 OR more than 1000) ?

10) Which are the job positions along with the number of petitions which have the success rate more than 70% in petitions (total petitions filed 1000 OR more than 1000)?

11) Export result for question no 10 to MySql database.

**SUCCESS RATE % = (Certified + Certified Withdrawn)/Total x 100**

**The dataset has nearly 3 million records.**

Bottom of Form

.

The dataset description is as follows:

The columns in the dataset include:

* CASE\_STATUS: Status associated with the last significant event or decision. Valid values include “Certified,” “Certified-Withdrawn,” Denied,” and “Withdrawn”.

Certified: Employer filed the LCA, which was approved by DOL

Certified Withdrawn: LCA was approved but later withdrawn by employer

Withdrawn: LCA was withdrawn by employer before approval

Denied: LCA was denied by DOL

* EMPLOYER\_NAME: Name of employer submitting labour condition application.
* SOC\_NAME: the Occupational name associated with the SOC\_CODE. SOC\_CODE is the occupational code associated with the job being requested for temporary labour condition, as classified by the Standard Occupational Classification (SOC) System.
* JOB\_TITLE: Title of the job
* FULL\_TIME\_POSITION: Y = Full Time Position; N = Part Time Position
* PREVAILING\_WAGE: Prevailing Wage for the job being requested for temporary labour condition. The wage is listed at annual scale in USD. The prevailing wage for a job position is defined as the average wage paid to similarly employed workers in the requested occupation in the area of intended employment. The prevailing wage is based on the employer’s minimum requirements for the position.
* YEAR: Year in which the H1B visa petition was filed
* WORKSITE: City and State information of the foreign worker’s intended area of employment
* lon: longitude of the Worksite
* lat: latitude of the Worksite

**SOLUTIONS**

**1A) Is the number of petitions with Data Engineer job title increasing over time?**

=>

import java.io.IOException;

import java.util.TreeMap;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.DoubleWritable;

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 Q1A\_DataEngineerJob {

//Mapper Class

public static class MyMapper extends Mapper<LongWritable, Text, Text, IntWritable>

{

Text myKey = new Text();

IntWritable one = new IntWritable(1);

@Override

protected void map(LongWritable key, Text value, Context context)throws IOException, InterruptedException

{

String[] record = value.toString().split("\t");

String job\_title = record[4];

String year = record[7];

if(job\_title.contains("DATA ENGINEER"))

{

String str = "DATA ENGINEER"+","+year;

myKey.set(str);

context.write(myKey, one);

}

}

}

//Reducer Class

public static class MyReducer extends Reducer<Text, IntWritable, Text, DoubleWritable>

{

String[] years = {"2011","2012","2013","2014","2015","2016"};

double[] arr = new double[6];

TreeMap<String,Double> map = new TreeMap<String,Double>();

int i = 0;

@Override

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

{

int sum =0;

for(IntWritable val : values)

{

sum += val.get();

}

arr[i++] = sum;

}

@Override

protected void cleanup(Context context)throws IOException, InterruptedException

{

double avg = 0.0;

double sum1 = 0.0;

for(int i=0; i<6; i++ )

{

/\*if(i == 0)

{

context.write(new Text(years[i]), new DoubleWritable(0));

}

else

{

context.write(new Text(years[i]), new DoubleWritable((arr[i]-arr[i-1])/arr[i-1]\*100));

}\*/

try {

sum1 += (arr[i]-arr[i-1])/arr[i-1]\*100;

} catch (Exception e) {

System.out.println(e.getMessage());

}

}

avg = sum1 /5;

context.write(new Text("Data Engineer Average Growth For Five Years"), new DoubleWritable(avg));

}

}

//Main Method

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

Configuration conf = new Configuration();

Job job = Job.getInstance(conf,"Data Engineer Job Increasing");

job.setJarByClass(Q1A\_DataEngineerJob.class);

job.setMapperClass(MyMapper.class);

job.setReducerClass(MyReducer.class);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(IntWritable.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(DoubleWritable.class);

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

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

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

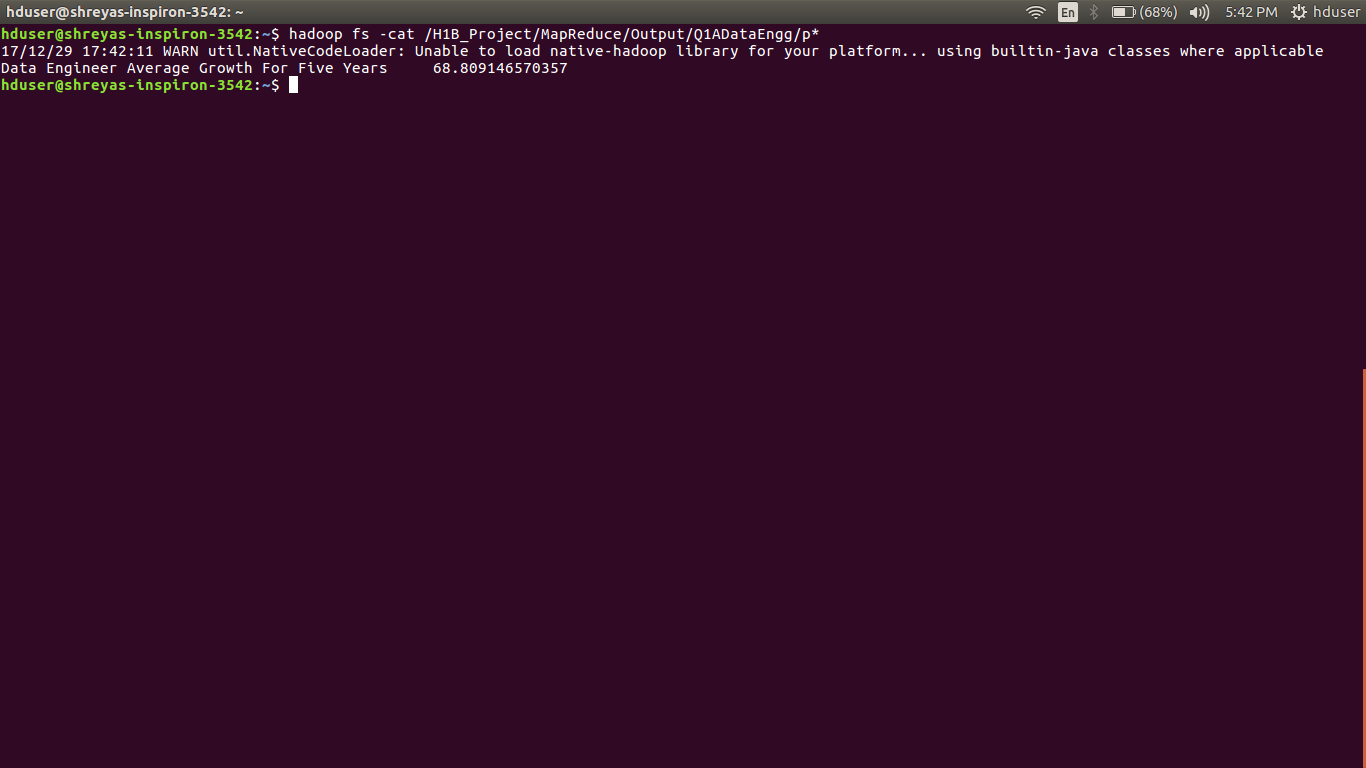
}

}

**Command:-**

hadoop jar h1bproject.jar Q1A\_DataEngineerJob/user/hive/warehouse/h1bproject.db/h1b\_final /H1B\_Project/MapReduce/Output/Q1ADataEngg

hadoop fs -cat /H1B\_Project/MapReduce/Output/Q1ADataEngg/p\*

**Output:-** 

========================================================

**Q1.B) Find top 5 job titles who are having highest avg growth in applications.[ALL]**

=>

import java.io.IOException;

import java.util.TreeMap;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.NullWritable;

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 Q1B\_Top5JobHighestGrowth {

public static class MapperClass extends Mapper<LongWritable, Text, Text, Text>

{

@Override

protected void map(LongWritable key, Text value, Context context)throws IOException, InterruptedException

{

String[] record = value.toString().split("\t");

String job\_title = record[4];

String year = record[7];

context.write(new Text(job\_title), new Text(year));

}

}

public static class ReducerClass extends Reducer<Text, Text, NullWritable, Text>

{

TreeMap<Double, Text> map = new TreeMap<Double, Text>();

@Override

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

{

int count2011 = 0;

int count2012 = 0;

int count2013 = 0;

int count2014 = 0;

int count2015 = 0;

int count2016 = 0;

double growth2012 = 0.0;

double growth2013 = 0.0;

double growth2014 = 0.0;

double growth2015 = 0.0;

double growth2016 = 0.0;

double averageGrowth = 0.0;

for (Text val : values) {

String year = val.toString();

if (year.equals("2011")) {

count2011++;

} else if (year.equals("2012")) {

count2012++;

} else if (year.equals("2013")) {

count2013++;

} else if (year.equals("2014")) {

count2014++;

} else if (year.equals("2015")) {

count2015++;

} else if (year.equals("2016")){

count2016++;

}

}

if (count2011 != 0) {

growth2012 = (double)(count2012 - count2011) \* 100 / (double)count2011;

} else {

growth2012 = 0;

}

if (count2012 != 0) {

growth2013 = (double)(count2013 - count2012) \* 100 / (double)count2012;

} else {

growth2013 = 0;

}

if (count2013 != 0) {

growth2014 = (double)(count2014 - count2013) \* 100 / (double)count2013;

} else {

growth2014 = 0;

}

if (count2014 != 0) {

growth2015 = (double)(count2015 - count2014) \* 100 / (double)count2014;

} else {

growth2015 = 0;

}

if (count2015 != 0) {

growth2016 = (double)(count2016 - count2015) \* 100 / (double)count2015;

} else {

growth2016 = 0;

}

averageGrowth = (growth2012 + growth2013 + growth2014 + growth2015 + growth2016) / 5;

String highestAvgGrowth = String.format("%f", averageGrowth);

/\*String total2011 = String.format("%d", count2011);

String total2012 = String.format("%d", count2012);

String total2013 = String.format("%d", count2013);

String total2014 = String.format("%d", count2014);

String total2015 = String.format("%d", count2015);

String total2016 = String.format("%d", count2016);\*/

String job\_title = key.toString();

//String myValue = job\_title+"\t"+total2011+"\t"+total2012+"\t"+total2013+"\t"+total2014+"\t"+total2015+"\t"+total2016;

String myValue = job\_title+"\t"+highestAvgGrowth;

//context.write(new DoubleWritable(averageGrowth), new Text(myValue));

//context.write(new DoubleWritable(averageGrowth), new Text(job\_title));

map.put(new Double(averageGrowth), new Text(myValue));

if(map.size() > 5)

{

map.remove(map.firstKey());

}

}

@Override

protected void cleanup(Context context) throws IOException, InterruptedException

{

for(Text top5 : map.descendingMap().values())

{

context.write(NullWritable.get(), top5);

}

}

}

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

Configuration conf = new Configuration();

Job job = Job.getInstance(conf,"Top 5 job titles who are having highest avg growth in applications");

job.setJarByClass(Q1B\_Top5JobHighestGrowth.class);

job.setMapperClass(MapperClass.class);

job.setReducerClass(ReducerClass.class);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(Text.class);

job.setOutputKeyClass(NullWritable.class);

job.setOutputValueClass(Text.class);

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

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

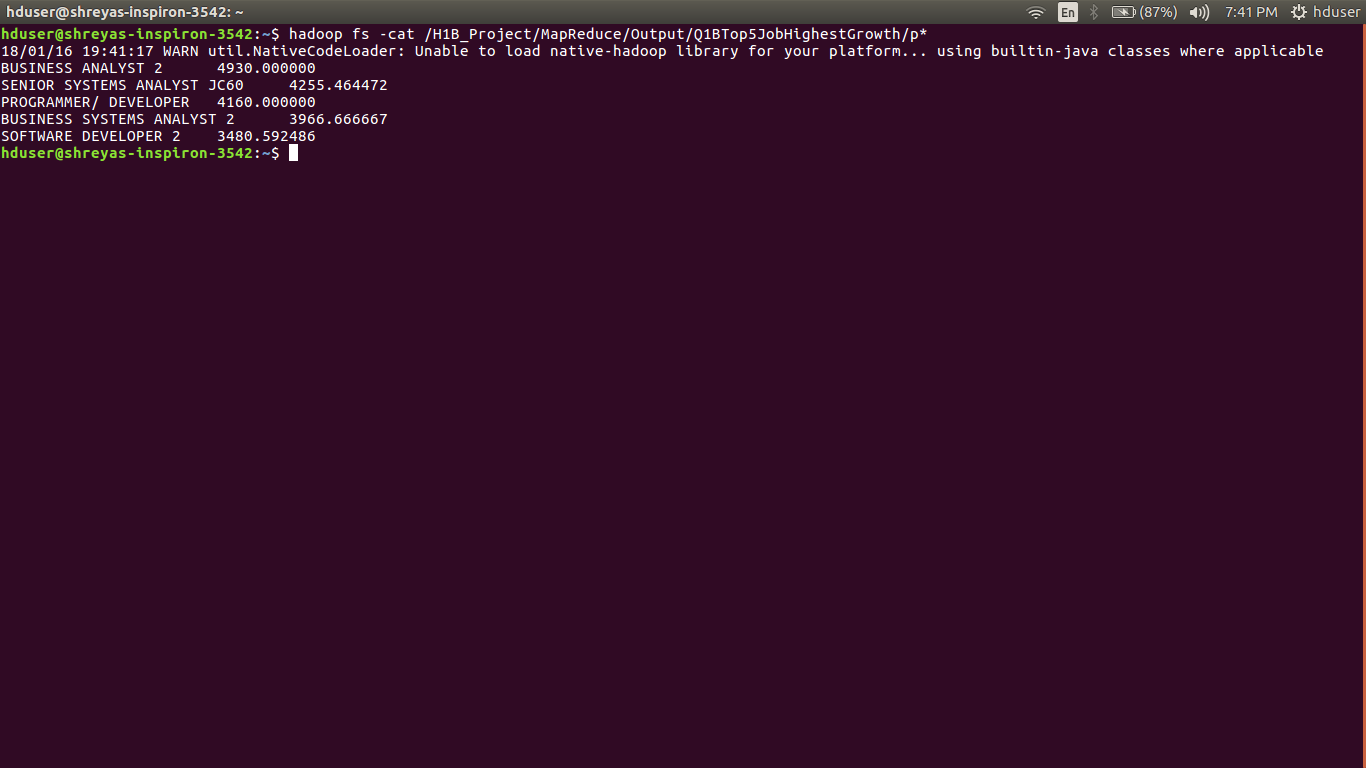
System.exit(job.waitForCompletion(true) ? 0 : 1);}}

**Command:-**

hadoop jar h1bproject.jar Q1B\_Top5JobHighestGrowth /user/hive/warehouse/h1bproject.db/h1b\_final /H1B\_Project/MapReduce/Output/Q1BTop5JobHighestGrowth

hadoop fs -cat /H1B\_Project/MapReduce/Output/Q1BTop5JobHighestGrowth/p\*

**Output:-**



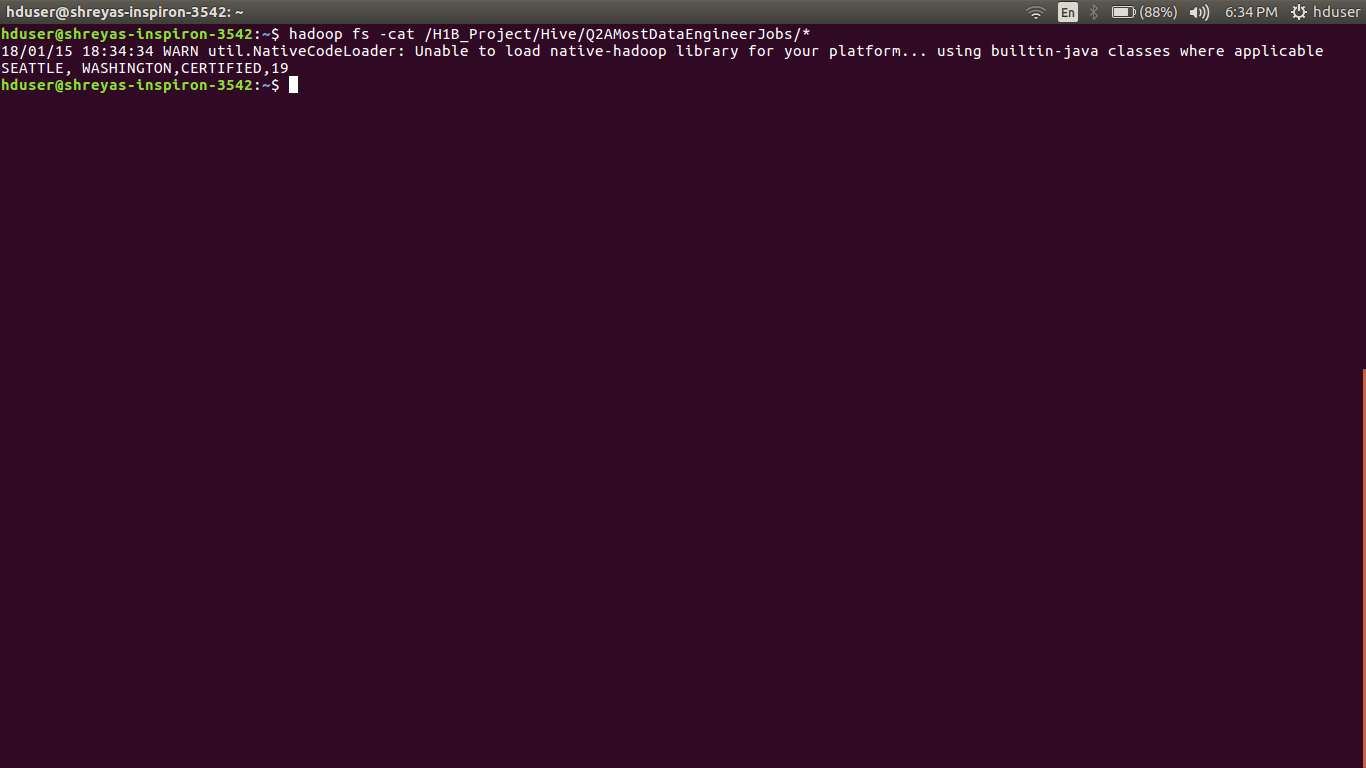
========================================================

**Q.2A) Which part of the US has the most Data Engineer jobs for each year?**

=>

hive -e "select worksite, case\_status,count(\*) as jobs from h1bproject.h1b\_final where job\_title like '%DATA ENGINEER%' and case\_status='CERTIFIED' and year='2011' group by worksite,year,case\_status order by jobs desc limit 1;" (Same Query For other Year Also.)

**Output:-**

****

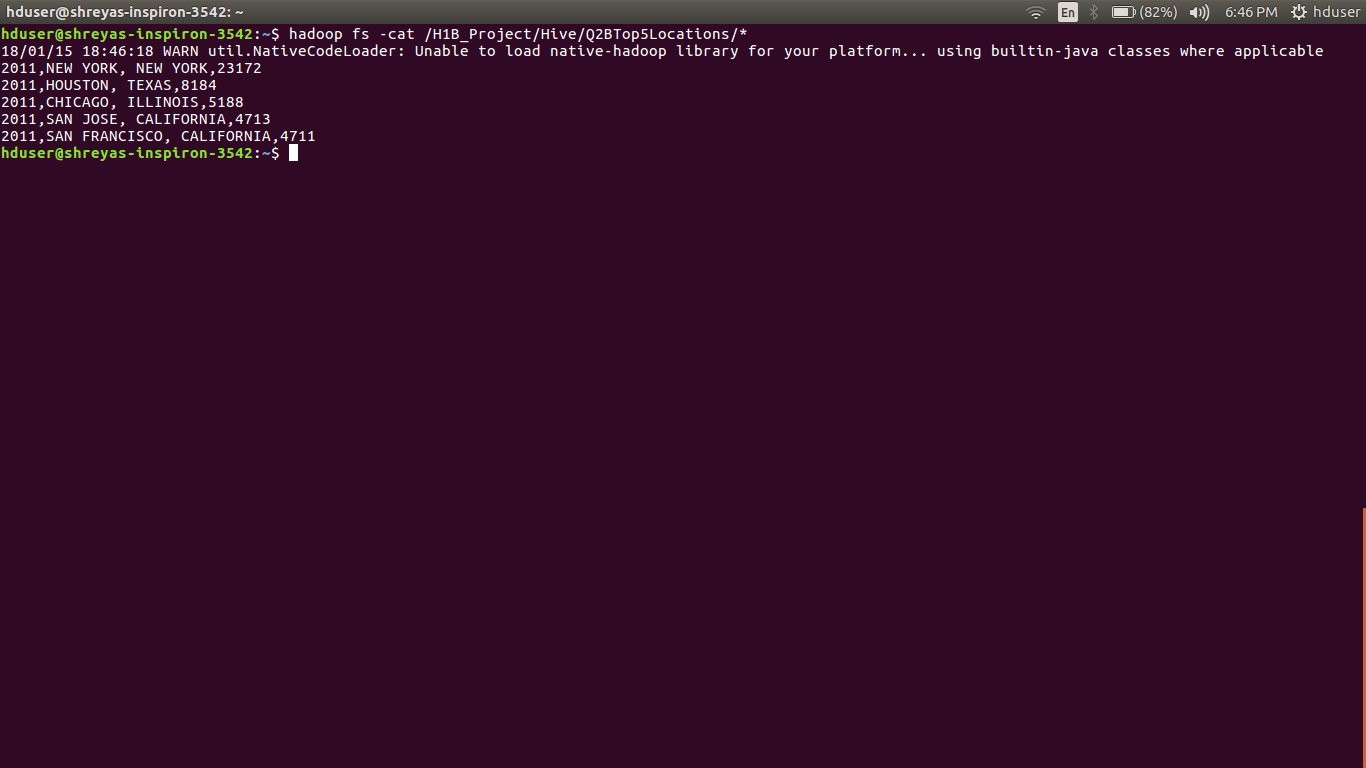
========================================================

**Q.2B) find top 5 locations in the US who have got certified visa for each year.[certified]**

=>

hive -e "select year, worksite,count(\*) as status from h1bproject.h1b\_final where case\_status='CERTIFIED' and year='2011' group by worksite,year order by status desc limit 5;" (Same Query For Other Years Also)

**Output:-**



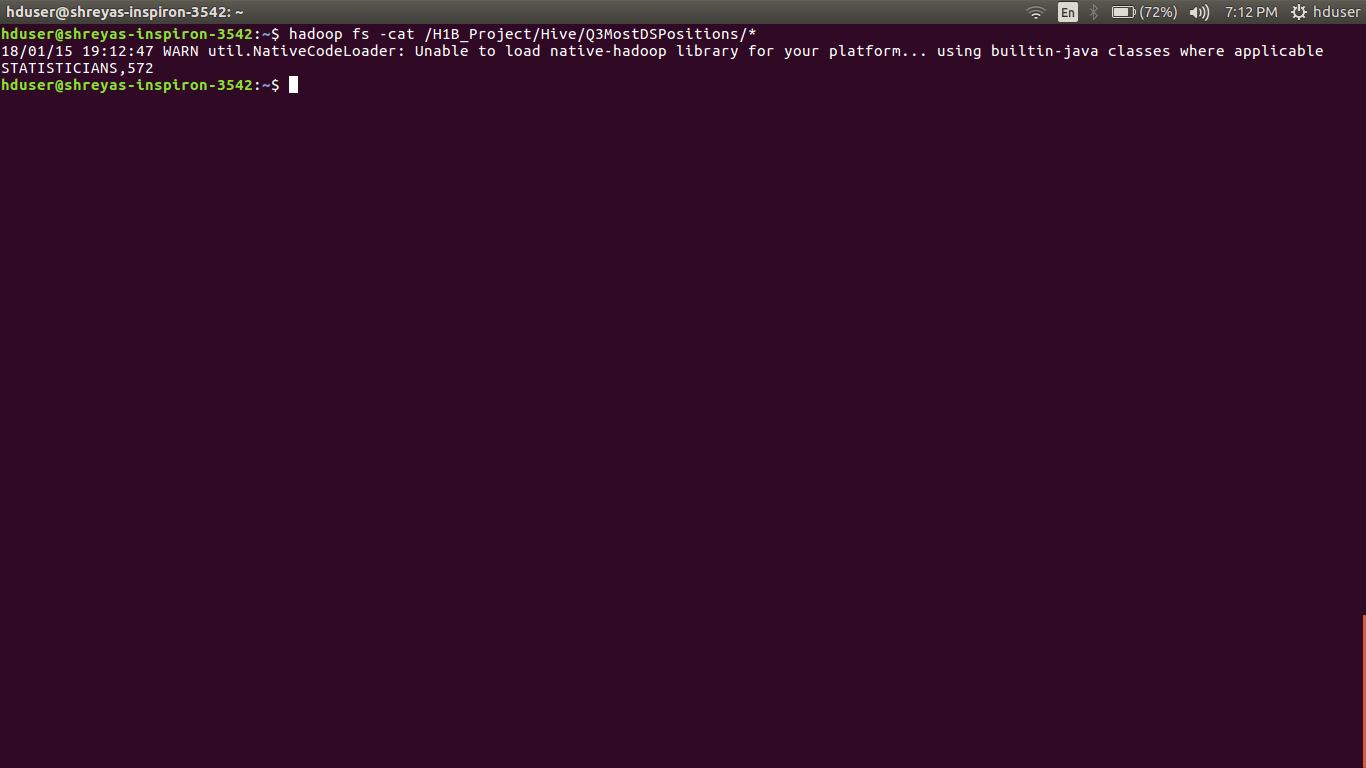
========================================================

**Q.3) Which industry(SOC\_NAME) has the most number of Data Scientist positions?[certified]**

=>

hive -e "select soc\_name, count(\*) as dscount from h1bproject.h1b\_final where job\_title like '%DATA SCIENTIST%' and case\_status='CERTIFIED' group by soc\_name order by dscount desc limit 1;"

**Output:-**

****

========================================================

**Q.4) Which top 5 employers file the most petitions each year? - Case Status – ALL**

=>

import java.io.IOException;

import java.util.TreeMap;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.NullWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Partitioner;

import org.apache.hadoop.mapreduce.Reducer;

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

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

public class Q4\_Top5Employers {

//Mapper Class

public static class EmployeeMapper extends Mapper<LongWritable, Text, Text, Text>

{

LongWritable one = new LongWritable(1);

@Override

protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException

{

String mySearchText = context.getConfiguration().get("myText");

String[] record = value.toString().split("\t");

String year = record[7];

String employee\_name = record[2];

String myVal = year+ ","+one;

if(mySearchText.equals("ALL"))

{

context.write(new Text(employee\_name), new Text(myVal));

}

else

{

if(year.equals(mySearchText))

{

context.write(new Text(employee\_name), new Text(myVal));

}

}

}

}

//Partitioner Class

public static class YearPartitioner extends Partitioner<Text, Text>

{

@Override

public int getPartition(Text key, Text value, int numReduceTasks) {

String[] str = value.toString().split(",");

String year = str[0];

if(year.equals("2011"))

{

return 0 % numReduceTasks;

}

else if(year.equals("2012"))

{

return 1 % numReduceTasks;

}

else if(year.equals("2013"))

{

return 2 % numReduceTasks;

}

else if(year.equals("2014"))

{

return 3 % numReduceTasks;

}

else if(year.equals("2015"))

{

return 4 % numReduceTasks;

}

else

{

return 5 % numReduceTasks;

}

}

}

//Reducer Class

public static class EmployeeReducer extends Reducer<Text, Text, NullWritable, Text>

{

TreeMap<Integer, Text> map = new TreeMap<Integer,Text>();

@Override

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

{

int sum = 0;

String year = "";

for(Text val : values)

{

String[] token = val.toString().split(",");

year = token[0];

int count = Integer.parseInt(token[1]);

sum += count;

}

String employee\_name = key.toString();

String myValue = year+ ","+employee\_name + "," + sum;

map.put(new Integer(sum), new Text(myValue));

if(map.size() > 5)

{

map.remove(map.firstKey());

}

}

@Override

protected void cleanup(Context context)throws IOException, InterruptedException

{

for(Text top5 : map.descendingMap().values())

{

context.write(NullWritable.get(), top5);

}

}

}

//Main Method

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

Configuration conf = new Configuration();

if(args.length > 2)

{

conf.set("myText", args[2]);

}

Job job = Job.getInstance(conf, "Top 5 Employers file the most Petitions for each Year");

job.setJarByClass(Q4\_Top5Employers.class);

job.setMapperClass(EmployeeMapper.class);

if(args[2].equals("ALL"))

{

job.setPartitionerClass(YearPartitioner.class);

job.setNumReduceTasks(6);

}

job.setReducerClass(EmployeeReducer.class);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(Text.class);

job.setOutputKeyClass(NullWritable.class);

job.setOutputValueClass(Text.class);

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

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

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

}

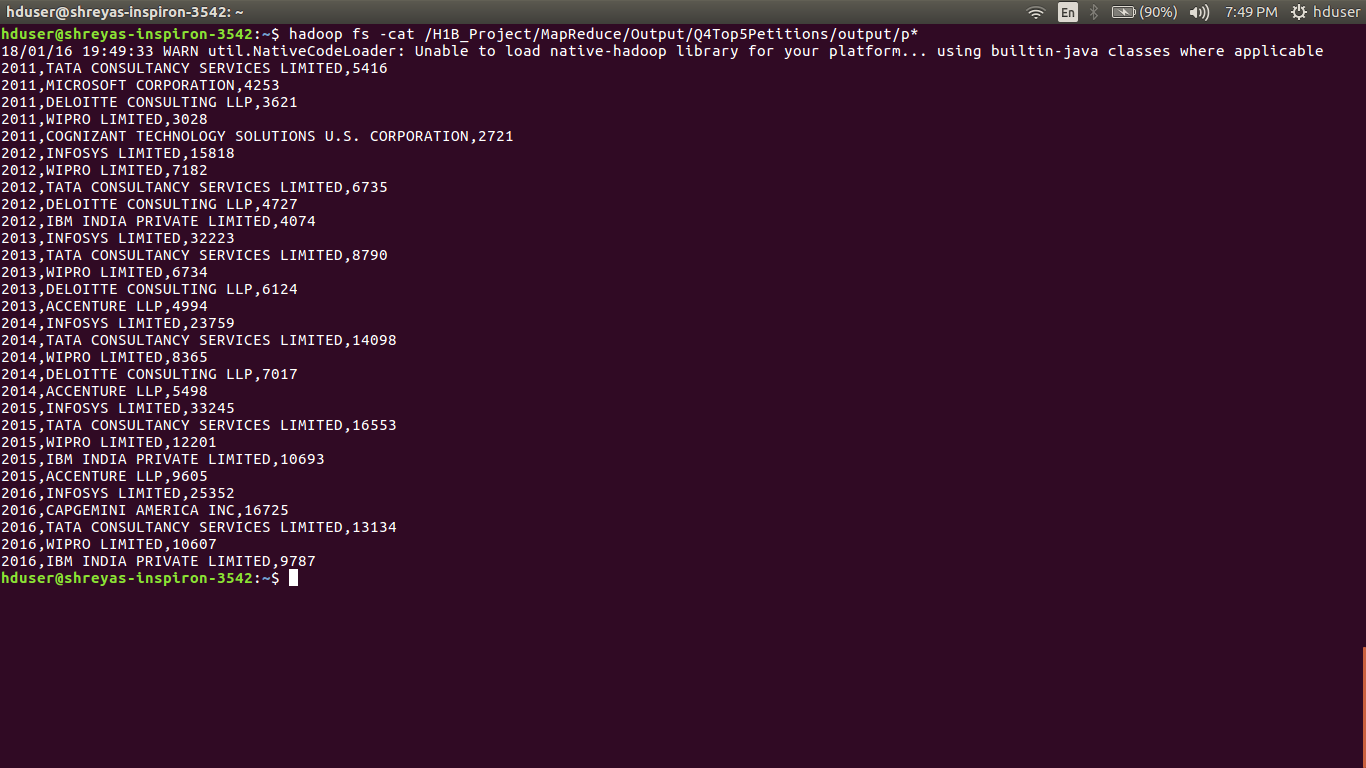
}

**Command:-**

hadoop jar h1bproject.jar Q4\_Top5Employers /user/hive/warehouse/h1bproject.db/h1b\_final /H1B\_Project/MapReduce/Output/Q4Top5Petitions/output1 ALL

hadoop fs -cat /H1B\_Project/MapReduce/Output/Q4Top5Petitions/output1/p\*

**Output:-**

****

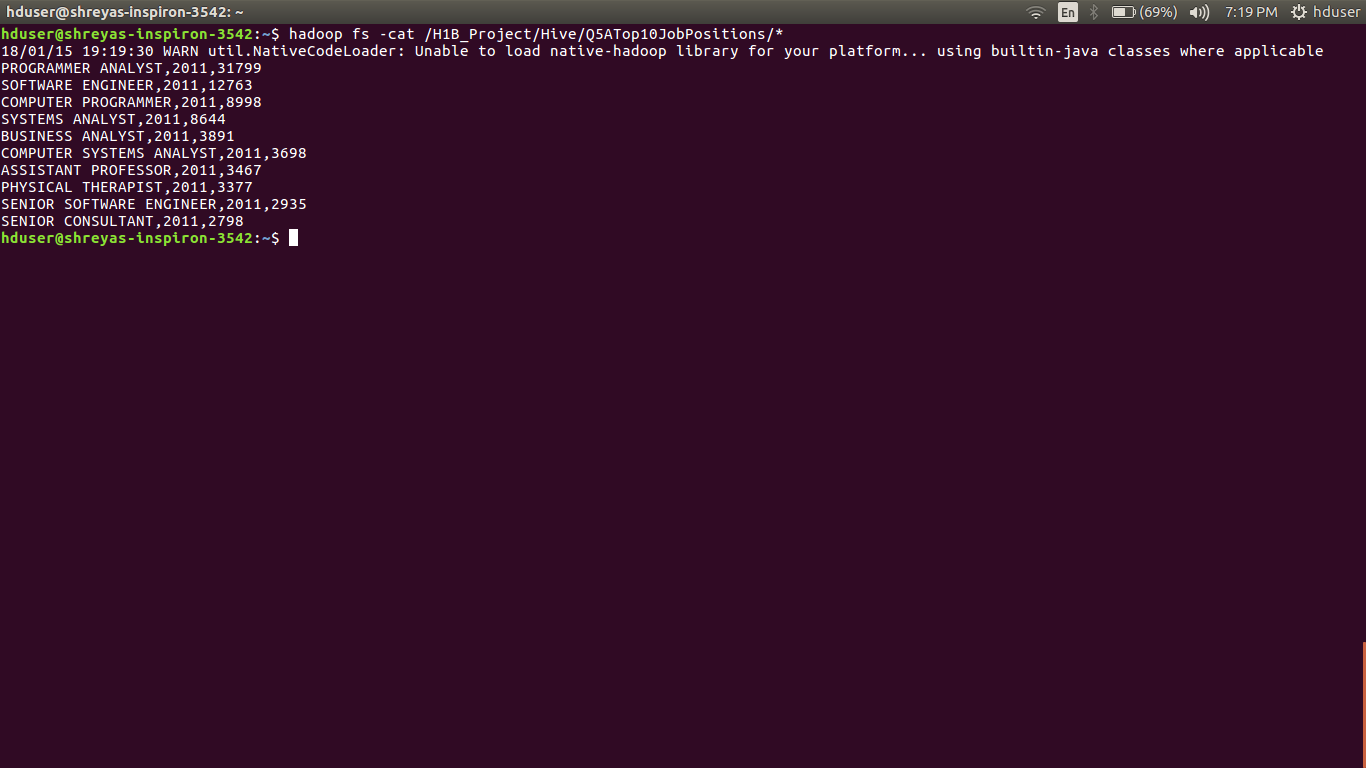
========================================================

**5) Find the most popular top 10 job positions for H1B visa applications for each year?**

**A) for all the applications:-**

hive -e "select job\_title, year,count(\*) as total from h1bproject.h1b\_final where year = '2011' group by job\_title,year order by total desc limit 10;"

**Output:-**

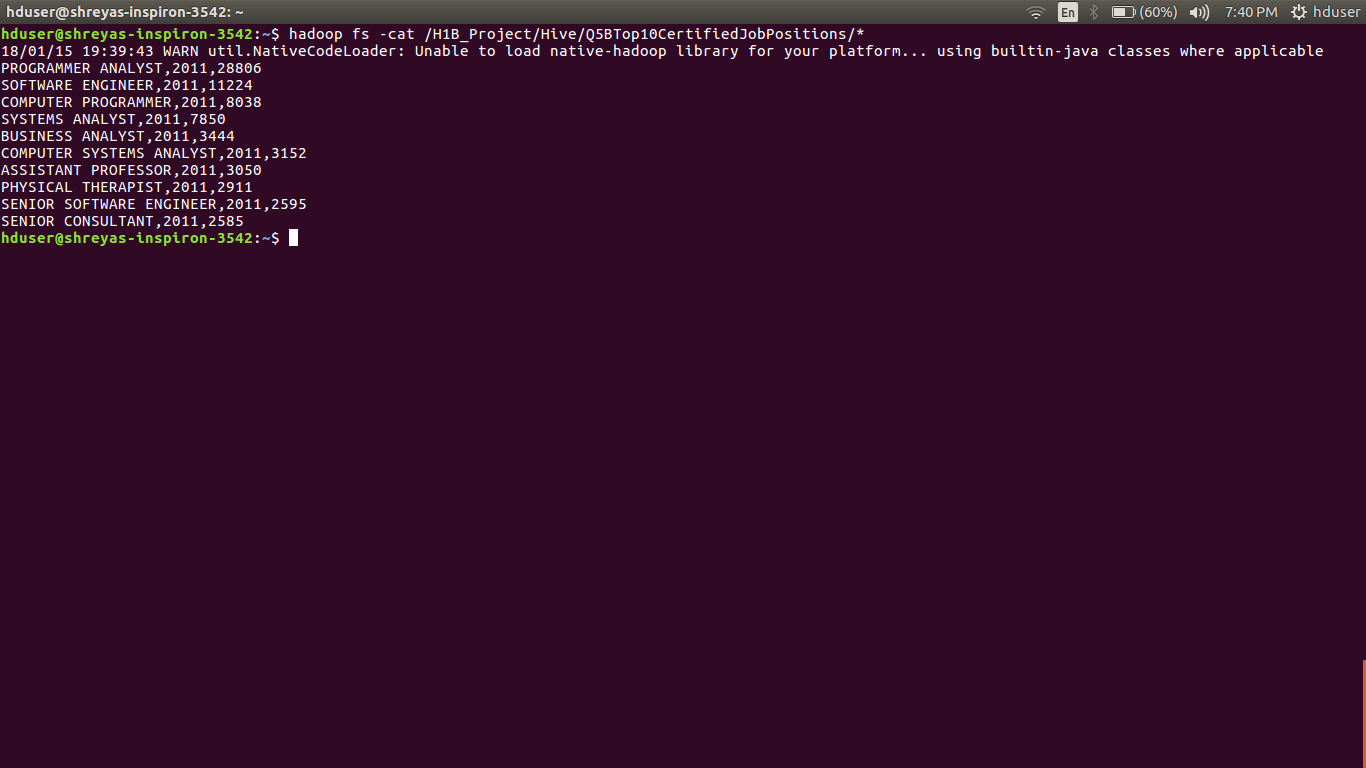
****

----------------------------------------------------------------------------------------------------

**B) for only certified applications:-**

hive -e "select job\_title, year,count(\*) as total from h1bproject.h1b\_final where year = '2011' and case\_status = 'CERTIFIED' group by job\_title,year order by total desc limit 10;"

**Output:-**

****

========================================================

**6) Find the percentage and the count of each case status on total applications for each year. Create a line graph depicting the pattern of All the cases over the period of time.**

=>

h1b = LOAD '/user/hive/warehouse/h1bproject.db/h1b\_final' USING PigStorage() AS

(s\_no,case\_status,employer\_name,soc\_name,job\_title:chararray,full\_time\_position,prevailing\_wage:long,year:chararray,

worksite:chararray,longitute:double,latitute:double);

applications = FOREACH h1b GENERATE case\_status, year;

grp1 = GROUP applications BY year;

count1 = FOREACH grp1 GENERATE group as year, COUNT(applications);

status = GROUP applications BY (year,case\_status);

count2 = FOREACH status GENERATE group, group.year, COUNT(applications);

joined = JOIN count2 BY $1,count1 BY $0;

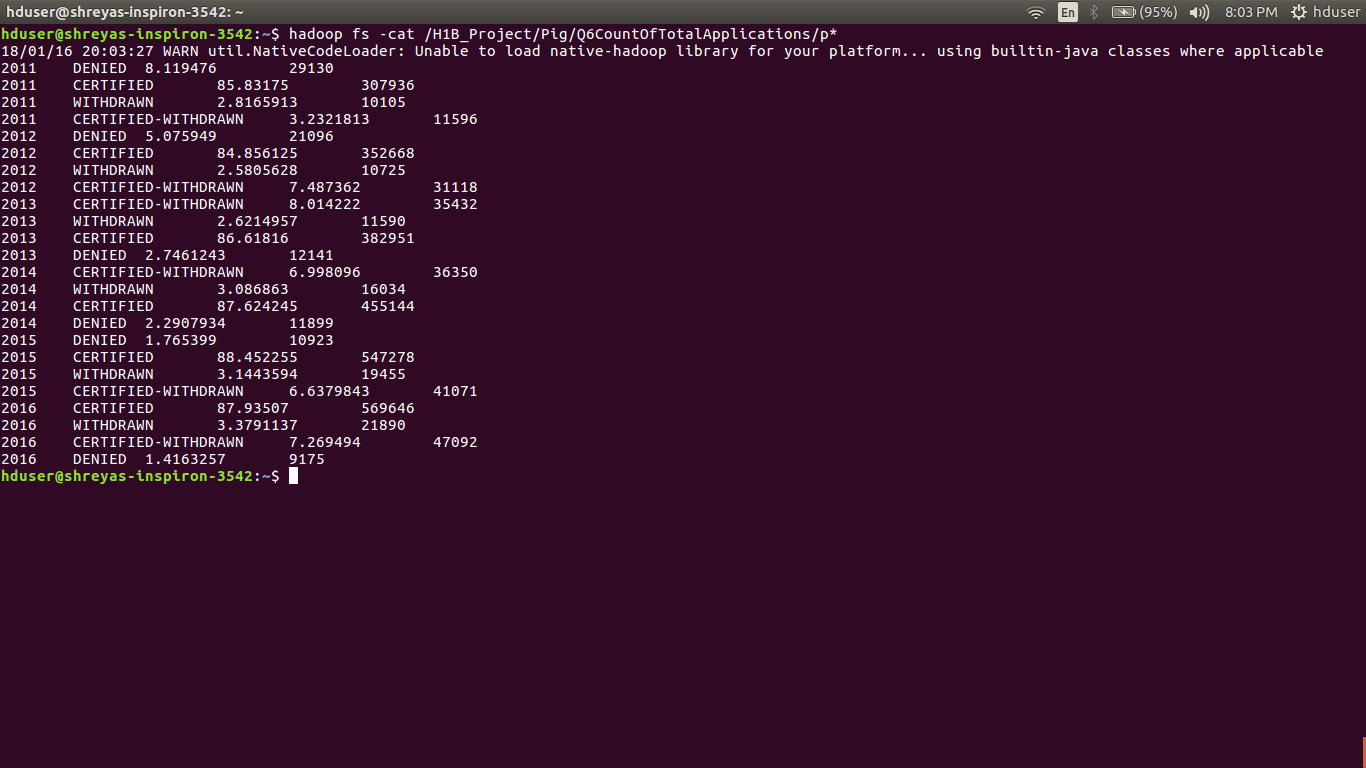
totalapplications = FOREACH joined GENERATE FLATTEN($0), (FLOAT)($2\*100)/$4, $2;

DUMP totalapplications;

**Command:-**

pig /home/hduser/Desktop/H1B-VisaProject/Pig/Q6\_CountOfTotalApplications.pig;

**Output:-**

****

**Graph:-**

========================================================

**Q.7) Create a bar graph to depict the number of applications for each year [All]**

=>h1b = LOAD '/user/hive/warehouse/h1bproject.db/h1b\_final' USING PigStorage() AS

(s\_no,case\_status,employer\_name,soc\_name,job\_title:chararray,full\_time\_position,prevailing\_wage:long,year:chararray,

worksite:chararray,longitute:double,latitute:double);

applications = FOREACH h1b GENERATE case\_status, year;

grp1 = GROUP applications BY year;

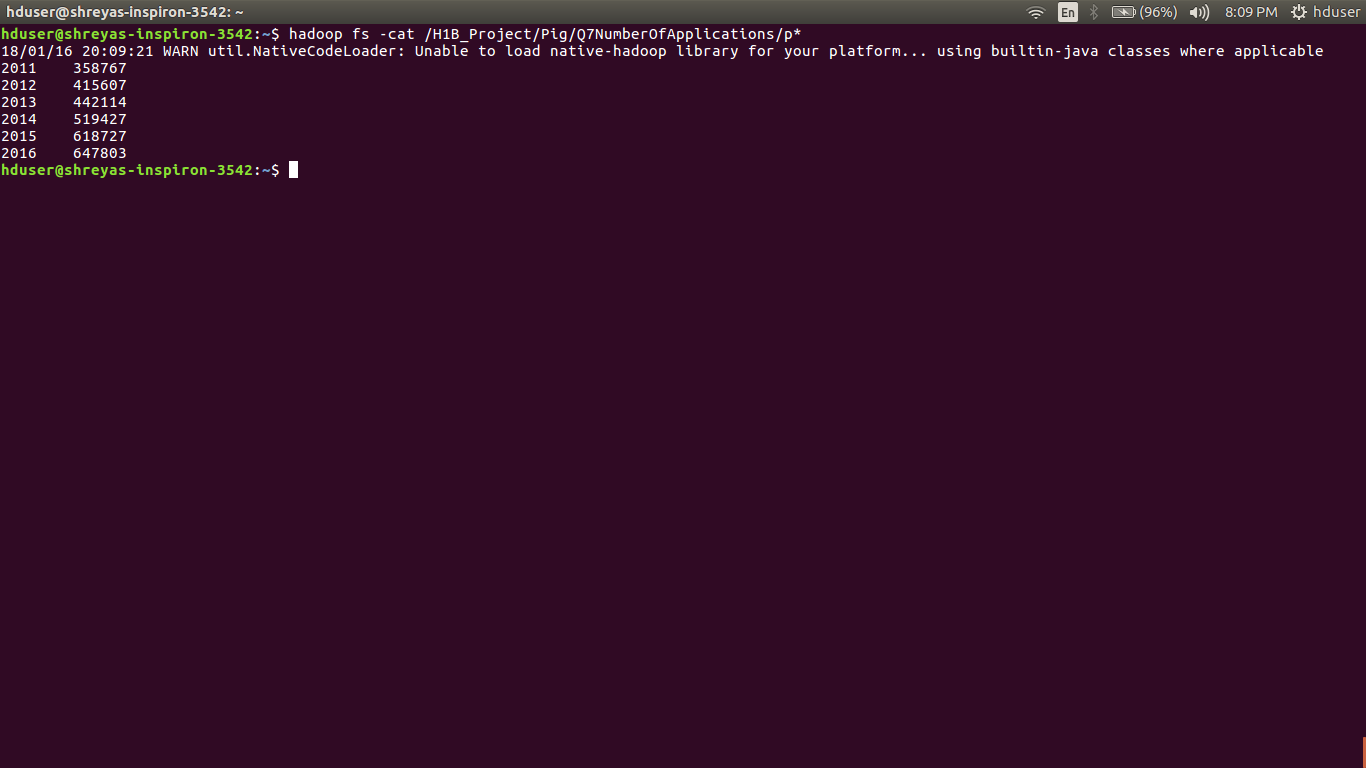
totalapplication = FOREACH grp1 GENERATE group as year, COUNT(applications);

DUMP totalapplication;

**Command:-**

Pig /home/hduser/Desktop/H1B-VisaProject/Pig/Q7\_NumberOfApplications.pig;

**Output:-**

****

**Graph:-**

========================================================

**8) Find the average Prevailing Wage for each Job for each Year (take part time and full time separate). Arrange the output in descending order - [Certified and Certified Withdrawn.]**

=> **A] Full Time:-**

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.DoubleWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.LongWritable.DecreasingComparator;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Partitioner;

import org.apache.hadoop.mapreduce.Reducer;

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

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

public class Q8A\_AvgPrWageForEachJobForEachYearFullTime {

public static class AvgPreWageMapper extends Mapper<LongWritable, Text, Text, Text>

{

@Override

protected void map(LongWritable key, Text value, Context context)throws IOException, InterruptedException

{

String mySearchText = context.getConfiguration().get("myText");

String[] record = value.toString().split("\t");

String case\_status = record[1];

String year = record[7];

String job\_title = record[4];

String full\_time\_position = record[5];

String prevailing\_wage = record[6];

if(mySearchText.equals("ALL"))

{

if(full\_time\_position.equals("Y"))

{

if(case\_status.equals("CERTIFIED") || case\_status.equals("CERTIFIED-WITHDRAWN"))

{

String myValue = year+"\t"+prevailing\_wage;

context.write(new Text(job\_title), new Text(myValue));

}

}

}

else

{

if(full\_time\_position.equals("Y") && year.equals(mySearchText))

{

if(case\_status.equals("CERTIFIED") || case\_status.equals("CERTIFIED-WITHDRAWN"))

{

String myValue = year+"\t"+prevailing\_wage;

context.write(new Text(job\_title), new Text(myValue));

}

}

}

}

}

public static class YearPartitioner extends Partitioner<Text, Text>

{

@Override

public int getPartition(Text key, Text value, int numReduceTasks) {

String[] token = value.toString().split("\t");

String year = token[0];

if(year.equals("2011"))

{

return 0 % numReduceTasks;

}

else if(year.equals("2012"))

{

return 1 % numReduceTasks;

}

else if(year.equals("2013"))

{

return 2 % numReduceTasks;

}

else if(year.equals("2014"))

{

return 3 % numReduceTasks;

}

else if(year.equals("2015"))

{

return 4 % numReduceTasks;

}

else

{

return 5 % numReduceTasks;

}

}

}

public static class AvgPreWageReducer extends Reducer<Text, Text, Text, DoubleWritable>

{

@Override

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

{

long count = 0;

long total = 0;

String year = "";

for (Text val : values)

{

String[] str = val.toString().split("\t");

year = str[0];

long prevailing\_wage = Long.parseLong(str[1]);

count++;

total = total+prevailing\_wage;

}

String job\_title = key.toString();

double average\_prevailing\_wage = total/count;

String myKey = job\_title+"\t"+year;

context.write(new Text(myKey), new DoubleWritable(average\_prevailing\_wage));

}

}

public static class SortMapper extends Mapper<LongWritable, Text, DoubleWritable, Text>

{

@Override

protected void map(LongWritable key, Text value,Context context)throws IOException, InterruptedException

{

String[] record = value.toString().split("\t");

String job\_title = record[0];

String year = record[1];

double average\_prevailing\_wage = Double.parseDouble(record[2]);

String myVal =year+"\t"+job\_title;

context.write(new DoubleWritable(average\_prevailing\_wage), new Text(myVal));

}

}

public static class SortYearPartitioner extends Partitioner<DoubleWritable, Text>

{

@Override

public int getPartition(DoubleWritable key, Text value, int numReduceTasks) {

String[] token = value.toString().split("\t");

String year = token[0];

if(year.equals("2011"))

{

return 0 % numReduceTasks;

}

else if(year.equals("2012"))

{

return 1 % numReduceTasks;

}

else if(year.equals("2013"))

{

return 2 % numReduceTasks;

}

else if(year.equals("2014"))

{

return 3 % numReduceTasks;

}

else if(year.equals("2015"))

{

return 4 % numReduceTasks;

}

else

{

return 5 % numReduceTasks;

}

}

}

public static class SortReducer extends Reducer<DoubleWritable, Text, Text, DoubleWritable>

{

int counter = 0;

@Override

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

{

for(Text val :values)

{

if(counter < 5)

{

context.write(new Text(val), key);

counter= counter+1;

}

}

}

}

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

Configuration conf = new Configuration();

if(args.length > 2)

{

conf.set("myText", args[2]);

}

Job job1 = Job.getInstance(conf, "Average Prevailing Wage for each Job for each Year");

job1.setJarByClass(Q8B\_AvgPrWageForEachJobForEachYearPartTime.class);

job1.setMapperClass(AvgPreWageMapper.class);

if(args[2].equals("ALL"))

{

job1.setPartitionerClass(YearPartitioner.class);

job1.setNumReduceTasks(6);

}

job1.setReducerClass(AvgPreWageReducer.class);

job1.setMapOutputKeyClass(Text.class);

job1.setMapOutputValueClass(Text.class);

job1.setOutputKeyClass(DoubleWritable.class);

job1.setOutputValueClass(Text.class);

Path outputpath1 = new Path("FirstMapper2");

FileInputFormat.addInputPath(job1, new Path(args[0]));

FileOutputFormat.setOutputPath(job1, outputpath1);

FileSystem.get(conf).delete(outputpath1, true);

job1.waitForCompletion(true);

Job job2 = Job.getInstance(conf, "Average Prevailing Wage for each Job for each Year");

job2.setJarByClass(Q8B\_AvgPrWageForEachJobForEachYearPartTime.class);

job2.setMapperClass(SortMapper.class);

if(args[2].equals("ALL"))

{

job2.setPartitionerClass(SortYearPartitioner.class);

job2.setNumReduceTasks(6);

}

job2.setSortComparatorClass(DecreasingComparator.class);

job2.setReducerClass(SortReducer.class);

job2.setMapOutputKeyClass(DoubleWritable.class);

job2.setMapOutputValueClass(Text.class);

job2.setOutputKeyClass(Text.class);

job2.setOutputValueClass(DoubleWritable.class);

FileInputFormat.addInputPath(job2, outputpath1);

FileOutputFormat.setOutputPath(job2, new Path(args[1]));

FileSystem.get(conf).delete(new Path(args[1]), true);

System.exit(job2.waitForCompletion(true) ? 0 : 1);

}

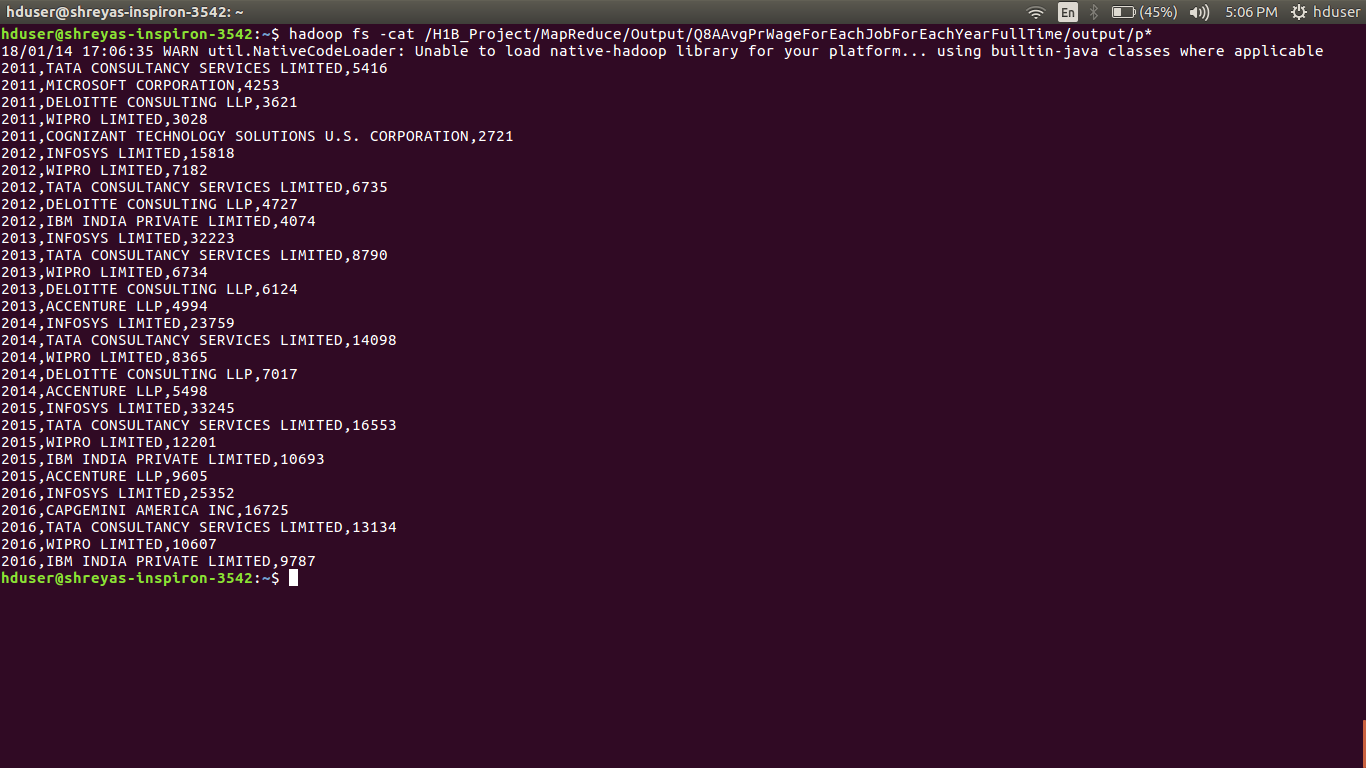
}

**Command:-**

hadoop jar h1bproject.jar Q8A\_AvgPrWageForEachJobForEachYearFullTime /user/hive/warehouse/h1bproject.db/h1b\_final /H1B\_Project/MapReduce/Output/Q8AAvgPrWageForEachJobForEachYearFullTime/output ALL

hadoop fs -cat /H1B\_Project/MapReduce/Output/Q8AAvgPrWageForEachJobForEachYearFullTime/output/p\*

**Output:-**

****

---------------------------------------------------------------------------------------------------

**B) Part Time:-**

=>

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.DoubleWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.io.LongWritable.DecreasingComparator;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Partitioner;

import org.apache.hadoop.mapreduce.Reducer;

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

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

public class Q8B\_AvgPrWageForEachJobForEachYearPartTime {

public static class AvgPreWageMapper extends Mapper<LongWritable, Text, Text, Text>

{

@Override

protected void map(LongWritable key, Text value, Context context)throws IOException, InterruptedException

{

String mySearchText = context.getConfiguration().get("myText");

String[] record = value.toString().split("\t");

String case\_status = record[1];

String year = record[7];

String job\_title = record[4];

String full\_time\_position = record[5];

String prevailing\_wage = record[6];

if(mySearchText.equals("ALL"))

{

if(full\_time\_position.equals("N"))

{

if(case\_status.equals("CERTIFIED") || case\_status.equals("CERTIFIED-WITHDRAWN"))

{

String myValue = year+"\t"+prevailing\_wage;

context.write(new Text(job\_title), new Text(myValue));

}

}

}

else

{

if(full\_time\_position.equals("N") && year.equals(mySearchText))

{

if(case\_status.equals("CERTIFIED") || case\_status.equals("CERTIFIED-WITHDRAWN"))

{

String myValue = year+"\t"+prevailing\_wage;

context.write(new Text(job\_title), new Text(myValue));

}

}

}

}

}

public static class YearPartitioner extends Partitioner<Text, Text>

{

@Override

public int getPartition(Text key, Text value, int numReduceTasks) {

String[] token = value.toString().split("\t");

String year = token[0];

if(year.equals("2011"))

{

return 0 % numReduceTasks;

}

else if(year.equals("2012"))

{

return 1 % numReduceTasks;

}

else if(year.equals("2013"))

{

return 2 % numReduceTasks;

}

else if(year.equals("2014"))

{

return 3 % numReduceTasks;

}

else if(year.equals("2015"))

{

return 4 % numReduceTasks;

}

else

{

return 5 % numReduceTasks;

}

}

}

public static class AvgPreWageReducer extends Reducer<Text, Text, Text, DoubleWritable>

{

@Override

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

{

long count = 0;

long total = 0;

String year = "";

for (Text val : values)

{

String[] str = val.toString().split("\t");

year = str[0];

long prevailing\_wage = Long.parseLong(str[1]);

count++;

total = total+prevailing\_wage;

}

String job\_title = key.toString();

double average\_prevailing\_wage = total/count;

String myKey = job\_title+"\t"+year;

context.write(new Text(myKey), new DoubleWritable(average\_prevailing\_wage));

}

}

public static class SortMapper extends Mapper<LongWritable, Text, DoubleWritable, Text>

{

@Override

protected void map(LongWritable key, Text value,Context context)throws IOException, InterruptedException

{

String[] record = value.toString().split("\t");

String job\_title = record[0];

String year = record[1];

double average\_prevailing\_wage = Double.parseDouble(record[2]);

String myVal =year+"\t"+job\_title;

context.write(new DoubleWritable(average\_prevailing\_wage), new Text(myVal));

}

}

public static class SortYearPartitioner extends Partitioner<DoubleWritable, Text>

{

@Override

public int getPartition(DoubleWritable key, Text value, int numReduceTasks) {

String[] token = value.toString().split("\t");

String year = token[0];

if(year.equals("2011"))

{

return 0 % numReduceTasks;

}

else if(year.equals("2012"))

{

return 1 % numReduceTasks;

}

else if(year.equals("2013"))

{

return 2 % numReduceTasks;

}

else if(year.equals("2014"))

{

return 3 % numReduceTasks;

}

else if(year.equals("2015"))

{

return 4 % numReduceTasks;

}

else

{

return 5 % numReduceTasks;

}

}

}

public static class SortReducer extends Reducer<DoubleWritable, Text, Text, DoubleWritable>

{

int counter = 0;

@Override

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

{

for(Text val :values)

{

if(counter < 5)

{

context.write(new Text(val), key);

counter= counter+1;

}

}

}

}

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

Configuration conf = new Configuration();

if(args.length > 2)

{

conf.set("myText", args[2]);

}

Job job1 = Job.getInstance(conf, "Average Prevailing Wage for Part Time each Job for each Year");

job1.setJarByClass(Q8B\_AvgPrWageForEachJobForEachYearPartTime.class);

job1.setMapperClass(AvgPreWageMapper.class);

if(args[2].equals("ALL"))

{

job1.setPartitionerClass(YearPartitioner.class);

job1.setNumReduceTasks(6);

}

job1.setReducerClass(AvgPreWageReducer.class);

job1.setMapOutputKeyClass(Text.class);

job1.setMapOutputValueClass(Text.class);

job1.setOutputKeyClass(DoubleWritable.class);

job1.setOutputValueClass(Text.class);

Path outputpath1 = new Path("Parttime1");

FileInputFormat.addInputPath(job1, new Path(args[0]));

FileOutputFormat.setOutputPath(job1, outputpath1);

FileSystem.get(conf).delete(outputpath1, true);

job1.waitForCompletion(true);

Job job2 = Job.getInstance(conf, "Average Prevailing Wage for Part Time each Job for each Year");

job2.setJarByClass(Q8B\_AvgPrWageForEachJobForEachYearPartTime.class);

job2.setMapperClass(SortMapper.class);

if(args[2].equals("ALL"))

{

job2.setPartitionerClass(SortYearPartitioner.class);

job2.setNumReduceTasks(6);

}

job2.setSortComparatorClass(DecreasingComparator.class);

job2.setReducerClass(SortReducer.class);

job2.setMapOutputKeyClass(DoubleWritable.class);

job2.setMapOutputValueClass(Text.class);

job2.setOutputKeyClass(Text.class);

job2.setOutputValueClass(DoubleWritable.class);

FileInputFormat.addInputPath(job2, outputpath1);

FileOutputFormat.setOutputPath(job2, new Path(args[1]));

FileSystem.get(conf).delete(new Path(args[1]), true);

System.exit(job2.waitForCompletion(true) ? 0 : 1);

}

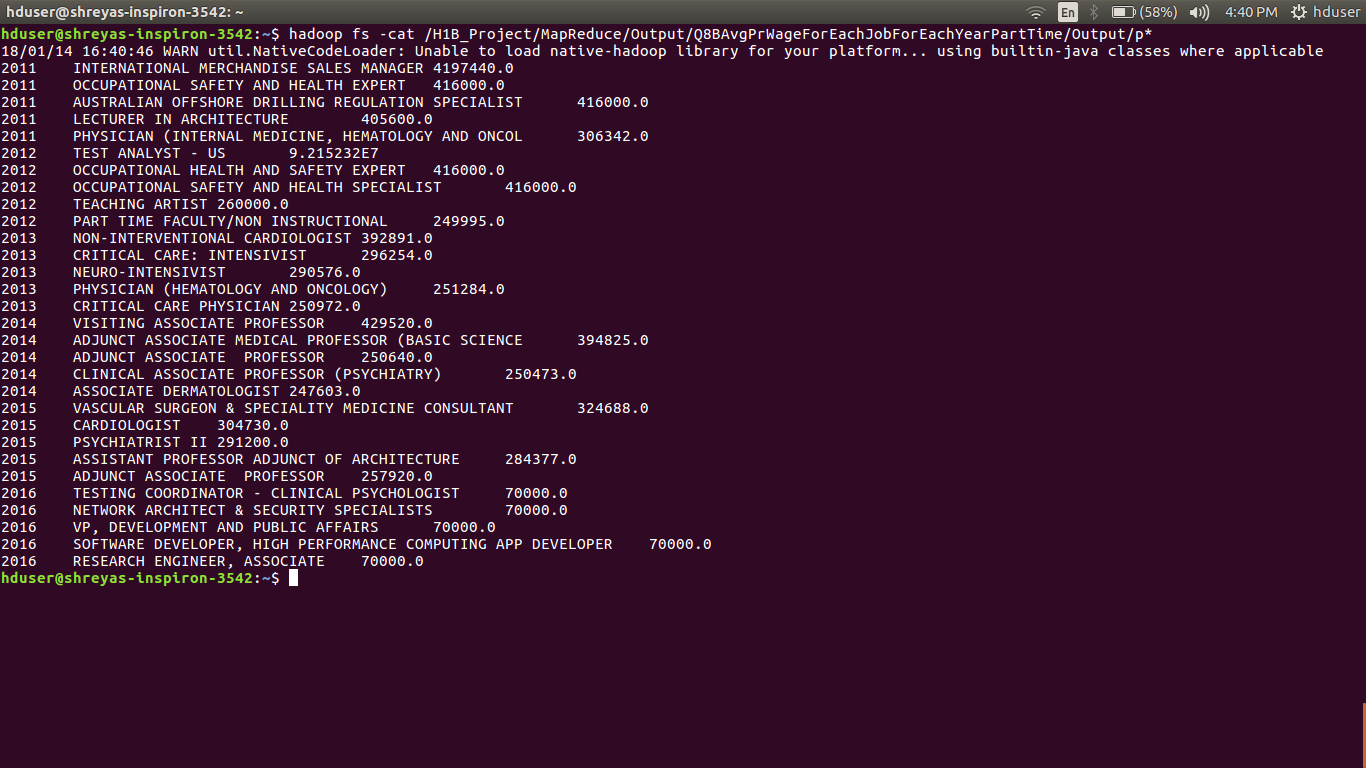
}

**Command:-**

hadoop jar h1bproject.jar Q8B\_AvgPrWageForEachJobForEachYearPartTime /user/hive/warehouse/h1bproject.db/h1b\_final /H1B\_Project/MapReduce/Output/Q8BAvgPrWageForEachJobForEachYearPartTime/Output

hadoop fs -cat /H1B\_Project/MapReduce/Output/Q8BAvgPrWageForEachJobForEachYearPartTime/Output/p\*

**Output:-**



========================================================

**Q.9) Which are the employers along with the number of petitions who have the success rate more than 70% in petitions. (total petitions filed 1000 OR more than 1000) ?**

=>

h1b = LOAD '/user/hive/warehouse/h1bproject.db/h1b\_final' USING PigStorage() AS

(s\_no,case\_status,employer\_name,soc\_name,job\_title:chararray,full\_time\_position,prevailing\_wage:long,year:chararray,worksite:chararray,longitute:double,latitute:double);

finalh1b = FOREACH h1b GENERATE case\_status,employer\_name;

allgrouped = GROUP finalh1b BY employer\_name;

allcount = FOREACH allgrouped GENERATE group as employer\_name,COUNT(finalh1b.case\_status) as totalapplicaion;

filterh1b = FILTER finalh1b BY case\_status == 'CERTIFIED-WITHDRAWN' OR case\_status == 'CERTIFIED';

successgrouped = GROUP filterh1b BY employer\_name;

successcount = FOREACH successgrouped GENERATE group AS employer\_name,COUNT(filterh1b.case\_status) as totalsuccess;

joined = JOIN allcount BY $0,successcount BY $0;

finalbag = FOREACH joined GENERATE $0,$1 as petitions, (FLOAT)(($3\*100)/$1) AS successrate;

filtersuccessrate = FILTER finalbag BY $1 >= 1000 AND $2 > 70;

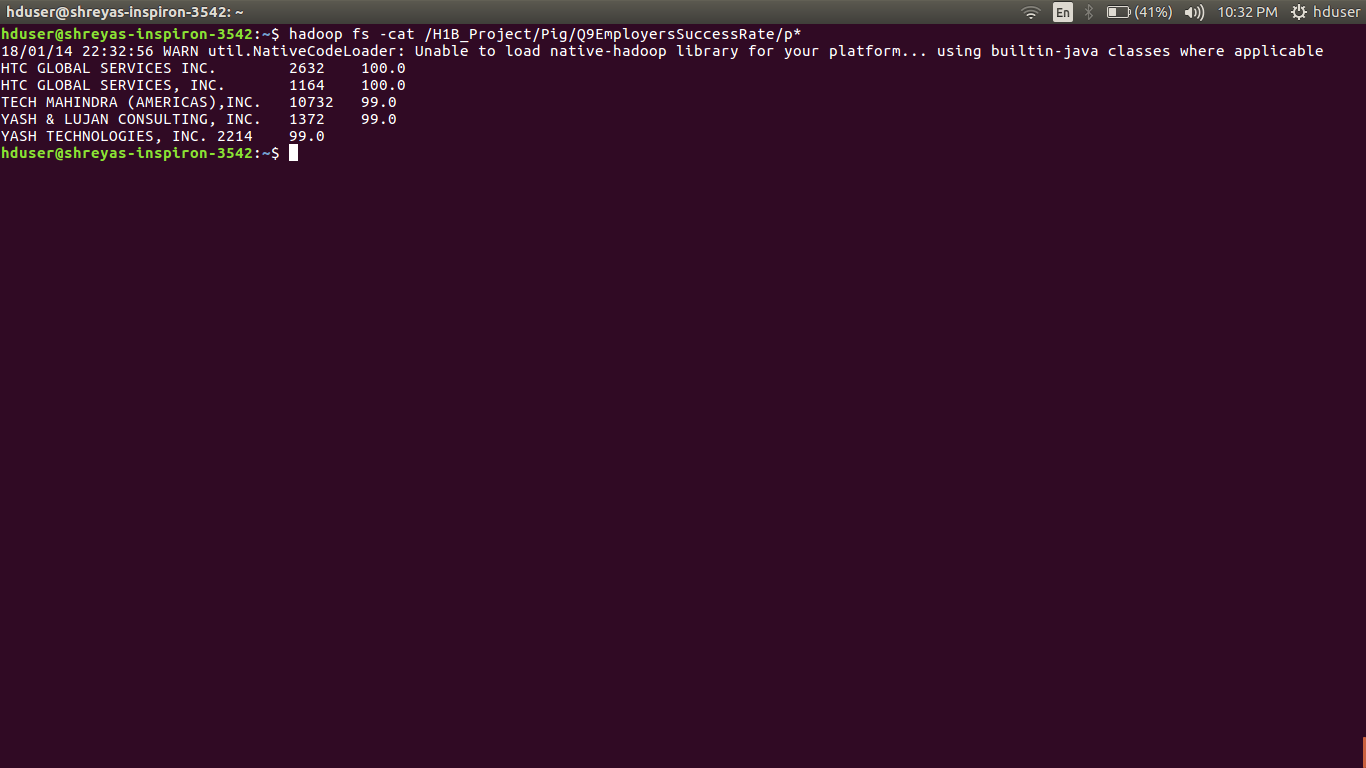
successrate= LIMIT (ORDER filtersuccessrate BY $2 DESC)5;

DUMP successrate;

**Command:-**

pig /home/hduser/Desktop/H1B- VisaProject/Pig/Q9\_EmployersSuccessRate.pig

**Output:-**



=========================================================

**10)** **Which are the job positions along with the number of petitions which have the success rate more than 70% in petitions (total petitions filed 1000 OR more than 1000)?**

=>

h1b = LOAD '/user/hive/warehouse/h1bproject.db/h1b\_final' USING PigStorage() AS

(s\_no,case\_status,employer\_name,soc\_name,job\_title:chararray,full\_time\_position,prevailing\_wage:long,year:chararray,worksite:chararray,longitute:double,latitute:double);

finalh1b = FOREACH h1b GENERATE case\_status,job\_title;

allgrouped = GROUP finalh1b BY job\_title;

allcount = FOREACH allgrouped GENERATE group as job\_title,COUNT(finalh1b.case\_status) as totalapplicaion;

filterh1b = FILTER finalh1b BY case\_status == 'CERTIFIED-WITHDRAWN' OR case\_status == 'CERTIFIED';

successgrouped = GROUP filterh1b BY job\_title;

successcount = FOREACH successgrouped GENERATE group AS job\_title,COUNT(filterh1b.case\_status) as totalsuccess;

joined = JOIN allcount BY $0,successcount BY $0;

finalbag = FOREACH joined GENERATE $0,$1 as petitions, (FLOAT)(($3\*100)/$1) AS successrate;

filtersuccessrate = FILTER finalbag BY $1 >= 1000 AND $2 > 70;

jobsuccessrate= LIMIT (ORDER filtersuccessrate BY $2 DESC)5;

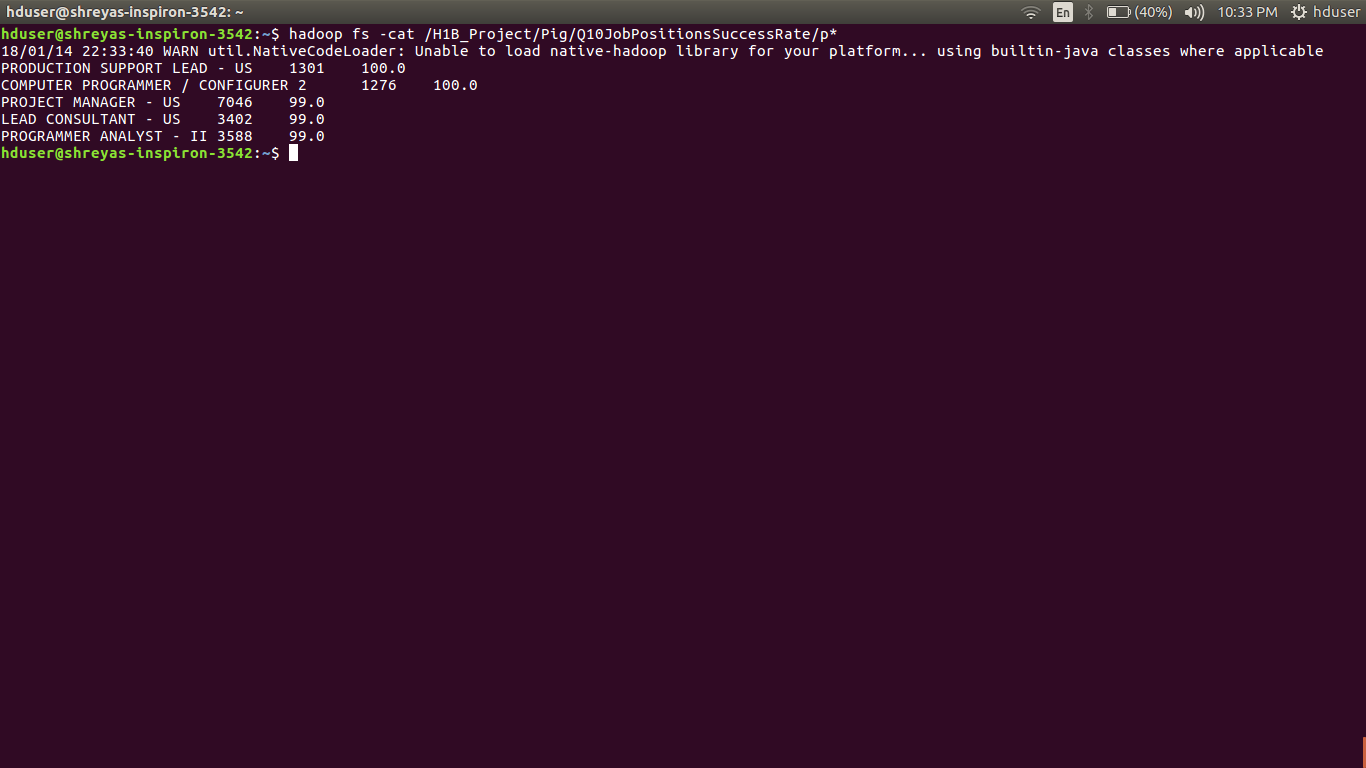
DUMP jobsuccessrate;

**Command:-**

pig /home/hduser/Desktop/H1B-

VisaProject/Pig/Q10\_JobPositionsSucessRate.pig

**Output:-**

****

========================================================

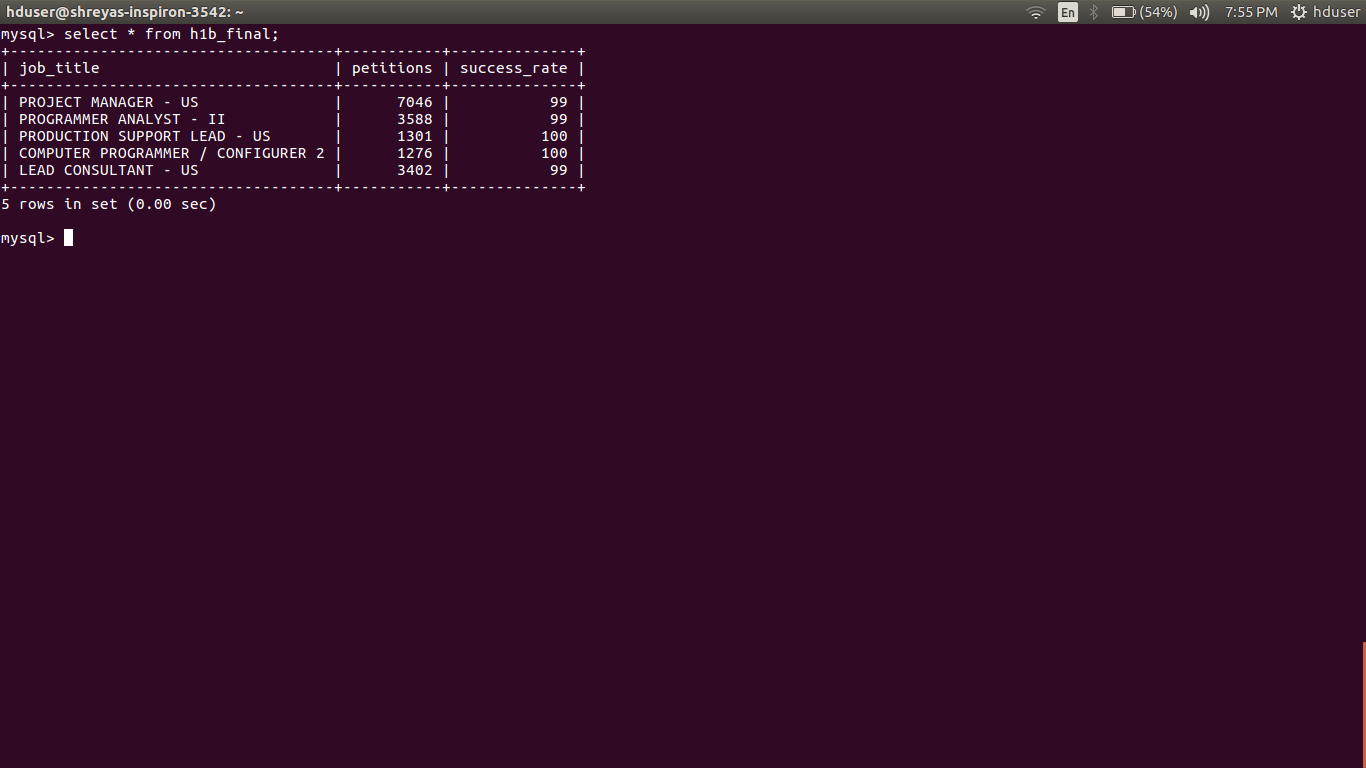
**Q.11) Export result for question no 10 to MySql database.**

=>

sqoop export --connect jdbc:mysql://localhost/h1b\_project --username root --password root --table h1b\_final --update-mode allowinsert --export-dir /H1B\_Project/Pig/Q10JobPositionsSuccessRate/part-r-00000 --input-fields-terminated-by '\t' ;

mysql –u root –p’root’

**Output:-**

****

================================================================================================================

**CONCLUSION**

Hadoop has been very effective solution for companies dealing with the data in perabytes.It has solved many problems in industry related to huge data management and distributed system.As it is open source, so it is adopted by companies widely.

There are various ways for Hadoop to run the Job. The three programming approaches that are MapReduce, Hive and Pig are used. MapReduce, highly recommendable to process whole data, its time consume and required project aptitudes like Java etc. Total data aggregates and sorts by utilizing mapper and reducer capacities. Hadoop uses it by default. An alternative to MapReduce, Hive and Pig were built so that developers could do the same thing in a less complex way by writing only fewer lines of code that will be very easy to understand. It does not require learning concepts of Any Programming Language.

Performing practically, I conclude that Pig takes less time as compared to both MapReduce and Hive. Complexity of MapReduce, Hive and Pig is High, Low and Low computational complexity respectively. Code efficiency for MapReduce, Hive and Pig is High, Relatively less and Relatively less respectively.

All approaches have pros and cons so we have to choose according to our need and data.