BIG DATA PROJECT ANALYSIS DOCUMENTATION

INTRODUCTION

"90% of the world's data was generated in the last few years."Due to the advent of new technologies, devices, and communication means like social networking sites, the amount of data produced by mankind is growing rapidly every year. The amount of data produced by us from the beginning of time till 2003 was 5 billion gigabytes. If you pile up the data in the form of disks it may fill an entire football field. The same amount was created in every two days in 2011, and in every ten minutes in 2013. This rate is still growing enormously. Though all this information produced is meaningful and can be useful when processed, it is being neglected.

What is Big Data?

Big Data' is a term used to describe collection of data that is huge in size and yet growing exponentially with time. In short, such a data is so large and complex that none of the traditional data management tools are able to store it or process it efficiently.

Examples Of 'Big Data'

Following are some the examples of 'Big Data'-

The **New York Stock Exchange** generates about ***one terabyte*** of new trade data per day.

Social Media Impact

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.

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**

Structured

Any data that can be stored, accessed and processed in the form of fixed format is termed as a 'structured' data.

Examples Of Structured Data

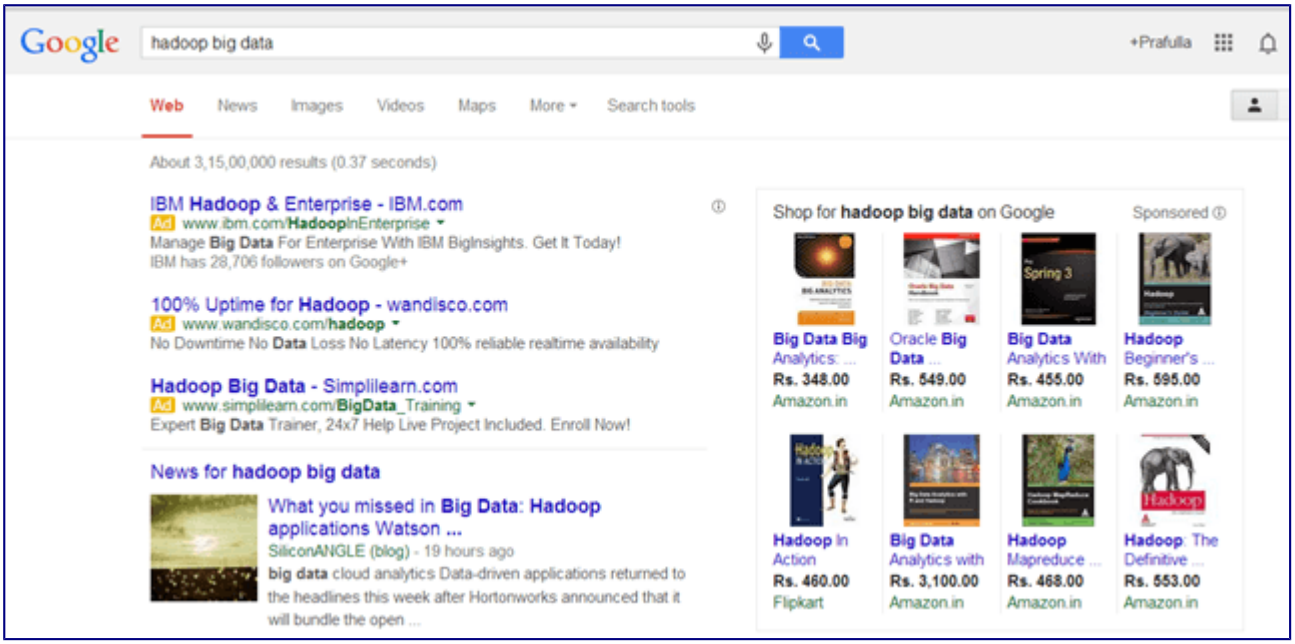
Employee_ID	Employee_Name	Gender	Department	Salary_In_lacs
2365	Rajesh Kulkarni	Male	Finance	650000
3398	Pratibha Joshi	Female	Admin	650000
7465	Shushil Roy	Male	Admin	500000
7500	Shubhojit Das	Male	Finance	500000

Unstructured

Any data with unknown form or the structure is classified as unstructured data.ypical example of unstructured data is, a heterogeneous data source containing a combination of simple text files, images, videos etc.

Examples Of Un-structured Data

Output returned by 'Google Search'



Semi-structured

Semi-structured data can contain both the forms of data. We can see semi-structured data as a strcutured 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>

Characteristics Of 'Big Data'

***(i)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'.

*(ii)**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 analysing data.

*(iii)**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.

*(iv)**Variability*** – This refers to the inconsistency which can be shown by the data at times, thus hampering the process of being able to handle and manage the data effectively.

v)**Veracity** - The quality of the data being captured can vary greatly. Accuracy of analysis depends

on the veracity of the source data.

Big Data Challenges

The major challenges associated with big data are as follows:

- Capturing data Sharing
- Transfer
- Analysis
- Presentation

Hadoop

Hadoop is an Apache open source framework written in java that allows distributed processing of large datasets across clusters of computers using simple programming models. A Hadoop frame-worked application works in an environment that provides distributed storage and computation across clusters of computers. Hadoop is designed to scale up from single server to thousands of machines, each offering local computation and storage. Doug Cutting, Mike Cafarella and team took the solution provided by Google and started an Open Source Project called HADOOP in 2005 and Doug named it after his son's toy elephant. Now Apache Hadoop is a registered trademark of the Apache Software Foundation.

Similar to data residing in a local file system of personal computer system, in Hadoop, data resides in a distributed file system which is called as a **Hadoop Distributed File system**. Processing model is based on **'Data Locality'** concept wherein computational logic is sent to cluster nodes(server) containing data. This computational logic is nothing but a compiled version of a program written in a high level language such as Java. Such a program, processes data stored in Hadoop HDFS.

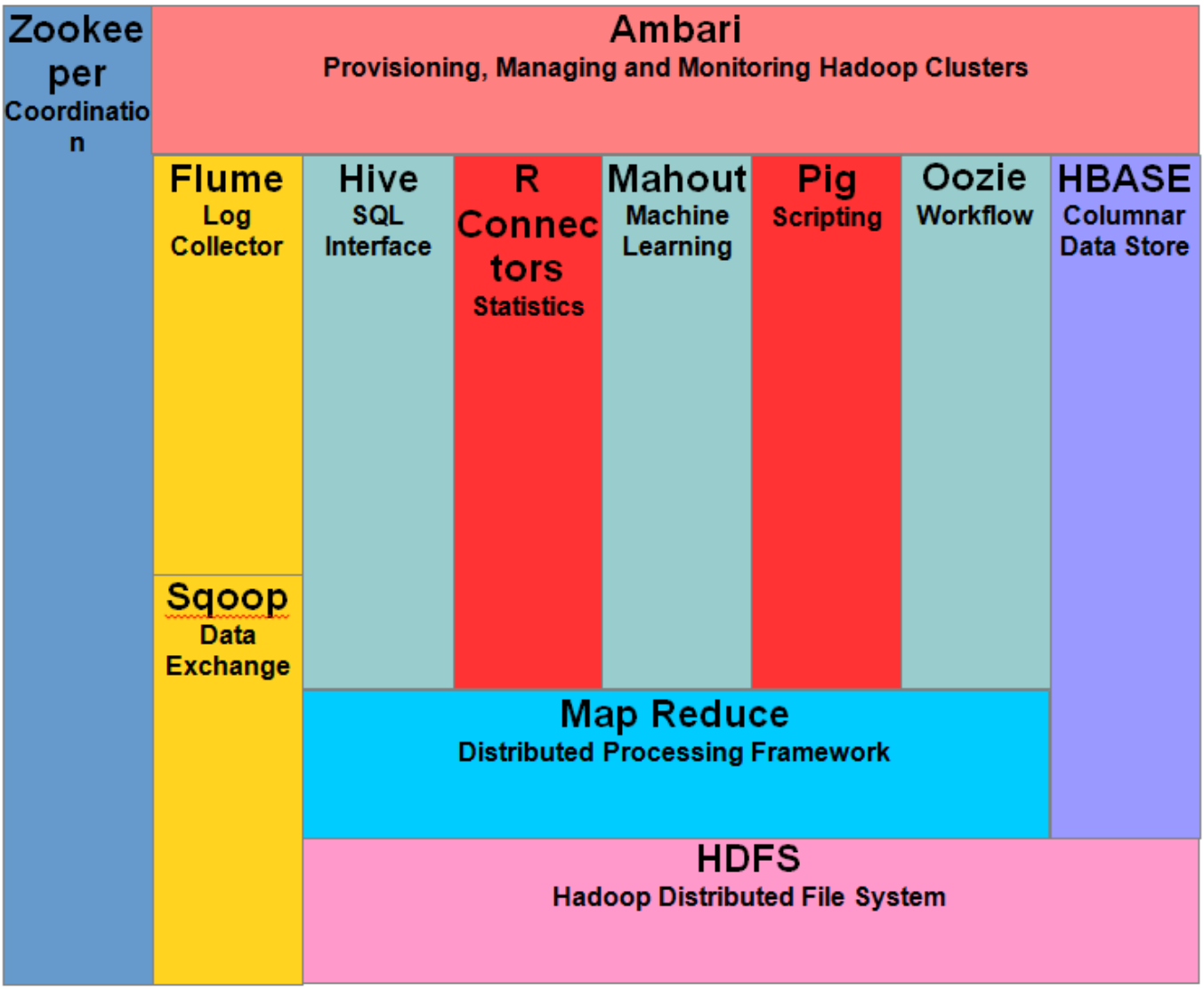
HADOOP is an open source software framework. Applications built using HADOOP are run on large data sets distributed across clusters of commodity computers. Commodity computers are cheap and widely available. These are mainly useful for achieving greater computational power at low cost.

Hadoop Architecture

Hadoop framework includes following four modules:

- **Hadoop Common:** These are Java libraries and utilities required by other Hadoop modules. These libraries provides filesystem 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.



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 include are **Hive**, **HBase**, **Mahout**, **Sqoop** , **Flume** and **ZooKeeper**.

Features Of 'Hadoop'

• Suitable for Big Data Analysis

As Big Data tends to be distributed and unstructured in nature, HADOOP clusters are best suited for analysis of Big Data. Since, it is processing logic (not the actual data) that flows to the computing nodes, less network bandwidth is consumed. This concept is called as **data locality concept** which helps increase efficiency of Hadoop based applications.

• Scalability

HADOOP clusters can easily be scaled to any extent by adding additional cluster nodes, and thus allows for growth of Big Data. Also, scaling does not require modifications to application logic.

• Fault Tolerance

HADOOP ecosystem has a provision to replicate the input data on to other cluster nodes. That way, in the event of a cluster node failure, data processing can still proceed by using data stored on another cluster node.

Hadoop Yarn

Apache Yarn – "**Y**et **A**nother **R**esource **N**egotiator" is the resource management layer of **Hadoop**. The Yarn was introduced in Hadoop 2.x. Yarn allows different data processing engines like graph processing, interactive processing, stream processing as well as batch processing to run and process data stored in **HDFS** (Hadoop Distributed File System). Apart from resource management, Yarn also does job Scheduling. Yarn extends the power of Hadoop to other evolving technologies, so they can take the advantages of HDFS (most reliable and popular storage system on the planet) and

economic cluster. To learn installation of Apache Hadoop 2 with Yarn follows this quick installation guide.

Apache yarn is also a data operating system for Hadoop 2.x. This architecture of Hadoop 2.x provides a general purpose data processing platform which is not just limited to the **MapReduce**. It enables Hadoop to process other purpose-built data processing system other than MapReduce. It allows running several different frameworks on the same hardware where Hadoop is deployed.

Hadoop Yarn Architecture

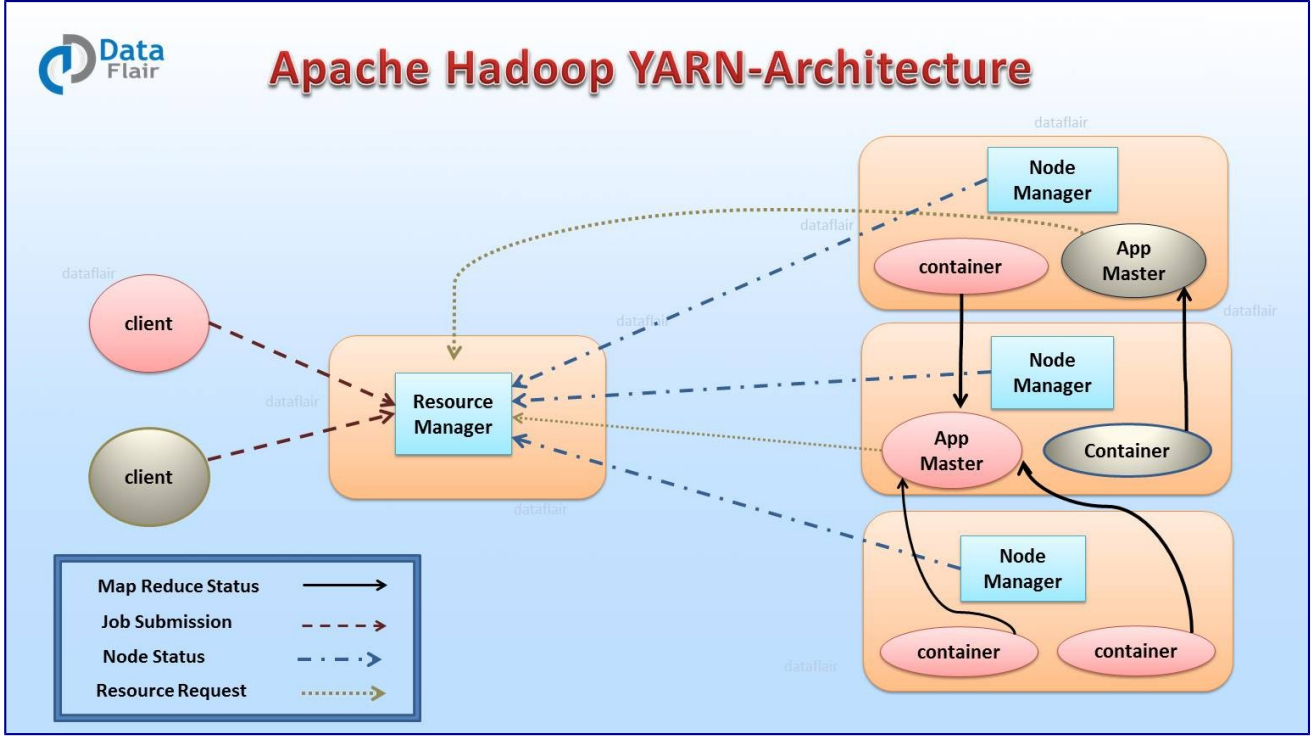
In this section of Hadoop Yarn tutorial, we will discuss the complete architecture of Yarn. Apache Yarn Framework consists of a master daemon known as “Resource Manager”, slave daemon called node manager (one per slave node) and Application Master (one per application).

Resource Manager (RM)

It is the master daemon of Yarn. RM manages the global assignments of resources (CPU and memory) among all the applications. It arbitrates system resources between competing applications. follow Resource Manager guide to learn Yarn Resource manager in great detail.

Resource Manager has two Main components

- Scheduler
- Application manager



a) Scheduler

The scheduler is responsible for allocating the resources to the running application. The scheduler is pure scheduler it means that it performs no monitoring no tracking for the application and even doesn't guarantees about restarting failed tasks either due to application failure or hardware failures.

b) Application Manager

It manages running Application Masters in the cluster, i.e., it is responsible for starting application masters and for monitoring and restarting them on different nodes in case of failures.

Node Manager (NM)

It is the slave daemon of Yarn. NM is responsible for containers monitoring their resource usage and reporting the same to the ResourceManager. Manage the user process on that machine. Yarn NodeManager also tracks the health of the node on which it is running. The design also allows plugging long-running auxiliary services to the NM; these are application-specific services, specified as part of the configurations and loaded by the NM during startup. A shuffle is a typical auxiliary service by the NMs for MapReduce applications on YARN

Application Master (AM)

One application master runs per application. It negotiates resources from the resource manager and works with the node manager. It Manages the application life cycle.The AM acquires containers

from the RM’s Scheduler before contacting the corresponding NMs to start the application’s individual tasks.

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

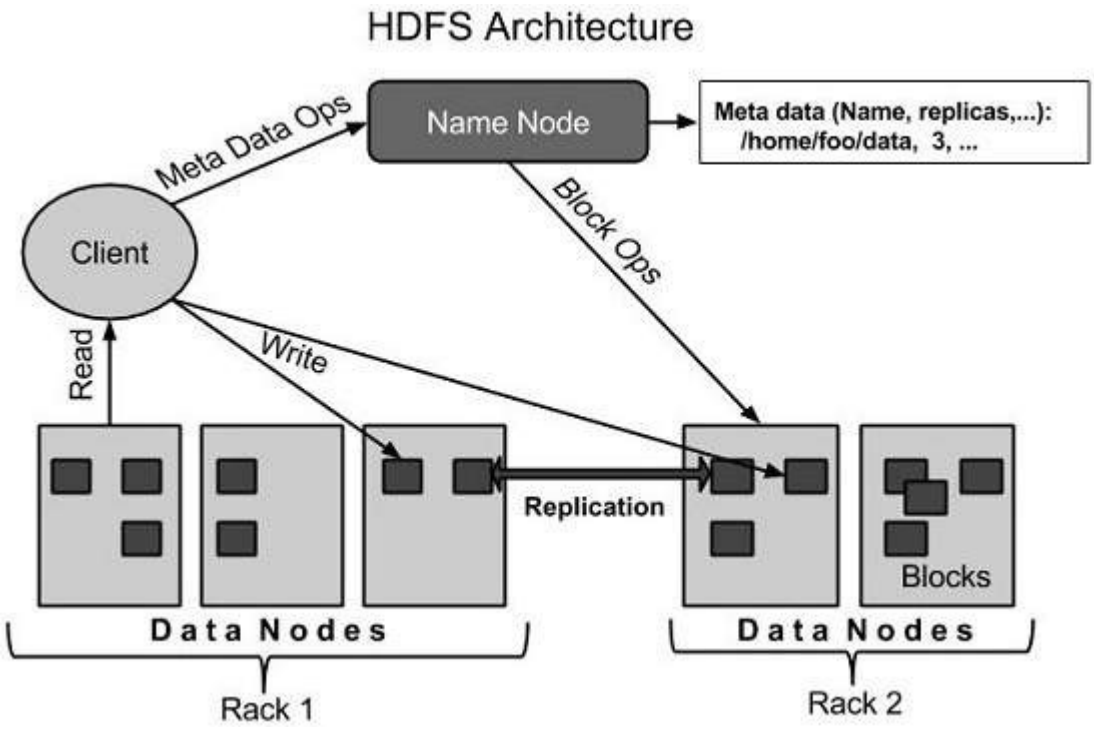
HDFS holds very large amount of data and provides easier access. To store such huge data, the files are stored across multiple machines. These files are stored in redundant fashion to rescue the system from possible data losses in case of failure. HDFS also makes applications available to parallel processing.

Features of HDFS

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

HDFS Architecture

Given below is the architecture of a Hadoop File System.



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

Namenode

The namenode is the commodity hardware that contains the GNU/Linux operating system and the namenode software. It is a software that can be run on commodity hardware. The system having the namenode acts as the master server and it does the following tasks:

- Manages the file system namespace.

- Regulates client’s access to files.
- It also executes file system operations such as renaming, closing, and opening files and directories.

Datanode

The datanode is a commodity hardware having the GNU/Linux operating system and datanode software. For every node (Commodity hardware/System) in a cluster, there will be a datanode. These nodes manage the data storage of their system.

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

Block

Generally the user data is stored in the files of HDFS. The file in a file system will be divided into one or more segments and/or stored in individual data nodes. These file segments are called as blocks. In other words, the minimum amount of data that HDFS can read or write is called a Block. The default block size is 64MB, but it can be increased as per the need to change in HDFS configuration.

Goals of HDFS

- **Fault detection and recovery** : Since HDFS includes a large number of commodity hardware, failure of components is frequent. Therefore HDFS should have mechanisms for quick and automatic fault detection and recovery.
- **Huge datasets** : HDFS should have hundreds of nodes per cluster to manage the applications having huge datasets.
- **Hardware at data** : A requested task can be done efficiently, when the computation takes place near the data. Especially where huge datasets are involved, it reduces the network traffic and increases the throughput

MapReduce

MapReduce is the processing layer of Hadoop. MapReduce programming model is designed for processing large volumes of data in parallel by dividing the work into a set of independent tasks. You need to put business logic in the way MapReduce works and rest things will be taken care by the framework. Work (complete job) which is submitted by the user to master is divided into small works (tasks) and assigned to slaves

Map-Reduce is the data processing component of Hadoop. Map-Reduce programs transform lists of input data elements into lists of output data elements. A Map-Reduce program will do this twice, using two different list processing idioms-

- Map
- Reduce

In between Map and Reduce, there is small phase called **Shuffle** and **Sort** in MapReduce.

What is a MapReduce Job?

MapReduce Job or a A "full program" is an execution of a **Mapper** and **Reducer** across a data set. It is an execution of 2 processing layers i.e mapper and reducer. A MapReduce job is a work that the client wants to be performed. It consists of the input data, the MapReduce Program, and configuration info. So client needs to submit input data, he needs to write Map Reduce program and set the configuration info (These were provided during **Hadoop setup** in the configuration file and also we specify some configurations in our program itself which will be specific to our **map reduce job)**.

- **What is Task in Map Reduce?**

A task in MapReduce is an execution of a Mapper or a Reducer on a slice of data. It is also called Task-In-Progress (TIP). It means processing of data is in progress either on mapper or reducer.

- **What is Task Attempt?**

Task Attempt is a particular instance of an attempt to execute a task on a node. There is a possibility that anytime any machine can go down. For example, while processing data if any node goes down, framework reschedules the task to some other node. This rescheduling of the task cannot be infinite. There is an upper limit for that as well. The default value of task attempt is 4. If a task (Mapper or

reducer) fails 4 times, then the job is considered as a failed job. For high priority job or huge job, the value of this task attempt can also be increased

Map Abstraction

Let us understand the abstract form of **Map** in MapReduce, the first phase of MapReduce paradigm, what is a map/mapper, what is the input to the mapper, how it processes the data, what is output from the mapper?

The **map** takes **key/value pair** as input. Whether data is in structured or unstructured format, framework converts the incoming data into key and value.

- Key is a reference to the input value.
- Value is the data set on which to operate.

Map Processing:

- A function defined by user – user can write custom business logic according to his need to process the data.
- Applies to every value in value input.

Map produces a new list of key/value pairs:

- An output of Map is called intermediate output.
- Can be the different type from input pair.
- An output of map is stored on the local disk from where it is shuffled to reduce nodes.

Next in Hadoop MapReduce Tutorial is the Hadoop Abstraction

Reduce Abstraction

Now let's discuss the second phase of MapReduce – **Reducer** in this MapReduce Tutorial, what is the input to the reducer, what work reducer does, where reducer writes output?

Reduce takes intermediate Key / Value pairs as input and processes the output of the mapper. Usually, in the reducer, we do aggregation or summation sort of computation.

- Input given to reducer is generated by Map (intermediate output)
- Key / Value pairs provided to reduce are sorted by key

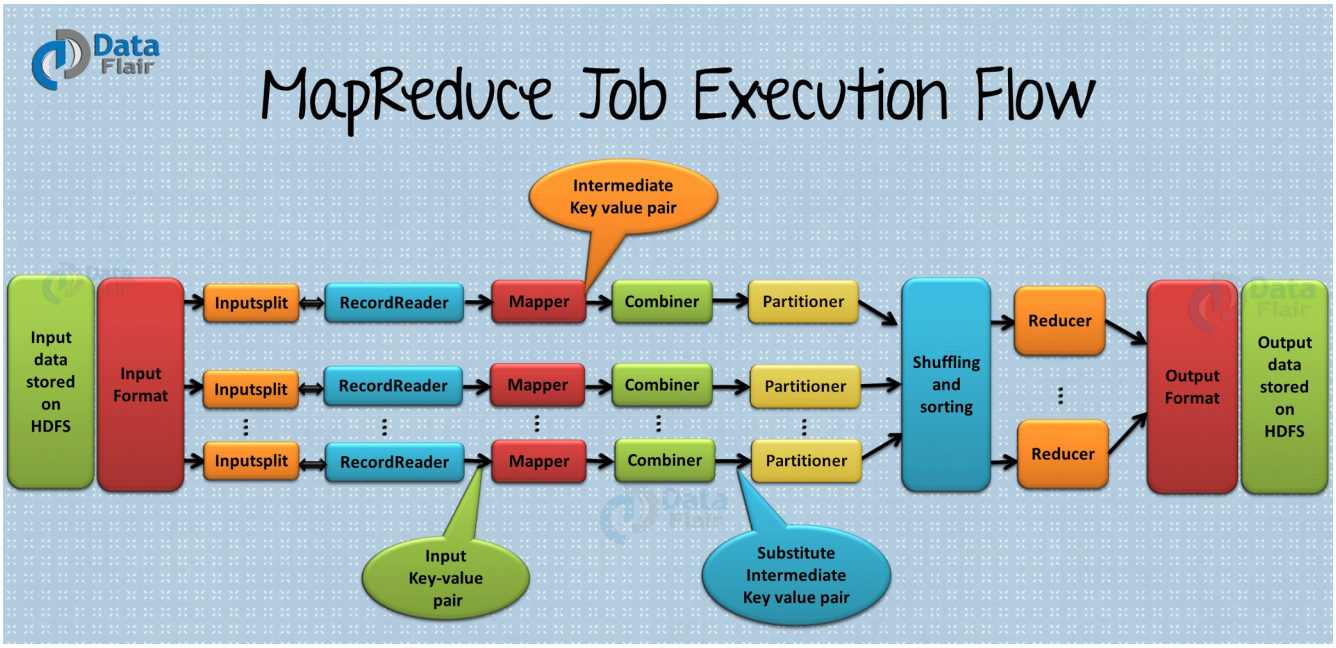
Reduce processing:

- A function defined by user – Here also user can write custom business logic and get the final output.
- Iterator supplies the values for a given key to the Reduce function.

Reduce produces a final list of key/value pairs:

- An output of Reduce is called Final output.
- It can be a different type from input pair.
- An output of Reduce is stored in **HDFS**.

How Hadoop MapReduce Works?



Input Files

The data for a MapReduce task is stored in **input files**, and input files typically lives in **HDFS**. The format of these files is arbitrary, while line-based log files and binary format can also be used.

Refer this quick guide to learn HDFS features in detail.

InputFormat

Now, **InputFormat** defines how these input files are split and read. It selects the files or other objects that are used for input. InputFormat creates InputSplit.

InputSplits

It is created by InputFormat, logically represent the data which will be processed by an individual **Mapper** (We will understand mapper below). One map task is created for each split; thus the number of map tasks will be equal to the number of InputSplits. The split is divided into records and each record will be processed by the mapper.

RecordReader

It communicates with the **InputSplit** in Hadoop MapReduce and converts the data into **key-value pairs** suitable for reading by the mapper. By default, it uses TextInputFormat for converting data into a key-value pair. RecordReader communicates with the InputSplit until the file reading is not completed. It assigns byte offset (unique number) to each line present in the file. Further, these key-value pairs are sent to the mapper for further processing.

Mapper

It processes each input record (from RecordReader) and generates new key-value pair, and this key-value pair generated by Mapper is completely different from the input pair. The output of Mapper is also known as intermediate output which is written to the local disk. The output of the Mapper is not stored on HDFS as this is temporary data and writing on HDFS will create unnecessary copies (also HDFS is a high latency system). Mappers output is passed to the combiner for further process.

Combiner

The combiner is also known as ‘Mini-reducer’. Hadoop MapReduce Combiner performs local aggregation on the mappers’ output, which helps to minimize the data transfer between mapper and **reducer** (we will see reducer below). Once the combiner functionality is executed, the output is then passed to the partitioner for further work.

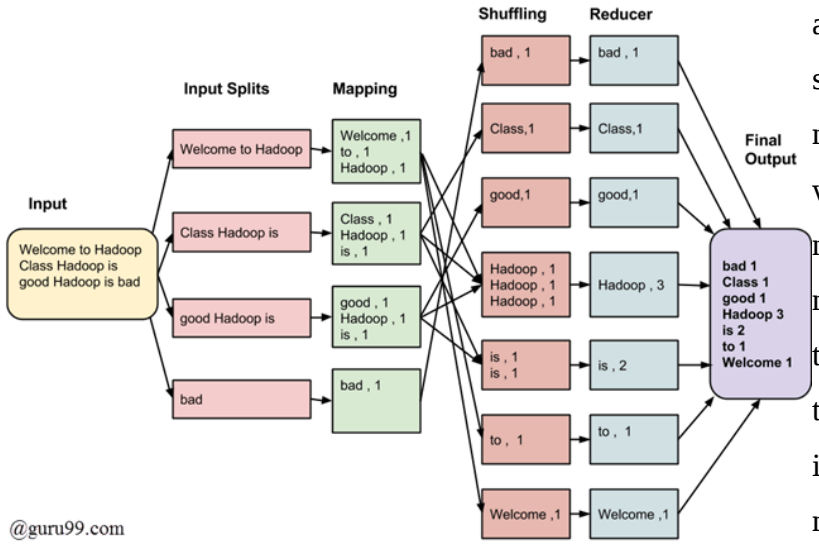
Partitioner

Hadoop MapReduce, **Partitioner** comes into the picture if we are working on more than one reducer (for one reducer partitioner is not used).

Partitioner takes the output from combiners and performs partitioning. Partitioning of output takes place on the basis of the key and then sorted. By hash function, key (or a subset of the key) is used to derive the partition.According to the key value in MapReduce, each combiner output is partitioned, and a record having the same key value goes into the same partition, and then each partition is sent to a reducer. Partitioning allows even distribution of the map output over the reducer.

Shuffling and Sorting

Now, the output is Shuffled to the reduce node (which is a normal slave node but reduce phase will run here hence called as reducer node). The shuffling is the physical movement of the data which is done over the network. Once all the mappers are finished and their output is shuffled on the reducer nodes, then this intermediate output is merged and sorted, which is then provided as input to reduce phase.



Reducer

It takes the set of intermediate key-value pairs produced by the mappers as the input and then runs a reducer function on each of them to generate the output. The output of the reducer is the final output, which is stored in HDFS.

RecordWriter

It writes these output key-value pair from the Reducer phase to the output files.

OutputFormat

The way these output key-value pairs are written in output files by RecordWriter is determined by the OutputFormat. OutputFormat instances provided by the Hadoop are used to write files in HDFS or on the local disk. Thus the final output of reducer written on HDFS by OutputFormat instances.

Apache Hive

Apache Hive is an open source data warehouse system built on top of Hadoop Haused for querying and analyzing large datasets stored in Hadoop files. It process structured and semi-structured data in Hadoop.This Apache Hive tutorial explains basics of Apache Hive & Hive history in great details. In this hive tutorial, we will learn about the need for a hive and its characteristics. This Hive guide also covers internals of Hive architecture, Hive Features and Drawbacks of Apache Hive.

What is Hive?

Apache Hive is an open source data warehouse system built on top of Hadoop Haused for querying and analyzing large datasets stored in Hadoop files.Initially, you have to write complex **Map-Reduce** jobs, but now with the help of Hive, you just need to submit merely **SQL** queries. Hive is mainly targeted towards users who are comfortable with SQL. Hive use language called **HiveQL** (HQL), which is similar to SQL. HiveQL automatically translates SQL-like queries into MapReduce jobs.Hive abstracts the complexity of Hadoop. The main thing to notice is that there is no need to learn java for Hive.

The Hive generally runs on your workstation and converts your SQL query into a series of jobs for execution on a **Hadoop cluster**. Apache Hive organizes data into tables. This provides a means for attaching the structure to data stored in **HDFS**.

History of Hive

Data Infrastructure Team at Facebook developed Hive. Apache Hive is also one of the technologies that are being used to address the requirements at Facebook. It is very popular with all the users internally at Facebook. It is being used to run thousands of jobs on the cluster with hundreds of users, for a wide variety of applications.Apache Hive-Hadoop cluster at Facebook stores more than 2PB of raw data. It regularly loads 15 TB of data on a daily basis.Now it is being used and developed by a number of companies like Amazon, IBM, Yahoo, Netflix, Financial Industry Regulatory Authority (FINRA) and many others.

Why Apache Hive?

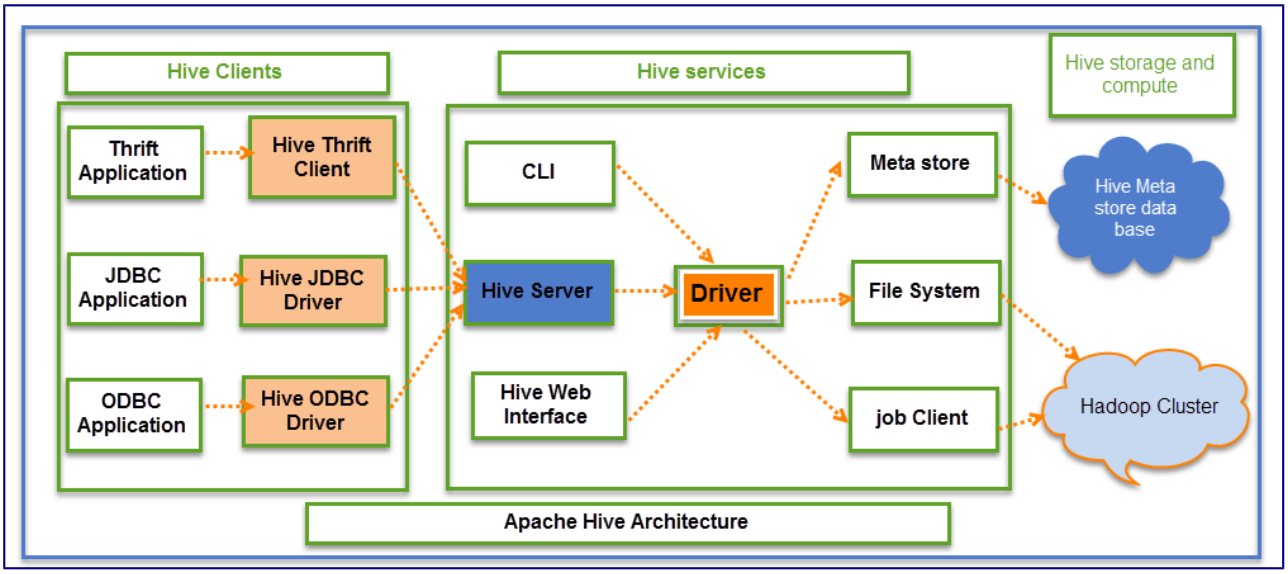
Facebook had faced a lot of challenges before implementation of Apache Hive. Challenges like the size of data being generated increased or exploded, making it very difficult to handle them. The traditional **RDBMS** could not handle the pressure. As a result, Facebook was looking out for better options. To overcome this problem, Facebook initially tried using **MapReduce**. But it has difficulty in programming and mandatory knowledge in SQL, making it an impractical solution. Hence, Apache Hive allowed them to overcome the challenges they were facing.

With Apache Hive, they are now able to perform the following:

- Schema flexibility and evolution
- Tables can be portioned and bucketed
- Apache Hive tables are defined directly in the HDFS
- JDBC/ODBC drivers are available

Apache Hive saves developers from writing complex Hadoop MapReduce jobs for ad-hoc requirements. Hence, hive provides summarization, analysis, and query of data. Hive is very fast and scalable. It is highly extensible. Since Apache Hive is similar to SQL, hence it becomes very easy for the SQL developers to learn and implement Hive Queries. Hive reduces the complexity of MapReduce by providing an interface where the user can submit SQL queries. So, now business analysts can play with **Big Data** using Apache Hive and generate insights. It also provides file access on various data stores like HDFS and HBase. The most important feature of Apache Hive is that to learn Hive we don’t have to learn Java.

Hive Architecture



The above screenshot explains the Apache Hive architecture in detail

Hive Consists of Mainly 3 core parts

1. Hive Clients
2. Hive Services
3. Hive Storage and Computing

Hive Clients:

Hive provides different drivers for communication with a different type of applications. For Thrift based applications, it will provide Thrift client for communication. For Java related applications, it provides JDBC Drivers. Other than any type of applications provided ODBC drivers. These Clients and drivers in turn again communicate with Hive server in the Hive services.

Hive Services:

Client interactions with Hive can be performed through Hive Services. If the client wants to perform any query related operations in Hive, it has to communicate through Hive Services. CLI is the command line interface acts as Hive service for DDL (Data definition Language) operations. All drivers communicate with Hive server and to the main driver in Hive services as shown in above architecture diagram

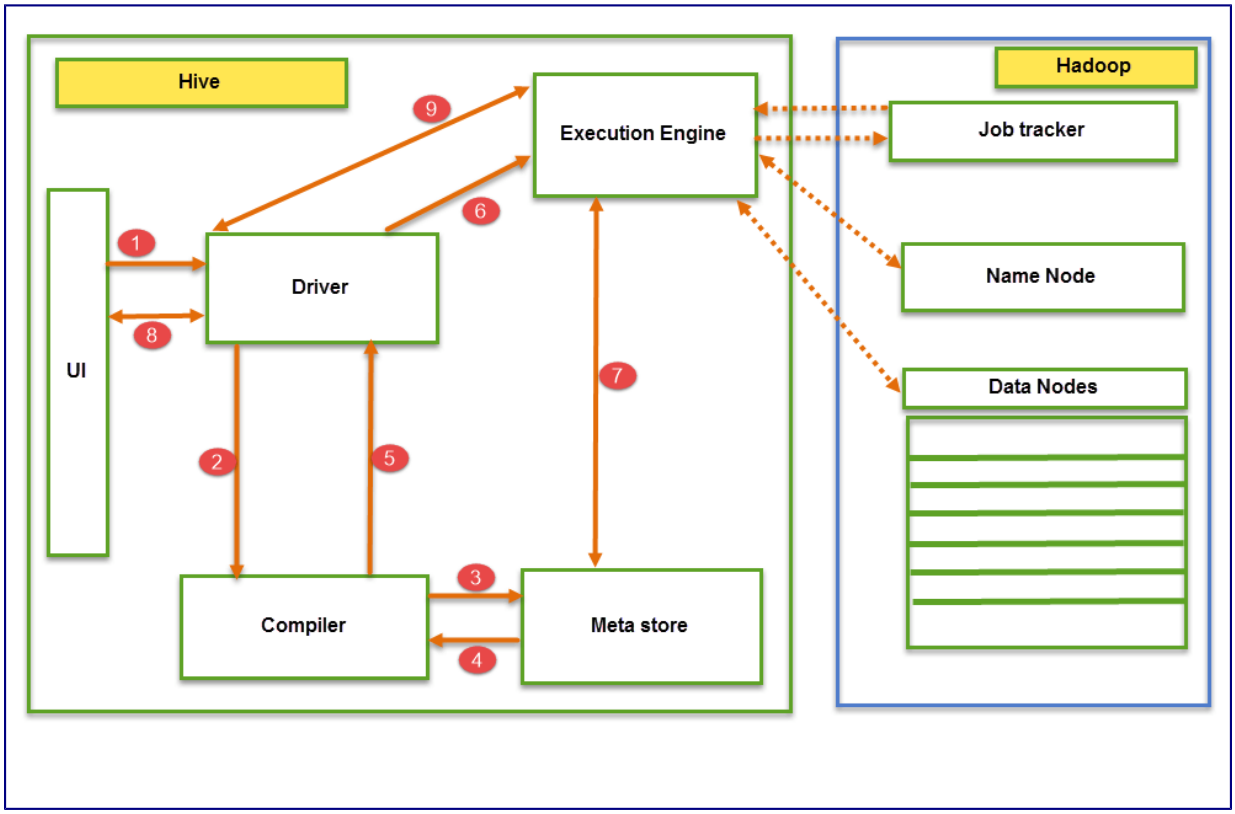
Driver present in the Hive services represents the main driver, and it communicates all type of JDBC, ODBC, and other client specific applications. Driver will process those requests from different applications to meta store and field systems for further processing.

Hive Storage and Computing:

Hive services such as Meta store, File system, and Job Client in turn communicates with Hive storage and performs the following actions

- Metadata information of tables created in Hive is stored in Hive "Meta storage database".
- Query results and data loaded in the tables are going to be stored in Hadoop cluster on HDFS.

Job exectution flow:



From the above screenshot we can understand the Job execution flow in Hive with Hadoop

The data flow in Hive behaves in the following pattern;

1. Executing Query from the UI(User Interface)
2. The driver is interacting with Compiler for getting the plan. (Here plan refers to query execution) process and its related metadata information gathering

3. The compiler creates the plan for a job to be executed. Compiler communicating with Meta store for getting metadata request
4. Meta store sends metadata information back to compiler
5. Compiler communicating with Driver with the proposed plan to execute the query
6. Driver Sending execution plans to Execution engine
7. Execution Engine (EE) acts as a bridge between Hive and Hadoop to process the query. For DFS operations.

- EE should first contacts Name Node and then to Data nodes to get the values stored in tables.
- EE is going to fetch desired records from Data Nodes. The actual data of tables resides in data node only. While from Name Node it only fetches the metadata information for the query.
- It collects actual data from data nodes related to mentioned query
- Execution Engine (EE) communicates bi-directionally with Meta store present in Hive to perform DDL (Data Definition Language) operations. Here DDL operations like CREATE, DROP and ALTERING tables and databases are done. Meta store will store information about database name, table names and column names only. It will fetch data related to query mentioned.
- Execution Engine (EE) in turn communicates with Hadoop daemons such as Name node, Data nodes, and job tracker to execute the query on top of Hadoop file system

8. Fetching results from driver
9. Sending results to Execution engine. Once the results fetched from data nodes to the EE, it will send results back to driver and to UI (front end)

Hive Continuously in contact with Hadoop file system and its daemons via Execution engine. The dotted arrow in the Job flow diagram shows the Execution engine communication with Hadoop daemons.

Different modes of Hive

Hive can operate in two modes depending on the size of data nodes in Hadoop.

These modes are,

- Local mode
- Map reduce mode

When to use Local mode:

- If the Hadoop installed under pseudo mode with having one data node we use Hive in this mode
- If the data size is smaller in term of limited to single local machine, we can use this mode
- Processing will be very fast on smaller data sets present in the local machine

When to use Map reduce mode:

- If Hadoop is having multiple data nodes and data is distributed across different node we use Hive in this mode
- It will perform on large amount of data sets and query going to execute in parallel way
- Processing of large data sets with better performance can be achieved through this mode

In Hive, we can set this property to mention which mode Hive can work? By default, it works on Map Reduce mode and for local mode you can have the following setting.

Hive to work in local mode set

SET mapred.job.tracker=local;

From the Hive version 0.7 it supports a mode to run map reduce jobs in local mode automatically.

Features of Hive

There are so many features of Apache Hive. Let’s discuss them one by one-

- Hive provides data summarization, query, and analysis in much easier manner.
- Hive supports external tables which make it possible to process data without actually storing in HDFS.
- Apache Hive fits the low-level interface requirement of Hadoop perfectly.
- It also supports partitioning of data at the level of tables to improve performance.
- Hive has a rule based optimizer for optimizing logical plans.
- It is scalable, familiar, and extensible.
- Using HiveQL doesn’t require any knowledge of programming language, Knowledge of basic SQL query is enough.
- We can easily process structured data in Hadoop using Hive.
- Querying in Hive is very simple as it is similar to SQL.

- We can also run Ad-hoc queries for the data analysis using Hive.

8. Limitation of Hive

Hive has the following limitations-

- Apache does not offer real-time queries and row level updates.
- Hive also provides acceptable latency for interactive data browsing.
- It is not good for online transaction processing.
- Latency for Apache Hive queries is generally very high.

Hadoop Pig

Hadoop Pig is nothing but an abstraction over MapReduce. While it comes to analyze large sets of data, as well as to represent them as data flows, we use Apache Pig. Generally, we use it with Hadoop. By using Pig, we can perform all the data manipulation operations in Hadoop.In addition, Pig offers a high-level language to write data analysis programs which we call as Pig Latin. One of the major advantages of this language is, it offers several operators. Through them, programmers can develop their own functions for reading, writing, and processing data.

It has following key properties such as:

- Ease of programming

Basically, when all the complex tasks comprised of multiple interrelated data transformations are explicitly encoded as data flow sequences, that makes them easy to write, understand, and maintain.

- Optimization opportunities

It allows users to focus on semantics rather than efficiency, to optimize their execution automatically, in which tasks are encoded permits the system.

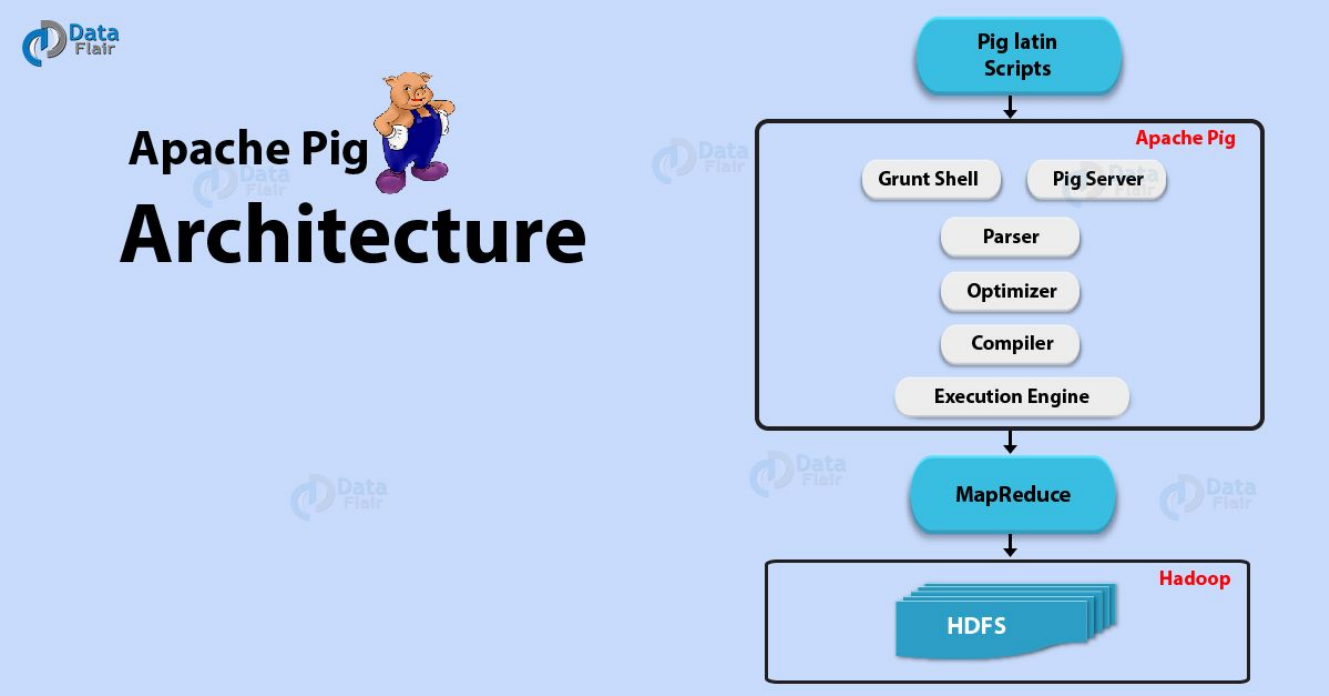
- Extensibility

In order to do special-purpose processing, users can create their own functions.

Hence, programmers need to write scripts using Pig Latin language to analyze data using Apache Pig. However, all these scripts are internally converted to Map and Reduce tasks. It is possible with a

component, we call as Pig Engine. That accepts the Pig Latin scripts as input and further convert those scripts into MapReduce jobs.

Architecture of Hadoop Pig



Now, you can see, several components in the Hadoop Pig framework. The major components are:

i. Parser

At first, all the Pig Scripts are handled by the Parser. Basically, Parser checks the syntax of the script, does type checking, and other miscellaneous checks. Afterward, Parser's output will be a DAG (directed acyclic graph). That represents the Pig Latin statements as well as logical operators.

Basically, the logical operators of the script are represented as the nodes and the data flows are represented as edges, in the DAG (the logical plan).

ii. Optimizer

Further, DAG is passed to the logical optimizer. That carries out the logical optimizations. Like projection and push down.

iii. Compiler

It compiles the optimized logical plan into a series of MapReduce jobs.

iv. Execution Engine

At last, MapReduce jobs are submitted to Hadoop in a sorted order. Hence, these MapReduce jobs are executed finally on Hadoop, that produces the desired results.

Features of Pig

Now in the Hadoop Pig Tutorial is the time to learn the Features of Pig which makes it what it is. There are several features of Pig. Such as:

i. Rich set of operators

In order to perform several operations, Pig offers many operators. Such as join, sort, filer and many more.

ii. Ease of programming

Since you are good at SQL, it is easy to write a Pig script. Because of Pig Latin as same as SQL.

iii. Optimization opportunities

In Apache Pig, all the tasks optimize their execution automatically. As a result, the programmers need to focus only on the semantics of the language.

iv. Extensibility

Through Pig, it is easy to read, process, and write data. It is possible by using the existing operators. Also, users can develop their own functions.

v. UDFs

By using Pig, we can create User-defined Functions in other programming languages like Java. Also, can invoke or embed them in Pig Scripts.

vi. Handles all kinds of data

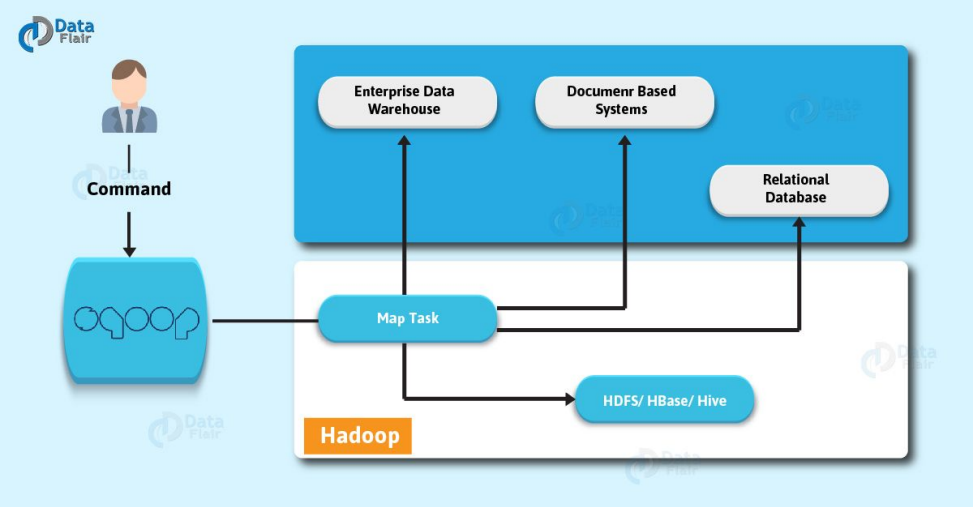
Pig generally analyzes all kinds of data. Even both structured and unstructured. Moreover, it stores the results in HDFS.

Sqoop

Basically, by using RDBMS applications generally interacts with the relational database. Therefore it makes relational databases one of the most important sources that generate Big Data. However, in the relational structures, such data is stored in RDB Servers. Although, by offering feasible interaction between the relational database server and HDFS, Sqoop plays the vital role in Hadoop ecosystem.

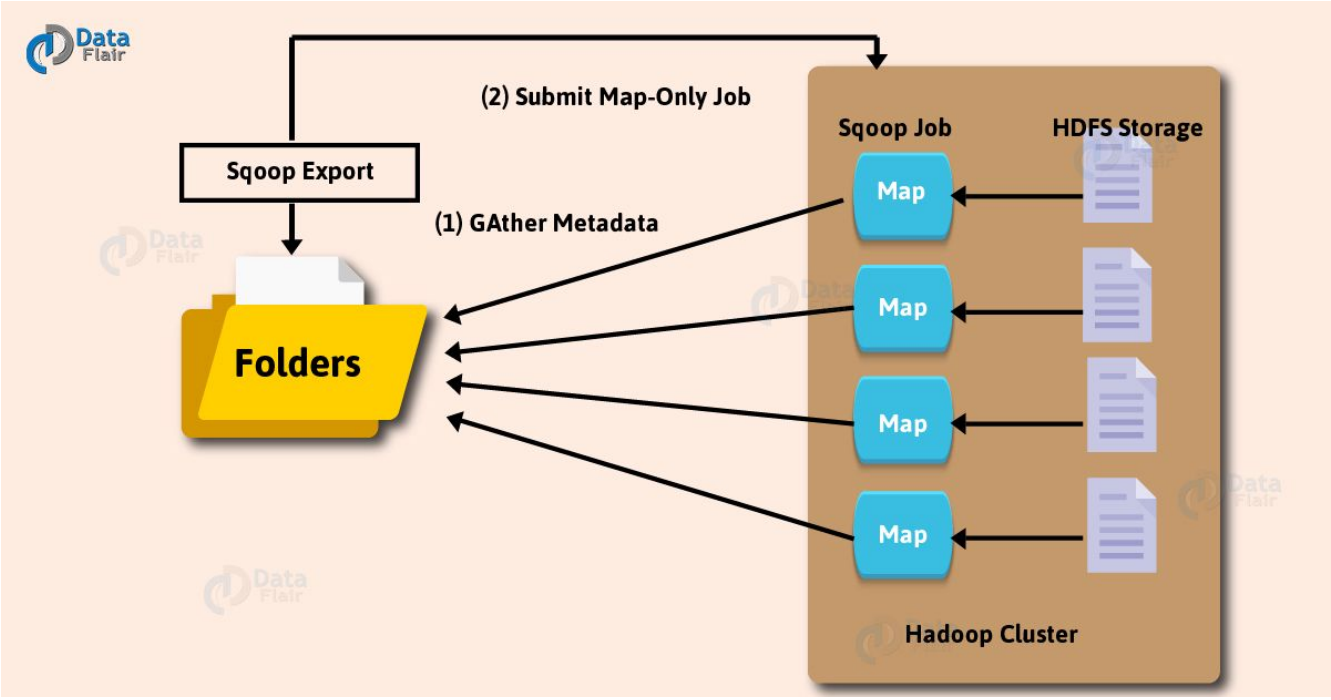
To be more specific, it is a tool that aims to transfer data between HDFS (Hadoop storage) and relational database servers. Such as MySQL, Oracle RDB, SQLite, Teradata, Netezza, Postgres and many more. In addition, imports data from relational databases to HDFS. Also, exports data from HDFS to relational databases. Moreover, Sqoop can transfer bulk data efficiently between Hadoop and external data stores. Like as enterprise data warehouses, relational databases, etc.However, it is very interesting to know that this is how Sqoop got its name. “SQL to Hadoop & Hadoop to SQL”.Moreover, to import data from external datastores into Hadoop ecosystem tools we use Sqoop. Such as Hive & HBase.Since we know what is Apache Sqoop now.

Sqoop Architecture and Working



Basically, a tool which imports individual tables from RDBMS to HDFS is what we call Sqoop import tool. However, in HDFS we treat each row in a table as a record.Moreover, our main task gets divided into subtasks, while we submit Sqoop command. However, map task individually handles it internally. On defining map task, it is the subtask that imports part of data to the Hadoop Ecosystem. Likewise, we can say all map tasks import the whole data collectively.A tool which exports a set of files from HDFS back to an RDBMS is a Sqoop Export tool. Moreover, there are files which behave as input to Sqoop which also contain records. Those files what we call as rows in the table.

Moreover, the job is mapped into map tasks, while we submit our job, that brings the chunk of data



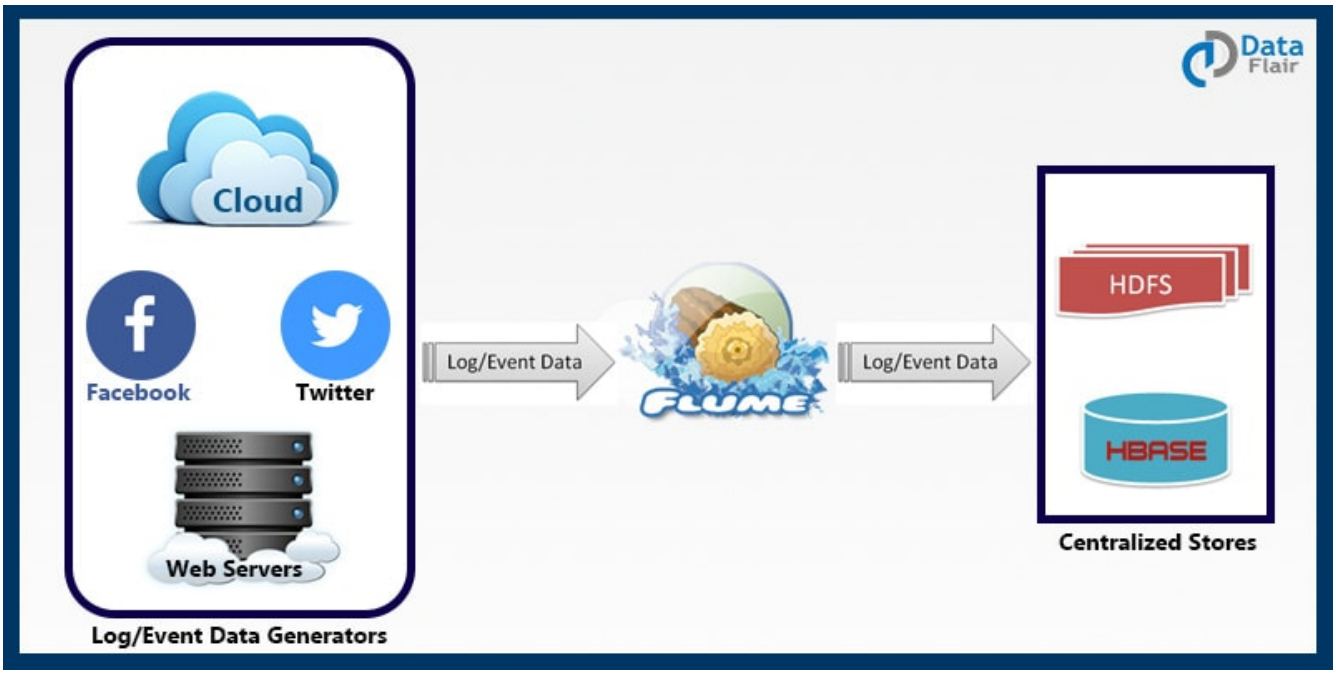
from HDFS. Then we export these chunks to a structured data destination. Likewise, we receive the whole data at the destination by combining all these exported chunks of data. However, in most of the cases, it is an RDBMS (MYSQL/Oracle/SQL Server).

In addition, in case of aggregations, we require reducing phase. However, Sqoop does not perform any aggregations it just imports and exports the data. Also, on the basis of the number defined by the user, map job launch multiple mappers.

In addition, each mapper task will be assigned with a part of data to be imported for Sqoop import. Also, to get high-performance Sqoop distributes the input data among the mappers equally. Afterwards, by using JDBC each mapper creates the connection with the database. Also fetches the part of data assigned by Sqoop.

Apache Flume

Apache flume is an open source data collection service for moving the data from source to destination. In this Apache Flume tutorial, we will discuss what is Apache Flume, what is the need of Flume, features of Apache Flume. This tutorial also covers the architecture of flume, how Flume works and various advantages of Apache Flume.



Apache Flume is a tool used to collect, aggregate and transports large amounts of streaming data like log files, events, etc., from a number of different sources to a centralized data store (say Hadoop Distributed File System – HDFS).Flume is a highly distributed, reliable, and configurable tool. Flume was mainly designed in order to collect streaming data (log data) from various web servers to HDFS.

Why Apache Flume?

A company has tons of services running on multiple servers. And lots of data (logs) are produced by them, now we need to analyze them altogether. In order process that logs, we need a reliable, scalable, extensible and manageable distributed data collection service which can perform flow of unstructured data (logs) from one location to another where they will be processed (say in

HDFS). Apache flume is an open source data collection service for moving the data from source to destination.Apache Flume is the most reliable, distributed, and available service for systematically collecting, aggregating, and moving large amounts of streaming data (logs) into the Hadoop Distributed File System (HDFS). Based on streaming data flows, it has a simple and flexible architecture. It is highly fault-tolerant and robust and with tunable reliability mechanisms for fail-over and recovery. Flume allows data collection in batch as well as streaming mode.

Flume Features

Some of the outstanding features of Flume are as follows:

- From multiple servers, it collects the log data and ingests them into a centralized store (HDFS, HBase) efficiently.
- With the help of Flume we can collect the data from multiple servers in real-time as well as in batch mode.
- Huge volumes of event data generated by social media websites like Facebook and Twitter and various e-commerce websites such as Amazon and Flipkart can also be easily imported and analyzed in real-time.
- Flume can collect data from a large set of sources and move them to multiple destinations.
- Multi-hop flows, fan-in fan-out flows, contextual routing, etc are supported by Flume.
- Flume can be scaled horizontally.

Apache Flume Architecture

In this section of Apache Flume tutorial, we will discuss various components of Flume and how this components work-

- Event: A single data record entry transported by flume is known as Event.
- Source: Source is an active component that listens for events and writes them on one or channels. It is the main component with the help of which data enters into the Flume. It collects the data from a variety of sources, like exec, avro, JMS, spool directory, etc.
- Sink: It is that component which removes events from a channel and delivers data to the destination or next hop. There are multiple sinks available that delivers data to a wide range of destinations. Example: HDFS, HBase, logger, null, file, etc.

- Channel: It is a conduit between the Source and the Sink that queues event data as transactions. Events are ingested by the sources into the channel and drained by the sinks from the channel.
- Agent: An Agent is a Java virtual machine in which Flume runs. It consists of sources, sinks, channels and other important components through which events get transferred from one place to another.
- Client: Events are generated by the clients and are sent to one or more agents.

Advantages of Flume

Below are some important advantages of using Apache Flume:

- Data into any of the centralized stores can be stored using Apache Flume.
- Flume acts as a mediator between data producers and the centralized stores when the rate of incoming data exceeds the rate at which data can be written to the destination and provides a steady flow of data between them.
- A feature of contextual routing is also provided by Flume.
- Flume guarantees reliable message delivery as in Flume transactions are channel-based where two transactions (1 sender & 1 receiver) are maintained for each message.
- Flume is highly fault-tolerant, reliable, manageable, scalable, and customizable.

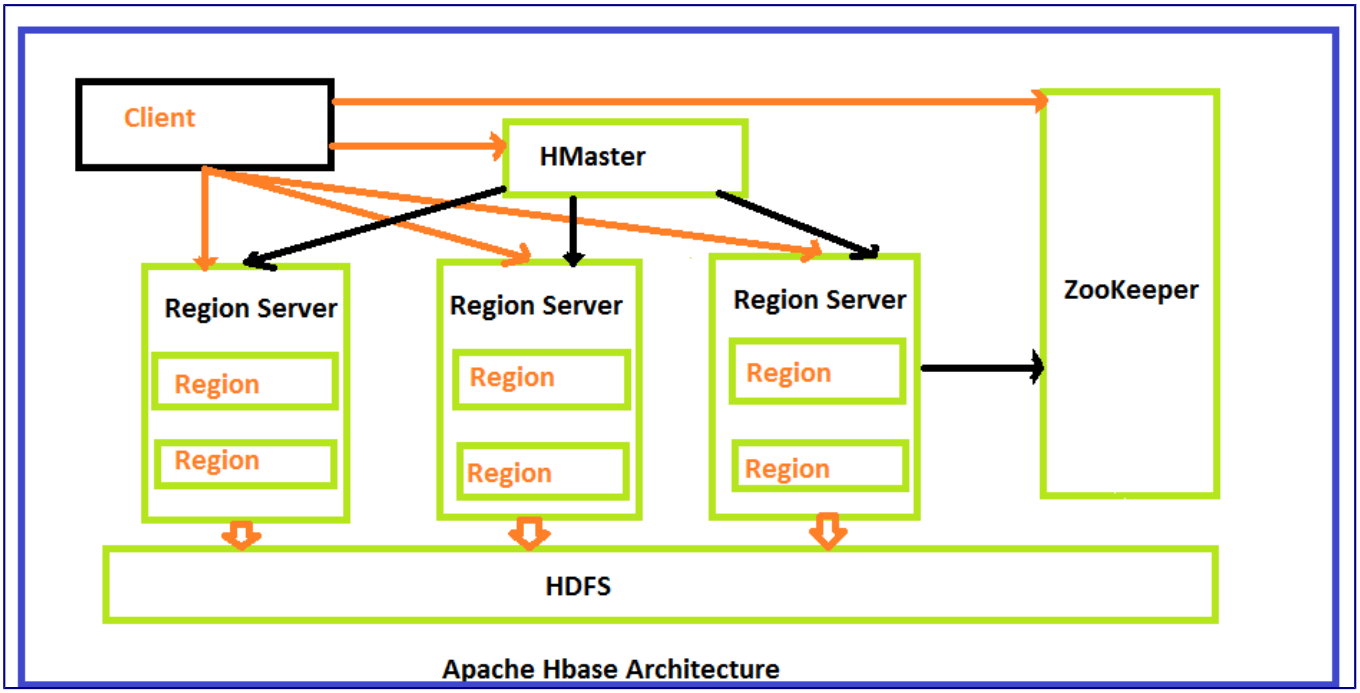
Hbase

HBase is an open source, distributed database, developed by Apache Software foundation. Initially, it was Google Big Table, afterwards it was re-named as HBase and is primarily written in Java. HBase can store massive amounts of data from terabytes to petabytes. HBase is an open-source, column-oriented distributed database system in Hadoop environment. Apache HBase is needed for real-time Big Data applications. The tables present in HBase consists of billions of rows having millions of columns.

Storage Mechanism in Hbase

HBase is a column-oriented database and data is stored in tables. The tables are sorted by RowId. As shown below, HBase has RowId, which is the collection of several column families that are present in the table.

HBase Architecture and its Important Components



HBase architecture consists mainly of four components

- HMaster
- HRegionserver
- HRegions
- Zookeeper

HMaster:

HMaster is the implementation of Master server in HBase architecture. It acts like monitoring agent to monitor all Region Server instances present in the cluster and acts as an interface for all the metadata changes. In a distributed cluster environment, Master runs on NameNode. Master runs several background threads.

The following are important roles performed by HMaster in HBase.

- Plays a vital role in terms of performance and maintaining nodes in the cluster.
- HMaster provides admin performance and distributes services to different region servers.
- HMaster assigns regions to region servers.
- HMaster has the features like controlling load balancing and failover to handle the load over nodes present in the cluster.

- When a client wants to change any schema and to change any Metadata operations, HMaster takes responsibility for these operations.

Some of the methods exposed by HMaster Interface are primarily Metadata oriented methods.

- Table (createTable, removeTable, enable, disable)
- ColumnFamily (add Column, modify Column)
- Region (move, assign)

The client communicates in a bi-directional way with both HMaster and ZooKeeper. For read and write operations, it directly contacts with HRegion servers. HMaster assigns regions to region servers and in turn check the health status of region servers. In entire architecture, we have multiple region servers. Hlog present in region servers which are going to store all the log files.

HRegions Servers:

When Region Server receives writes and read requests from the client, it assigns the request to a specific region, where actual column family resides. However, the client can directly contact with HRegion servers, there is no need of HMaster mandatory permission to the client regarding communication with HRegion servers. The client requires HMaster help when operations related to metadata and schema changes are required. HRegionServer is the Region Server implementation. It is responsible for serving and managing regions or data that is present in distributed cluster. The region servers run on Data Nodes present in the Hadoop cluster.

HMaster can get into contact with multiple HRegion servers and performs the following functions.

- Hosting and managing regions
- Splitting regions automatically
- Handling read and writes requests
- Communicating with the client directly

HRegions:

HRegions are the basic building elements of HBase cluster that consists of the distribution of tables and are comprised of Column families. It contains multiple stores, one for each column family. It consists of mainly two components, which are Memstore and Hfile.

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.Oozie is an Open Source Java Web-Application available under Apache license 2.0. It is responsible for triggering the workflow actions, which in turn uses the Hadoop execution engine to actually execute the task. Hence, Oozie is able to 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.

Apache Oozie is a workflow scheduler for Hadoop. It is a system which runs workflow of dependent jobs. Here, users are permitted to create Directed Acyclic Graphs of workflows, which can be run in parallel and sequentially in Hadoop

It consists of two parts:

- Workflow engine : Responsibility of a workflow engine is to store and run workflows composed of Hadoop jobs e.g., MapReduce, Pig, Hive.
- Coordinator engine: It runs workflow jobs based on predefined schedules and availability of data.

Oozie is scalable and can manage timely execution of thousands of workflows (each consisting of dozens of jobs) in a Hadoop cluster.

How does OOZIE work?

Oozie runs as a service in the cluster and clients submit workflow definitions for immediate or later processing. Oozie workflow consists of action nodes and control-flow nodes. An action node represents a workflow task, e.g., moving files into HDFS, running a MapReduce, Pig or Hive jobs, importing data using Sqoop or running a shell script of a program written in Java. A control-flow node controls the workflow execution between actions by allowing constructs like conditional logic wherein different branches may be followed depending on the result of earlier action node.

Start Node, End Node and Error Node fall under this category of nodes.

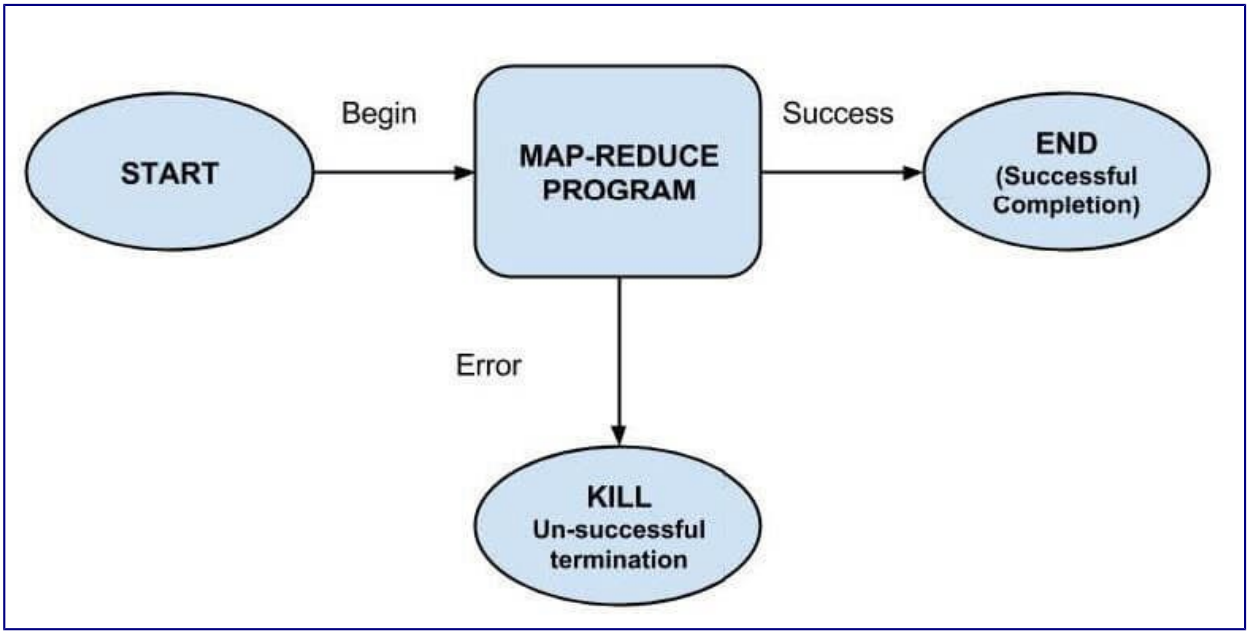
Start Node, designates start of the workflow job.

End Node, signals end of the job.

Error Node, designates an occurrence of error and corresponding error message to be printed.

At the end of execution of workflow, HTTP callback is used by Oozie to update client with the workflow status. Entry-to or exit-from an action node may also trigger callback.

Example Workflow Diagram



FEATURES OF OOZIE

- Oozie has client API and command line interface which can be used to launch, control and monitor job from Java application.
- Using its Web Service APIs one can control jobs from anywhere.
- Oozie has provision to execute jobs which are scheduled to run periodically.
- Oozie has provision to send email notifications upon completion of jobs.

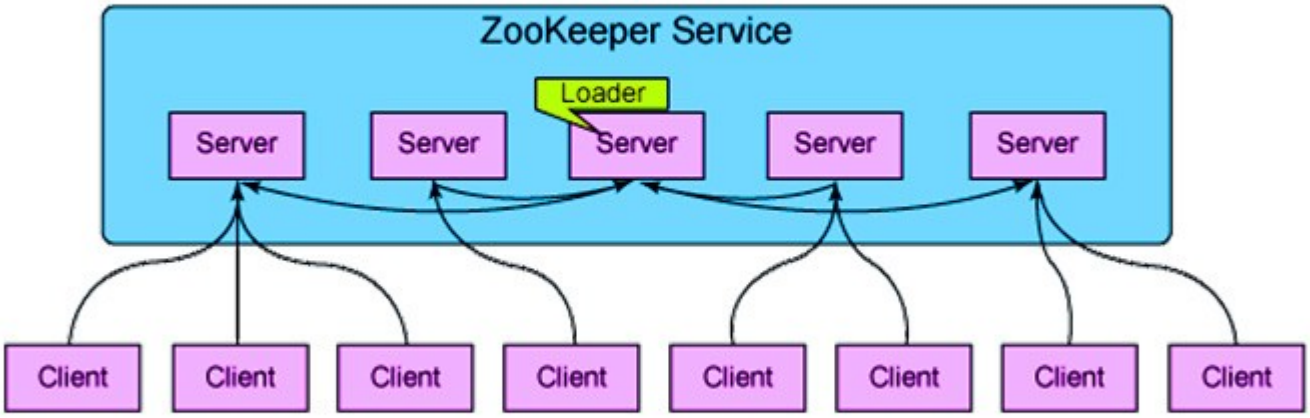
Apache ZooKeeper

Apache ZooKeeper is an open-source project which deals with maintaining configuration information, naming, providing distributed synchronization, group services for various distributed applications. It implements various protocols on the cluster so that the applications need not to implement them on their own. Developed originally at Yahoo, ZooKeeper facilitates synchronization among the process by maintaining a status on ZooKeeper servers that stores information in local log files. The Zookeeper servers are capable of supporting a large Hadoop cluster. Each client machine communicates to one of the servers to retrieve information.

Apache ZooKeeper architecture

Following constituents form the architecture of this technology:

Apache ZooKeeper basics	Tasks Performed
Server Applications	Facilitate interaction with client applications through a common interface.
Client Applications	Consist of tools that interact with distributed applications.
Zookeeper Nodes	Systems on which a cluster runs.
znode	Can be updated and/or modified by any node in the cluster.



Apache ZooKeeper Features

Though the distributed applications provide a lot of applications, they impose quite many difficulties which are hard to address. Apache ZooKeeper is an application which tries to overcome all these complexities by providing a wide range of services to the user.

- Naming service– Identifying ZooKeeper attaches a unique identification to every node which is quite similar to the DNA that helps identify it.
- Updating the node's status– Apache Zookeeper is capable of updating every node which allows it to store updated information about each node across the cluster.
- Managing the cluster– This technology is able to manage the cluster in such a way that the status of each node is maintained in real-time leaving lesser chances for errors and ambiguity.

Automatic failure recovery– Apache ZooKeeper locks the data while modifying which helps the cluster to recover it automatically if a failure occurs in the database.

PROJECT

H1-B Case Study

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.We will be performing analysis on the H1B visa applicants between the years 2011-2015. After analyzing the data, we can derive the following facts.

TOOLS USED FOR ANALYZING THE H1-B CASE STUDY

*MAP REDUCE
*HIVE
*PIG

UPLOADING INPUT DATA AS TABLE USING HIVE

CREATE TABLE h2b_applications(s_no int,case_status string, employer_name string, soc_name string, job_title string, full_time_position string,prevailing_wage int,year string, worksite string, longitute double, latitute double)row format delimited fields terminated by '#';

insert overwrite local directory 'file:///home/ancy/Documents/projectinput'row format delimited fields terminated by '#' select * from h1b_applications;
load data local inpath 'file:///home/ancy/Documents/projectinput/000000_0' into table h2b_applications;
load data local inpath 'file:///home/ancy/Documents/projectinput/000001_0' into table h2b_applications;

MAP REDUCE

1 b) Find top 5 job titles who are having highest growth in applications.

hadoop jar /home/ancy/Documents/jarfile/demo1bar.jar file:///home/ancy/Documents/newproinput \

1b

18/10/15 02:45:14 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

18/10/15 02:45:20 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032

18/10/15 02:45:25 WARN mapreduce.JobSubmitter: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.

18/10/15 02:45:30 INFO input.FileInputFormat: Total input paths to process : 1

18/10/15 02:45:31 INFO mapreduce.JobSubmitter: number of splits:1

18/10/15 02:45:33 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1539594939608_0001

18/10/15 02:45:35 INFO impl.YarnClientImpl: Submitted application application_1539594939608_0001

18/10/15 02:45:36 INFO mapreduce.Job: The url to track the job: http://ubuntu:8088/proxy/application_1539594939608_0001/

18/10/15 02:45:36 INFO mapreduce.Job: Running job: job_1539594939608_0001

18/10/15 02:46:23 INFO mapreduce.Job: Job job_1539594939608_0001 running in uber mode : false

18/10/15 02:46:23 INFO mapreduce.Job: map 0% reduce 0%

18/10/15 02:46:43 INFO mapreduce.Job: map 100% reduce 0%

18/10/15 02:46:59 INFO mapreduce.Job: map 100% reduce 100%

18/10/15 02:47:00 INFO mapreduce.Job: Job job_1539594939608_0001 completed successfully

18/10/15 02:47:01 INFO mapreduce.Job: Counters: 49

File System Counters

FILE: Number of bytes read=5036

FILE: Number of bytes written=213659

FILE: Number of read operations=0

FILE: Number of large read operations=0

FILE: Number of write operations=0

HDFS: Number of bytes read=102

HDFS: Number of bytes written=201

HDFS: Number of read operations=5

HDFS: Number of large read operations=0

HDFS: Number of write operations=2

Job Counters

Launched map tasks=1

Launched reduce tasks=1

Rack-local map tasks=1
Total time spent by all maps in occupied slots (ms)=15033
Total time spent by all reduces in occupied slots (ms)=14161
Total time spent by all map tasks (ms)=15033
Total time spent by all reduce tasks (ms)=14161
Total vcore-seconds taken by all map tasks=15033
Total vcore-seconds taken by all reduce tasks=14161
Total megabyte-seconds taken by all map tasks=15393792
Total megabyte-seconds taken by all reduce tasks=14500864
Map-Reduce Framework
Map input records=30
Map output records=29
Map output bytes=1067
Map output materialized bytes=1131
Input split bytes=102
Combine input records=0
Combine output records=0
Reduce input groups=11
Reduce shuffle bytes=1131
Reduce input records=29
Reduce output records=5
Spilled Records=58
Shuffled Maps =1
Failed Shuffles=0
Merged Map outputs=1
GC time elapsed (ms)=545
CPU time spent (ms)=4880
Physical memory (bytes) snapshot=424402944
Virtual memory (bytes) snapshot=3926773760
Total committed heap usage (bytes)=305135616
Shuffle Errors
BAD_ID=0
CONNECTION=0

IO_ERROR=0

WRONG_LENGTH=0

WRONG_MAP=0

WRONG_REDUCE=0

File Input Format Counters

Bytes Read=3905

File Output Format Counters

Bytes Written=201

hduser@ubuntu:~$ hadoop fs -cat /user/hduser/1b/p*

18/10/15 02:49:01 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

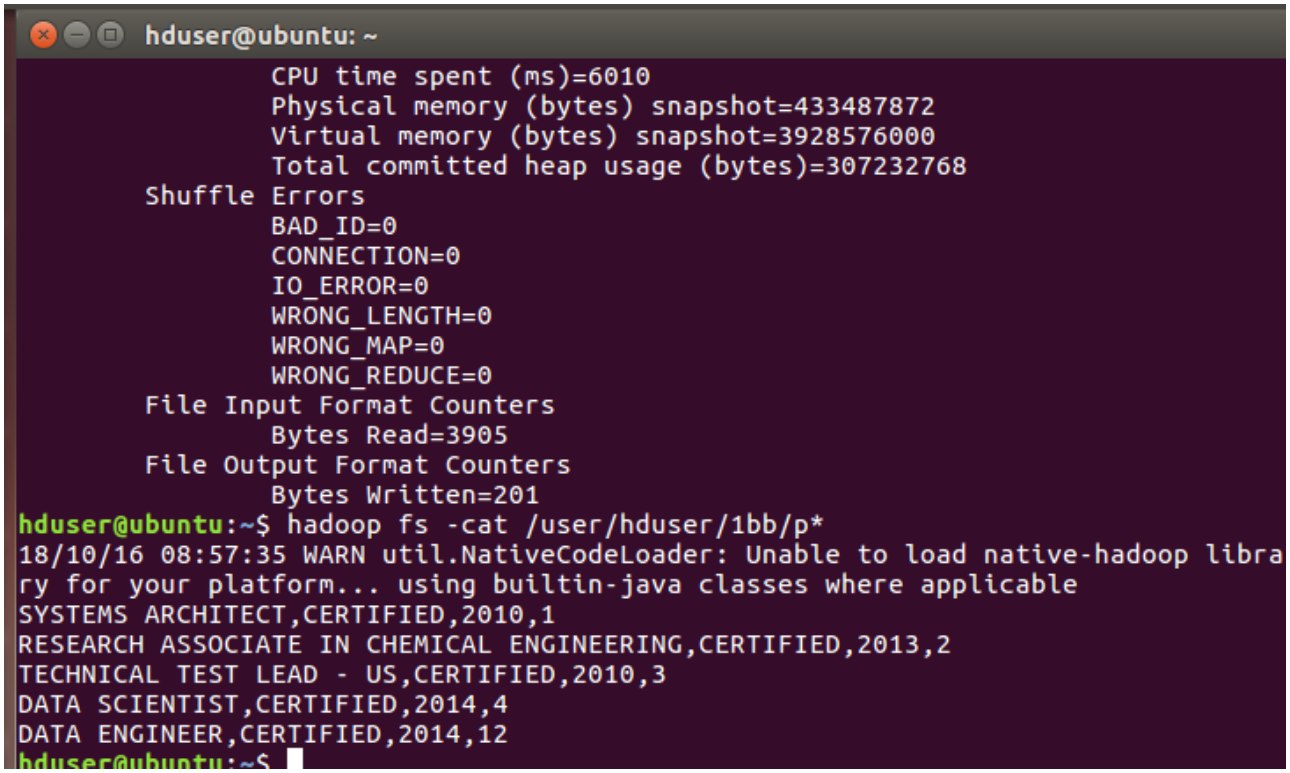
SYSTEMS ARCHITECT,CERTIFIED,2010,1

RESEARCH ASSOCIATE IN CHEMICAL ENGINEERING,CERTIFIED,2013,2

TECHNICAL TEST LEAD - US,CERTIFIED,2010,3

DATA SCIENTIST,CERTIFIED,2014,4

DATA ENGINEER,CERTIFIED,2014,12



2b) find top 5 locations in the US who have got certified visa for each year.

hadoop jar /home/ancy/Documents/jarfile/demo2bar.jar file:///home/ancy/Documents/newproinput \2b

18/10/15 02:57:33 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

18/10/15 02:57:36 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032

18/10/15 02:57:38 WARN mapreduce.JobSubmitter: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.

18/10/15 02:57:39 INFO input.FileInputFormat: Total input paths to process : 1

18/10/15 02:57:39 INFO mapreduce.JobSubmitter: number of splits:1

18/10/15 02:57:40 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1539594939608_0002

18/10/15 02:57:41 INFO impl.YarnClientImpl: Submitted application application_1539594939608_0002

18/10/15 02:57:41 INFO mapreduce.Job: The url to track the job: http://ubuntu:8088/proxy/application_1539594939608_0002/

18/10/15 02:57:41 INFO mapreduce.Job: Running job: job_1539594939608_0002

18/10/15 02:58:01 INFO mapreduce.Job: Job job_1539594939608_0002 running in uber mode : false

18/10/15 02:58:01 INFO mapreduce.Job: map 0% reduce 0%

18/10/15 02:58:20 INFO mapreduce.Job: map 100% reduce 0%

18/10/15 02:58:47 INFO mapreduce.Job: map 100% reduce 25%

18/10/15 02:58:48 INFO mapreduce.Job: map 100% reduce 100%

18/10/15 02:58:49 INFO mapreduce.Job: Job job_1539594939608_0002 completed successfully

18/10/15 02:58:49 INFO mapreduce.Job: Counters: 49

File System Counters

FILE: Number of bytes read=4646

FILE: Number of bytes written=530784

FILE: Number of read operations=0

FILE: Number of large read operations=0

FILE: Number of write operations=0

HDFS: Number of bytes read=102

HDFS: Number of bytes written=94

HDFS: Number of read operations=14

HDFS: Number of large read operations=0

HDFS: Number of write operations=8

Job Counters

Launched map tasks=1

Launched reduce tasks=4

Rack-local map tasks=1

Total time spent by all maps in occupied slots (ms)=15258

Total time spent by all reduces in occupied slots (ms)=92443

Total time spent by all map tasks (ms)=15258

Total time spent by all reduce tasks (ms)=92443

Total vcore-seconds taken by all map tasks=15258

Total vcore-seconds taken by all reduce tasks=92443

Total megabyte-seconds taken by all map tasks=15624192

Total megabyte-seconds taken by all reduce tasks=94661632

Map-Reduce Framework

Map input records=30

Map output records=28

Map output bytes=661

Map output materialized bytes=741

Input split bytes=102

Combine input records=0

Combine output records=0

Reduce input groups=4

Reduce shuffle bytes=741

Reduce input records=28

Reduce output records=4

Spilled Records=56

Shuffled Maps =4

Failed Shuffles=0

Merged Map outputs=4

GC time elapsed (ms)=5275

CPU time spent (ms)=10640

Physical memory (bytes) snapshot=905306112

Virtual memory (bytes) snapshot=9833820160

Total committed heap usage (bytes)=586678272

Shuffle Errors

BAD_ID=0

CONNECTION=0

IO_ERROR=0

WRONG_LENGTH=0

WRONG_MAP=0

WRONG_REDUCE=0

File Input Format Counters

Bytes Read=3905

File Output Format Counters

Bytes Written=94

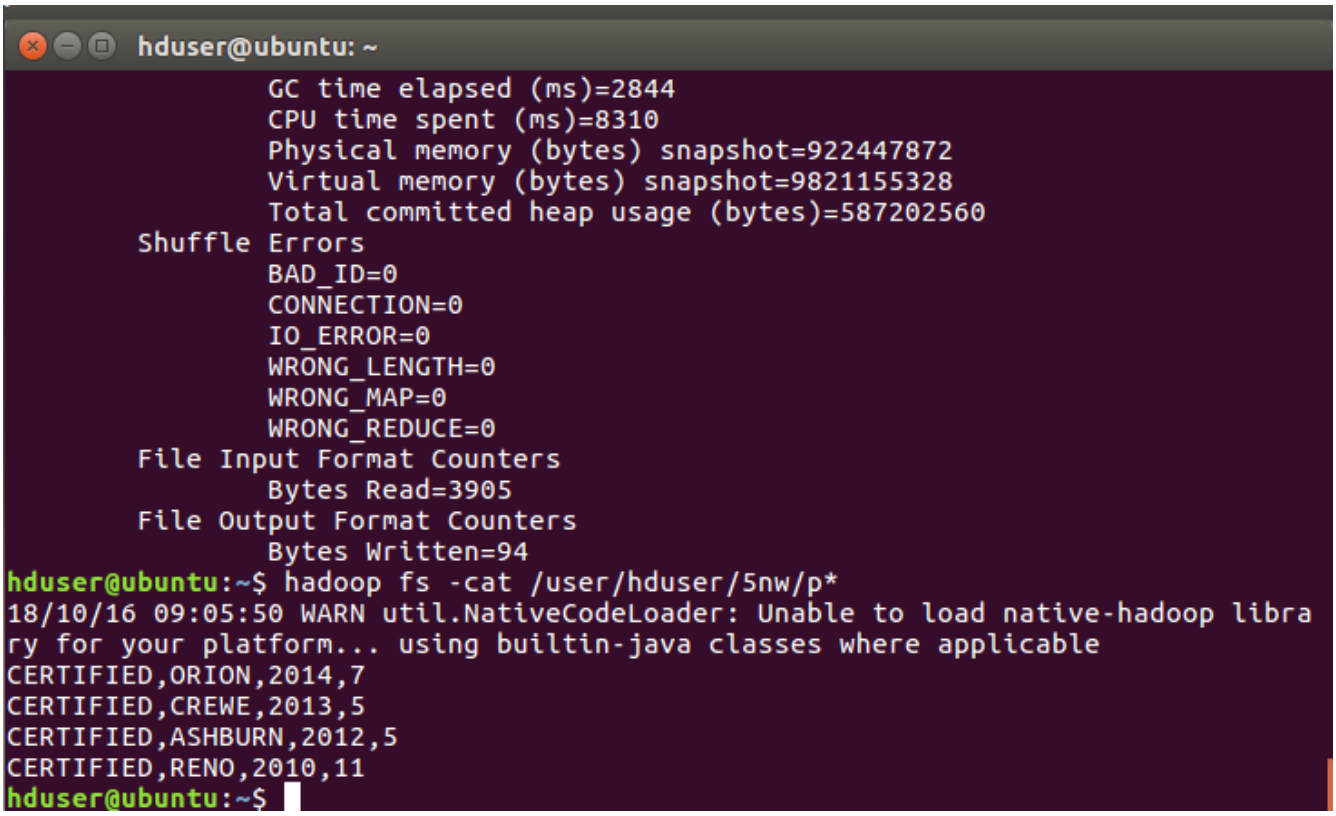
hduser@ubuntu:~$ hadoop fs -cat /user/hduser/2b/p*18/10/15 02:59:04 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

CERTIFIED,ORION,2014,7

CERTIFIED,CREWE,2013,5

CERTIFIED,ASHBURN,2012,5

CERTIFIED,RENO,2010,11



4)Which top 5 employers file the most petitions each year?

hadoop jar /home/ancy/Documents/jarfile/demo4jar.jar file:///home/ancy/Documents/newproinput \4b

18/10/15 03:02:38 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
18/10/15 03:02:42 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
18/10/15 03:02:45 WARN mapreduce.JobSubmitter: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
18/10/15 03:02:47 INFO input.FileInputFormat: Total input paths to process : 1
18/10/15 03:02:47 INFO mapreduce.JobSubmitter: number of splits:1
18/10/15 03:02:48 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1539594939608_0003
18/10/15 03:02:50 INFO impl.YarnClientImpl: Submitted application application_1539594939608_0003
18/10/15 03:02:51 INFO mapreduce.Job: The url to track the job: http://ubuntu:8088/proxy/application_1539594939608_0003/
18/10/15 03:02:51 INFO mapreduce.Job: Running job: job_1539594939608_0003
18/10/15 03:03:26 INFO mapreduce.Job: Job job_1539594939608_0003 running in uber mode : false
18/10/15 03:03:26 INFO mapreduce.Job: map 0% reduce 0%
18/10/15 03:03:46 INFO mapreduce.Job: map 100% reduce 0%
18/10/15 03:04:12 INFO mapreduce.Job: map 100% reduce 33%
18/10/15 03:04:13 INFO mapreduce.Job: map 100% reduce 67%
18/10/15 03:04:14 INFO mapreduce.Job: map 100% reduce 100%
18/10/15 03:04:16 INFO mapreduce.Job: Job job_1539594939608_0003 completed successfully
18/10/15 03:04:17 INFO mapreduce.Job: Counters: 49
	File System Counters
		FILE: Number of bytes read=5114
		FILE: Number of bytes written=425197
		FILE: Number of read operations=0
		FILE: Number of large read operations=0
		FILE: Number of write operations=0
		HDFS: Number of bytes read=102
		HDFS: Number of bytes written=289
		HDFS: Number of read operations=11
		HDFS: Number of large read operations=0
		HDFS: Number of write operations=6

Job Counters

Launched map tasks=1

Launched reduce tasks=3

Rack-local map tasks=1

Total time spent by all maps in occupied slots (ms)=16096

Total time spent by all reduces in occupied slots (ms)=68476

Total time spent by all map tasks (ms)=16096

Total time spent by all reduce tasks (ms)=68476

Total vcore-seconds taken by all map tasks=16096

Total vcore-seconds taken by all reduce tasks=68476

Total megabyte-seconds taken by all map tasks=16482304

Total megabyte-seconds taken by all reduce tasks=70119424

Map-Reduce Framework

Map input records=30

Map output records=29

Map output bytes=1133

Map output materialized bytes=1209

Input split bytes=102

Combine input records=0

Combine output records=0

Reduce input groups=15

Reduce shuffle bytes=1209

Reduce input records=29

Reduce output records=7

Spilled Records=58

Shuffled Maps =3

Failed Shuffles=0

Merged Map outputs=3

GC time elapsed (ms)=2037

CPU time spent (ms)=7660

Physical memory (bytes) snapshot=762241024

Virtual memory (bytes) snapshot=7862894592

Total committed heap usage (bytes)=501743616

Shuffle Errors

BAD_ID=0

CONNECTION=0

IO_ERROR=0

WRONG_LENGTH=0

WRONG_MAP=0

WRONG_REDUCE=0

File Input Format Counters

Bytes Read=3905

File Output Format Counters

Bytes Written=289

hduser@ubuntu:~$ hadoop fs -cat /user/hduser/4b/p*

18/10/15 03:05:06 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

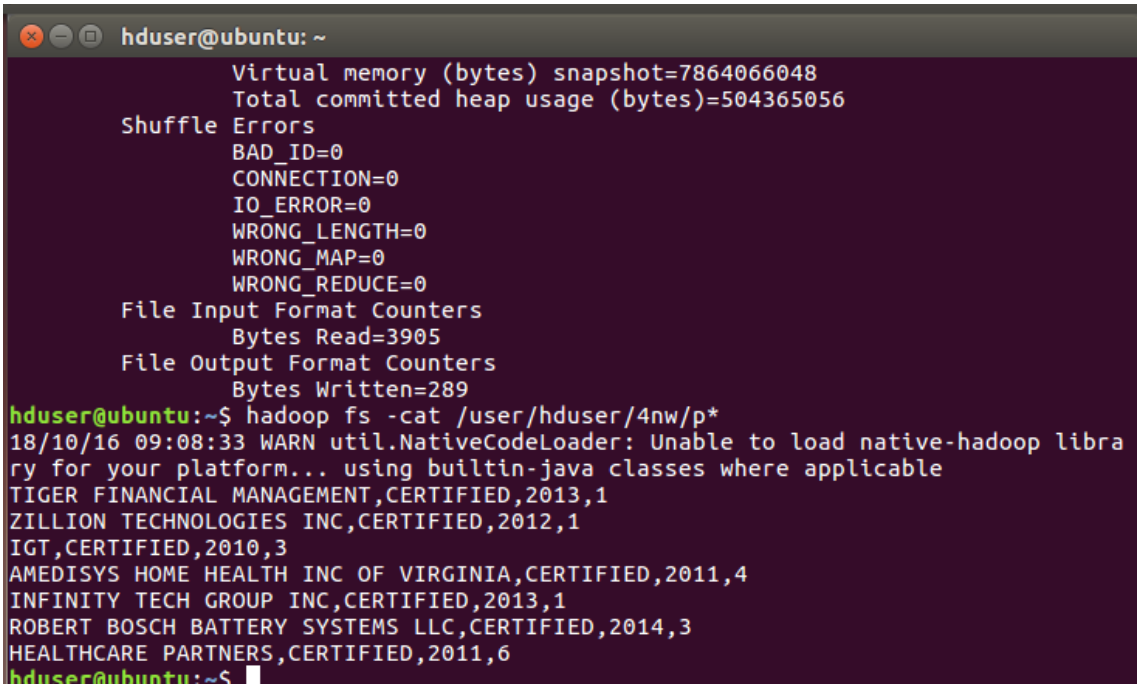
ZILLION TECHNOLOGIES INC,CERTIFIED,2012,1

IGT,CERTIFIED,2010,3

INFINITY TECH GROUP INC,CERTIFIED,2013,1

ROBERT BOSCH BATTERY SYSTEMS LLC,CERTIFIED,2014,3

HEALTHCARE PARTNERS,CERTIFIED,2011,6



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

hadoop jar /home/ancy/Documents/jarfile/demo5jar.jar file:///home/ancy/Documents/newproinput \
5b

18/10/15 03:08:02 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

18/10/15 03:08:07 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032

18/10/15 03:08:10 WARN mapreduce.JobSubmitter: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.

18/10/15 03:08:12 INFO input.FileInputFormat: Total input paths to process : 1

18/10/15 03:08:12 INFO mapreduce.JobSubmitter: number of splits:1

18/10/15 03:08:13 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1539594939608_0004

18/10/15 03:08:14 INFO impl.YarnClientImpl: Submitted application application_1539594939608_0004

18/10/15 03:08:15 INFO mapreduce.Job: The url to track the job: http://ubuntu:8088/proxy/application_1539594939608_0004/

18/10/15 03:08:15 INFO mapreduce.Job: Running job: job_1539594939608_0004

18/10/15 03:08:40 INFO mapreduce.Job: Job job_1539594939608_0004 running in uber mode : false

18/10/15 03:08:40 INFO mapreduce.Job: map 0% reduce 0%

18/10/15 03:08:53 INFO mapreduce.Job: map 100% reduce 0%

18/10/15 03:09:09 INFO mapreduce.Job: map 100% reduce 100%

18/10/15 03:09:10 INFO mapreduce.Job: Job job_1539594939608_0004 completed successfully

18/10/15 03:09:11 INFO mapreduce.Job: Counters: 49

File System Counters

FILE: Number of bytes read=5036

FILE: Number of bytes written=213655

FILE: Number of read operations=0

FILE: Number of large read operations=0

FILE: Number of write operations=0

HDFS: Number of bytes read=102

HDFS: Number of bytes written=201

HDFS: Number of read operations=5

HDFS: Number of large read operations=0

HDFS: Number of write operations=2

Job Counters

Launched map tasks=1

Launched reduce tasks=1

Rack-local map tasks=1

Total time spent by all maps in occupied slots (ms)=11002

Total time spent by all reduces in occupied slots (ms)=12416

Total time spent by all map tasks (ms)=11002

Total time spent by all reduce tasks (ms)=12416

Total vcore-seconds taken by all map tasks=11002

Total vcore-seconds taken by all reduce tasks=12416

Total megabyte-seconds taken by all map tasks=11266048

Total megabyte-seconds taken by all reduce tasks=12713984

Map-Reduce Framework

Map input records=30

Map output records=29

Map output bytes=1067

Map output materialized bytes=1131

Input split bytes=102

Combine input records=0

Combine output records=0

Reduce input groups=11

Reduce shuffle bytes=1131

Reduce input records=29

Reduce output records=5

Spilled Records=58

Shuffled Maps =1

Failed Shuffles=0

Merged Map outputs=1

GC time elapsed (ms)=521

CPU time spent (ms)=3320

Physical memory (bytes) snapshot=418230272

Virtual memory (bytes) snapshot=3928678400

Total committed heap usage (bytes)=308805632

Shuffle Errors

BAD_ID=0

CONNECTION=0

IO_ERROR=0

WRONG_LENGTH=0

WRONG_MAP=0

WRONG_REDUCE=0

File Input Format Counters

Bytes Read=3905

File Output Format Counters

Bytes Written=201

hduser@ubuntu:~$ hadoop fs -cat /user/hduser/5b/p*18/10/15 03:09:29 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

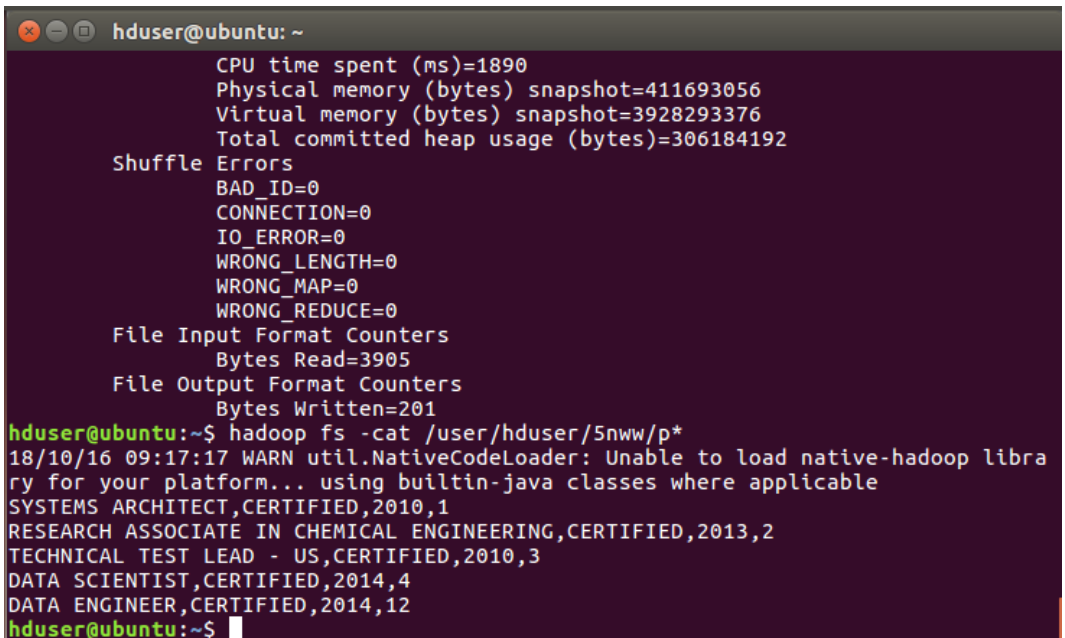
SYSTEMS ARCHITECT,CERTIFIED,2010,1

RESEARCH ASSOCIATE IN CHEMICAL ENGINEERING,CERTIFIED,2013,2

TECHNICAL TEST LEAD - US,CERTIFIED,2010,3

DATA SCIENTIST,CERTIFIED,2014,4

DATA ENGINEER,CERTIFIED,2014,12



USING HIVE

* IS THE NUMBER OF PETITIONS WITH DATA ENGINEER JOB TITLE INCREASING OVER TIME?(1a)

---

select count(case_status),job_title,year from h2b_applications where job_title="DATA ENGINEER" group by job_title,year;

3 DATA ENGINEER 2012

10 DATA ENGINEER 2014

26 DATA ENGINEER 2016

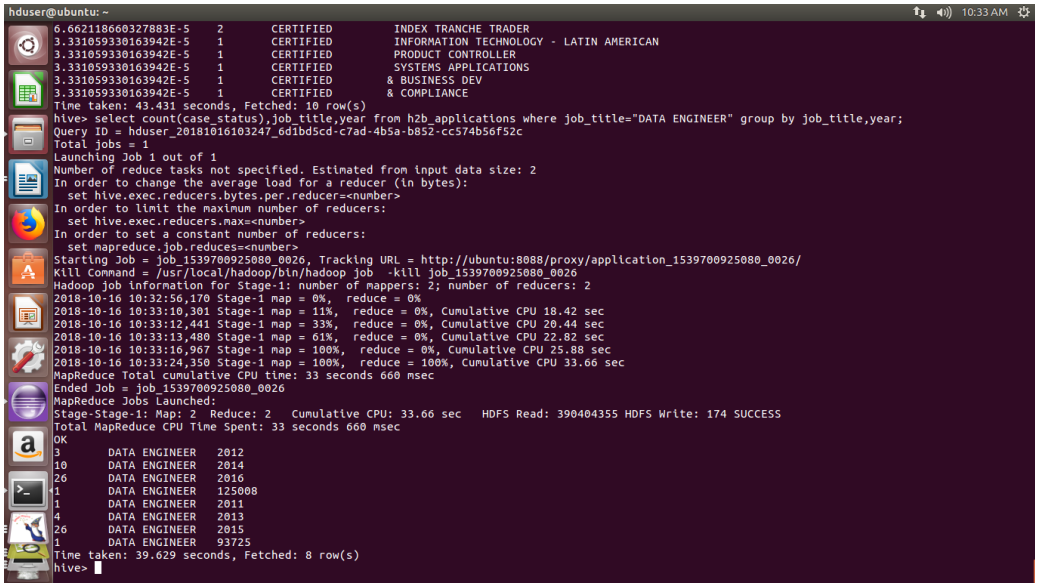
1 DATA ENGINEER 125008

1 DATA ENGINEER 2011

4 DATA ENGINEER 2013

26 DATA ENGINEER 2015

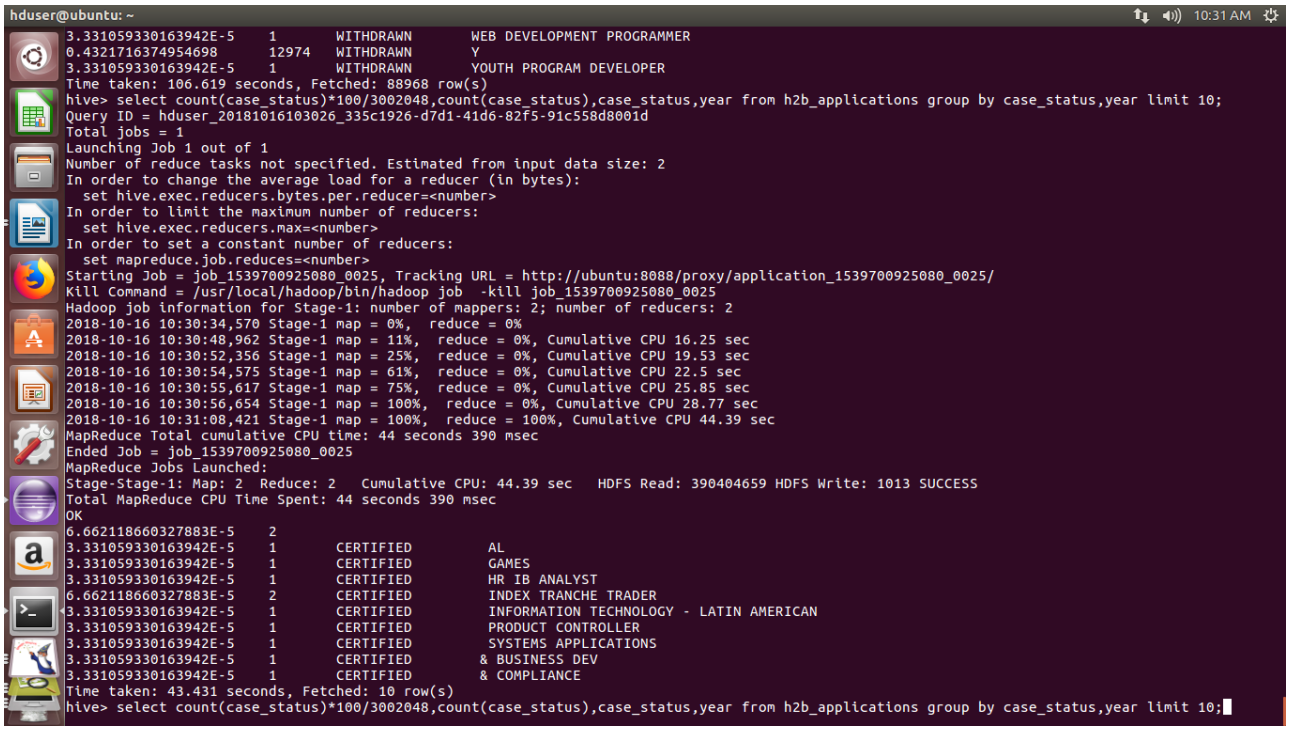
1 DATA ENGINEER 93725



*FIND THE PERCENTAGE AND THE COUNT OF EACH CASE STATUS ON TOTAL APPLICATIONS FOR EACH YEAR. CREATE A GRAPH DEPICTING THE PATTERN OF ALL THE CASES OVER THE PERIOD OF TIME.(6)

select count(case_status)*100/3002048,count(case_status),case_status,year from h2b_applications group by year,case_status limit 10;

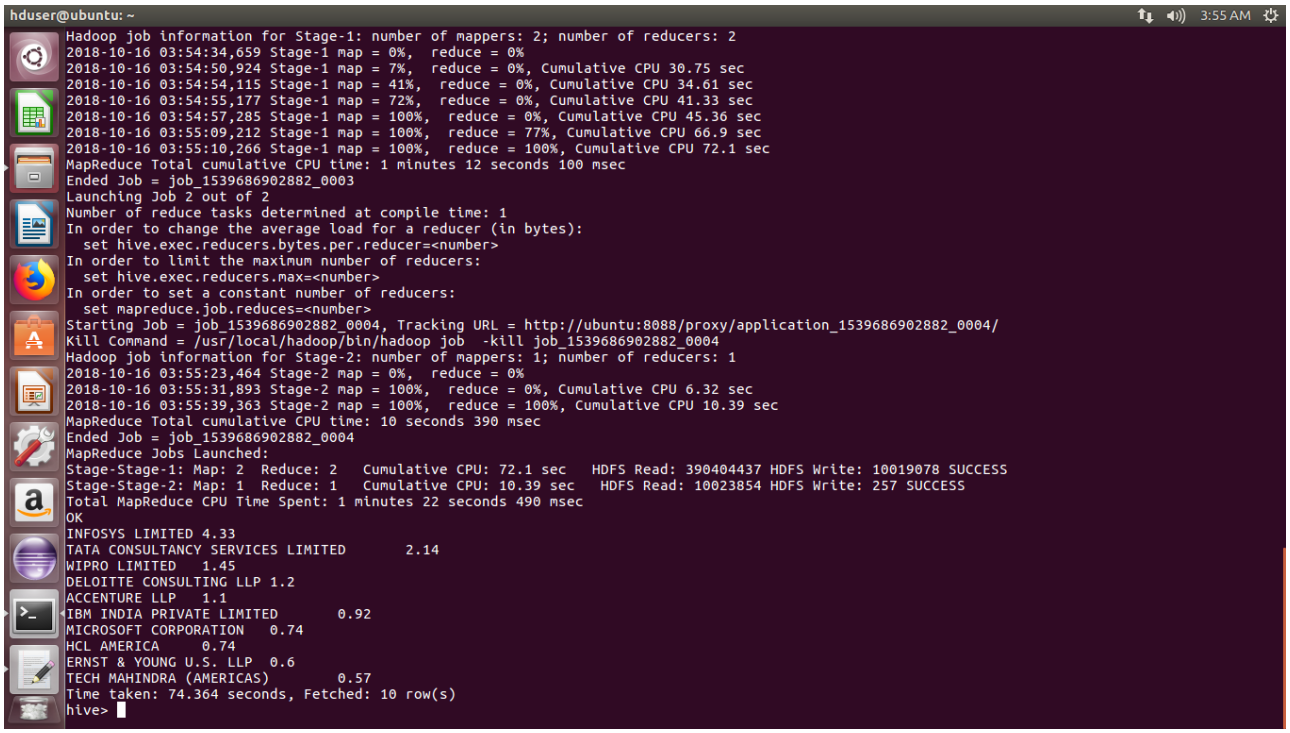
6.662118660327883E-5 2
3.331059330163942E-5 1 CERTIFIED AL
3.331059330163942E-5 1 CERTIFIED GAMES
3.331059330163942E-5 1 CERTIFIED HR IB ANALYST
6.662118660327883E-5 2 CERTIFIED INDEX TRANCHE TRADER
3.331059330163942E-5 1 CERTIFIED INFORMATION TECHNOLOGY - LATIN AMERICAN
3.331059330163942E-5 1 CERTIFIED PRODUCT CONTROLLER
3.331059330163942E-5 1 CERTIFIED SYSTEMS APPLICATIONS
3.331059330163942E-5 1 CERTIFIED & BUSINESS DEV
3.331059330163942E-5 1 CERTIFIED & COMPLIANCE



*WHICH ARE TOP TEN EMPLOYERS WHO HAVE THE HIGHEST SUCCESS RATE IN PETITIONS?(9)

select employer_name,ROUND(count(case_status)*100/3002048,2) as c from h2b_applications where case_status="CERTIFIED" or case_status="CERTIFIED_WITHDRAWN" group by employer_name order by c desc limit 10;

INFOSYS LIMITED 4.33
TATA CONSULTANCY SERVICES LIMITED 2.14
WIPRO LIMITED 1.45
DELOITTE CONSULTING LLP 1.2
ACCENTURE LLP 1.1
IBM INDIA PRIVATE LIMITED 0.92
MICROSOFT CORPORATION 0.74
HCL AMERICA 0.74
ERNST & YOUNG U.S. LLP 0.6
TECH MAHINDRA (AMERICAS) 0.57



*WHICH ARE THE TOP 10 JOB POSITIONS WHICH HAVE THE HIGHEST SUCCESS RATE IN PETITIONS?(10)

select job_title,ROUND(count(case_status)*100/3002048,2) as c from h2b_applications where case_status="CERTIFIED" or case_status="CERTIFIED_WITHDRAWN" group by job_title order by c desc limit 10;

PROGRAMMER ANALYST 3.81

SOFTWARE DEVELOPERS 3.51

Software Developers 3.5

APPLICATIONS 3.45

Computer Systems Analysts 3.19

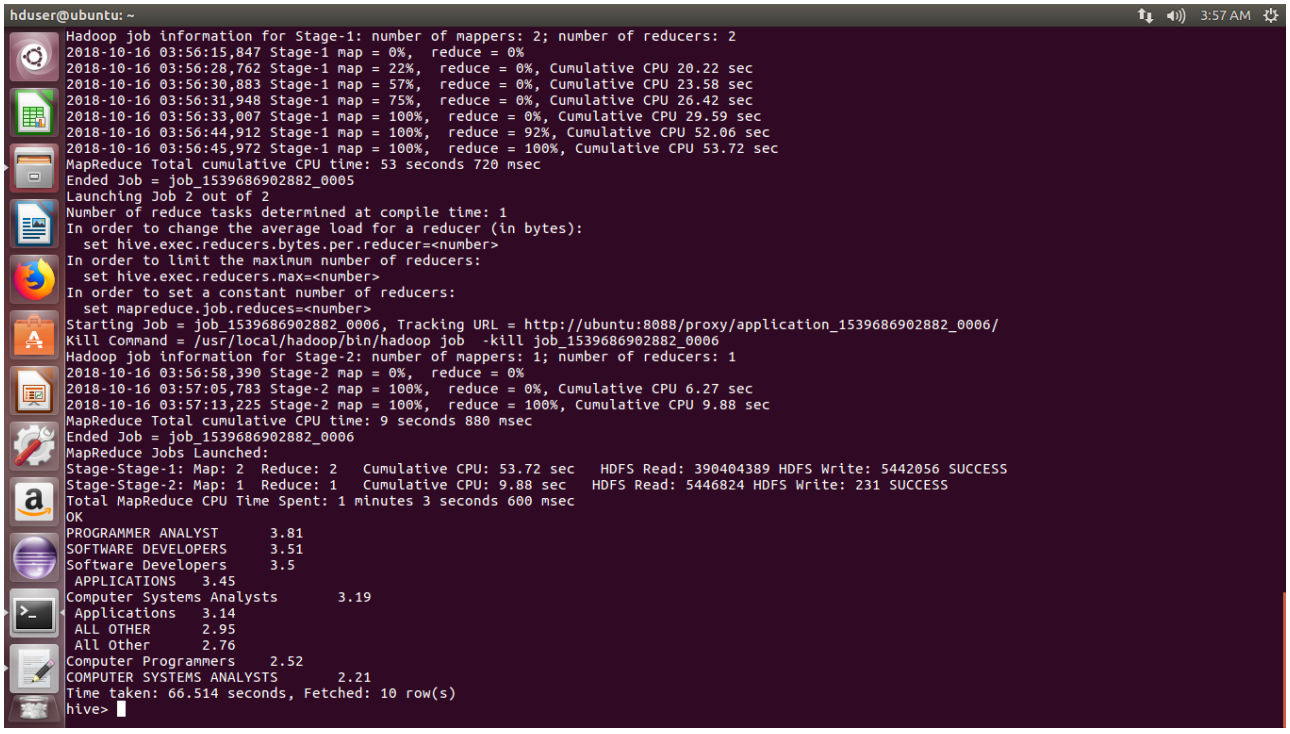
Applications 3.14

ALL OTHER 2.95

All Other 2.76

Computer Programmers 2.52

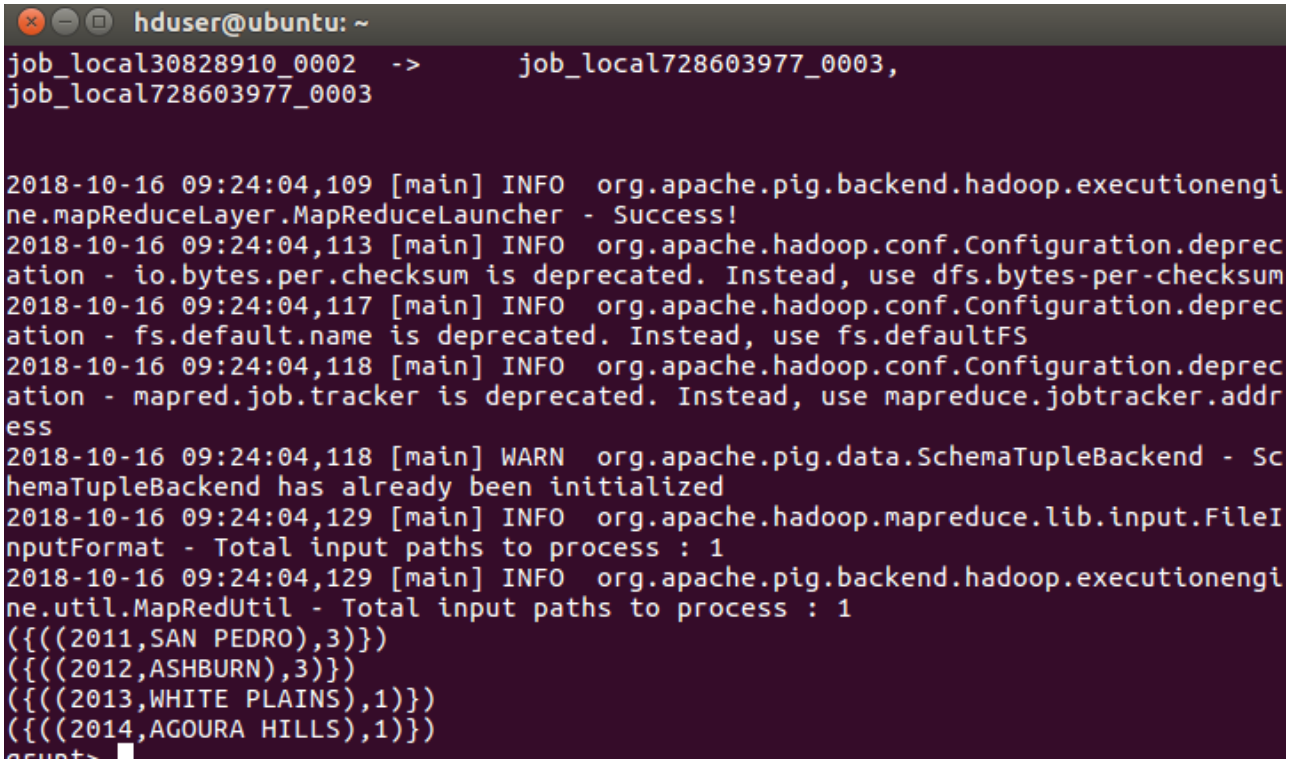
COMPUTER SYSTEMS ANALYSTS 2.21A



USING PIG

* WHICH PART OF THE US HAS THE MOST DATA ENGINEER JOBS FOR EACH YEAR? (2a)

```
A= load '/home/ancy/Documents/newproinput' using PigStorage (',') as (id,cs,empname,socname,soccode,jobtitle,ftpos,wage,year,worksite,lat,longi);
dump A;
b= foreach A generate $5,$8,$9;
dump b;
grunt> c= filter b by $0 == 'DATA ENGINEER';
dump c;
d= group c by ($1,$2);
e= foreach d generate group ,COUNT(c.$0);
e1= group e by $0.year;
f= foreach e1 { g= order e by $1 desc;h= limit g 1;generate h;};
({((2011,SAN PEDRO),3)})
({((2012,ASHBURN),3)})
({((2013,CREWE),2)})
({((2014,AGOURA HILLS),1)})
```



* WHICH INDUSTRY HAS THE MOST NUMBER OF DATA SCIENTIST POSITIONS?(3)

A= load '/home/ancy/Documents/newproinput' using PigStorage (',') as (id,cs,empname,socname,soccode,jobtitle,ftpos,wage,year,worksite,lat,longi);

dump A;

b= foreach A generate $2,$5;

dump b;

c= filter b by $1== 'DATA SCIENTIST';

dump c;

d= group c by $0;

dump d;

e= foreach d generate group ,COUNT(c.$1);

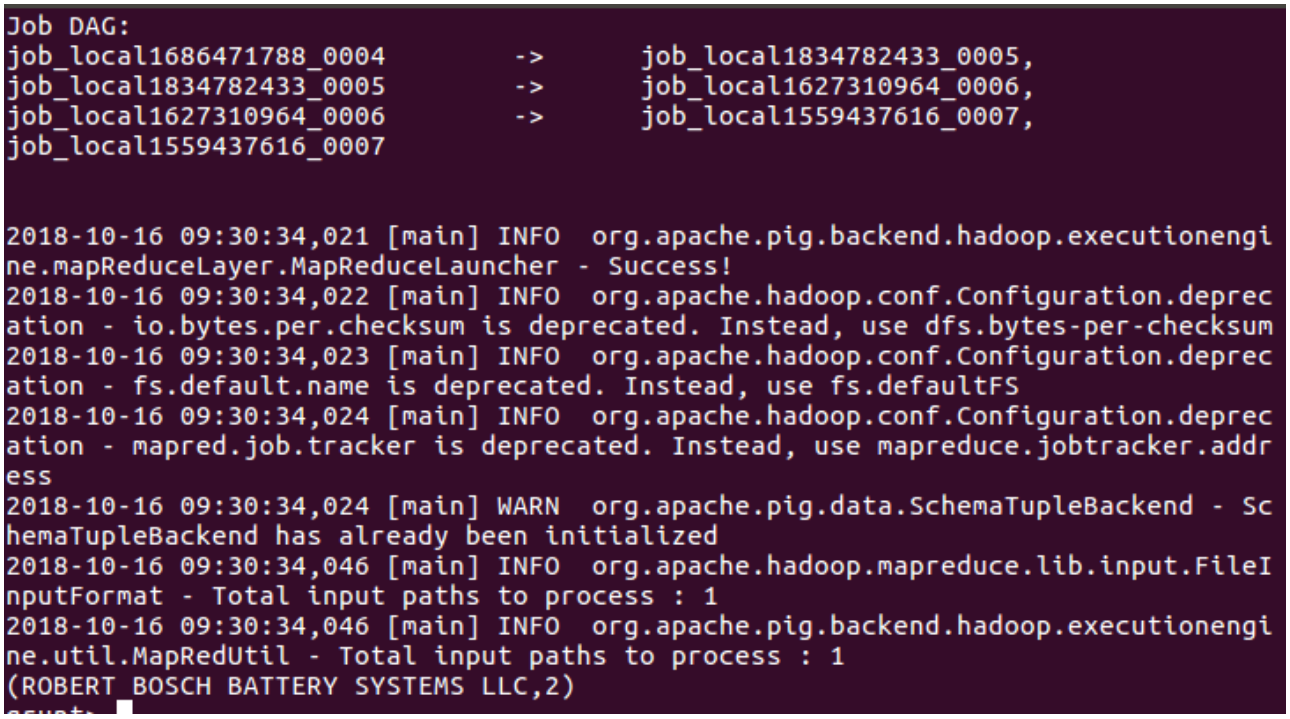
dump e;

f= ORDER e by $1 DESC;

dump f;

g= LIMIT f 1;

(ROBERT BOSCH BATTERY SYSTEMS LLC,2);



* CREATE A BAR GRAPH TO DEPICT THE NUMBER OF APPLICATIONS FOR EACH YEAR(7)

-

A= load '/home/ancy/Documents/newproinput' using PigStorage (',') as (id,cs,empname,socname,soccode,jobtitle,ftpos,wage,year,worksite,lat,longi);

dump A;

b= foreach A generate $1,$8;

dump b;

c= group b by $1;

dump c;

d= foreach c generate group,COUNT(b.$0);

dump d;

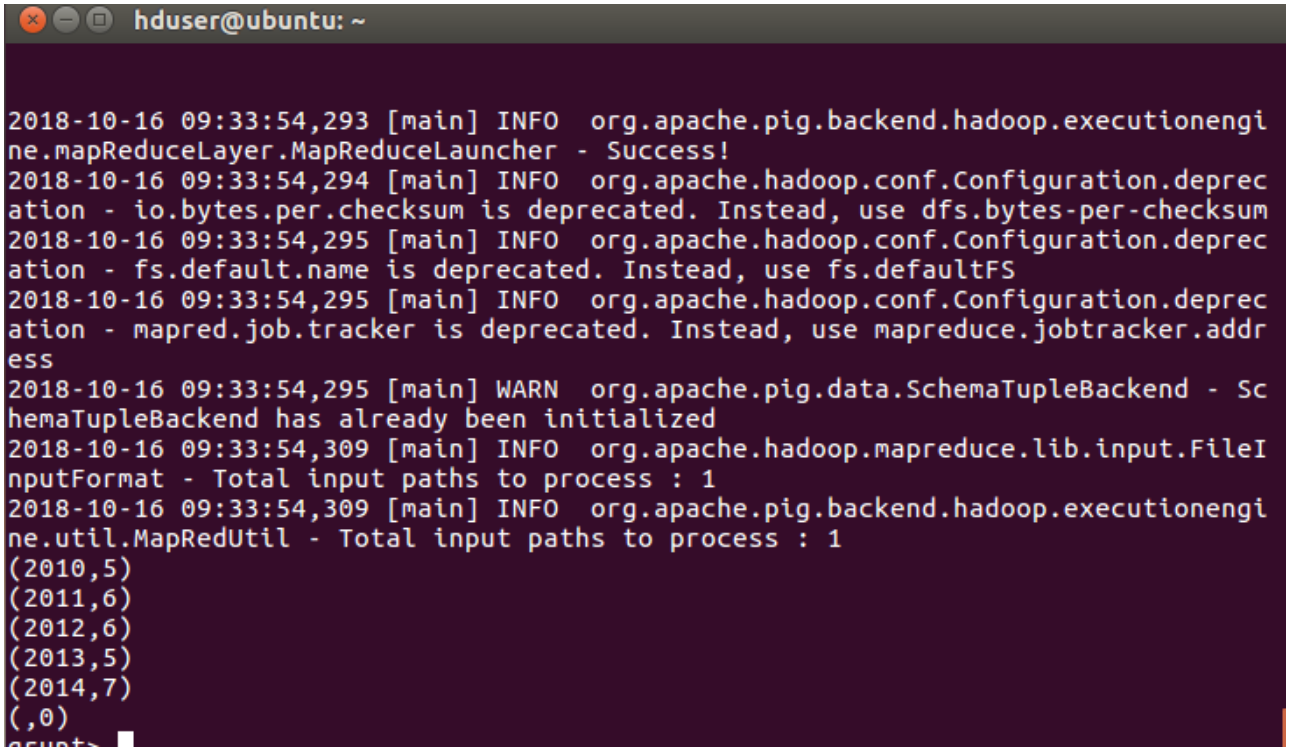
(2010,5)

(2011,6)

(2012,6)

(2013,6)

(2014,9)



* FIND THE AVERAGE PREVAILING WAGE FOR EACH JOB FOR EACH YEAR(8)

A= load '/home/ancy/Documents/newproinput' using PigStorage (',') as (id,cs,empname,socname,soccode,jobtitle,ftpos,wage,year,worksite,lat,longi);

dump A;

B= foreach A generate $5,$8,$7;

dump B;

C= group B by ($0,$1);

dump C;

D= foreach C generate group, AVG(B.$1);

dump D;

e= group D by $0.jobtitle;

dump e;

f= foreach e { g= order D by $1 desc;h= limit g 1;generate h;};

dump f;

({((DATA ENGINEER,2011),122153.25)})

({((DATA SCIENTIST,2014),73341.0)})

({((SYSTEMS ARCHITECT,2010),90501.0)})

({((PHYSICAL THERAPIST,2014),71094.0)})

({((POSTDOCTORAL ASSOCIATE,2010),62171.0)})

({((PHYSICIAN - HOSPITALIST,2014),139173.0)})

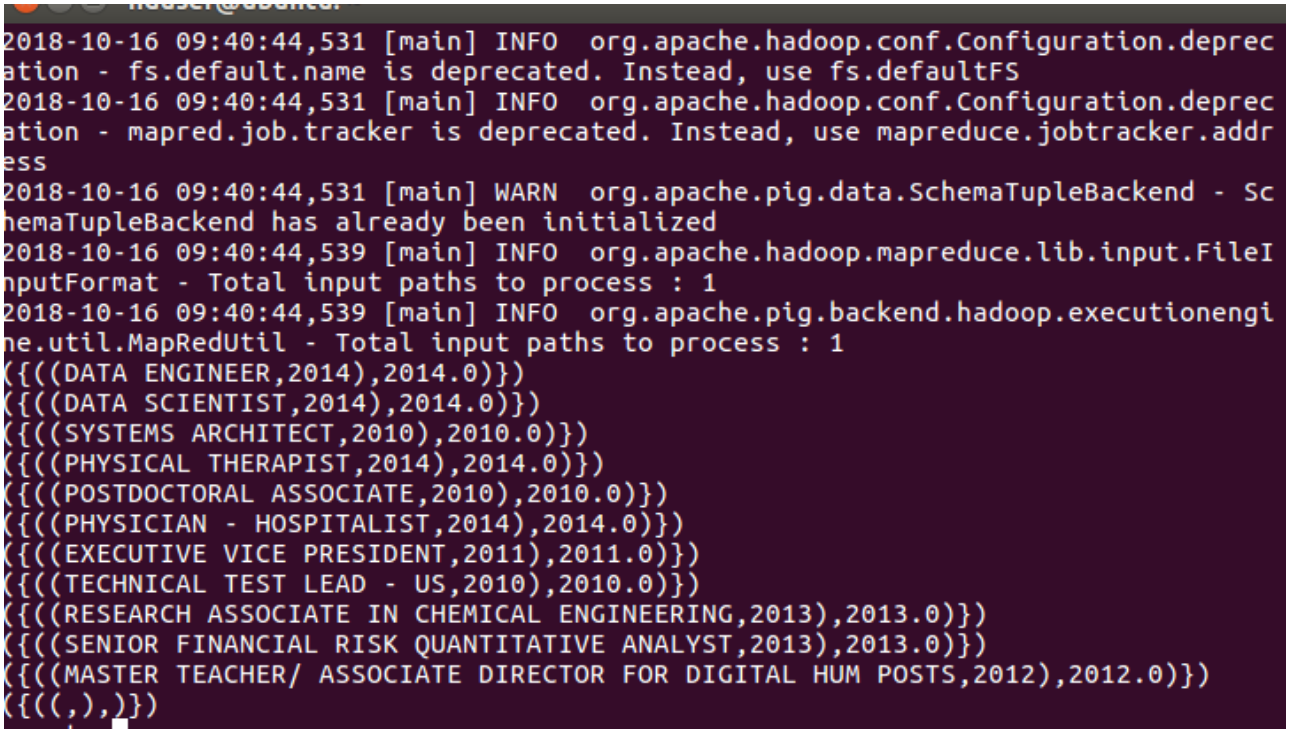
({((EXECUTIVE VICE PRESIDENT,2011),139173.0)})

({((TECHNICAL TEST LEAD - US,2010),72072.0)})

({((RESEARCH ASSOCIATE IN CHEMICAL ENGINEERING,2013),37440.0)})

({((SENIOR FINANCIAL RISK QUANTITATIVE ANALYST,2013),70283.0)})

({((MASTER TEACHER/ ASSOCIATE DIRECTOR FOR DIGITAL HUM POSTS,2012),56908.8)})



Using SQOOP

- insert overwrite local directory '/hanoi'row format delimited fields terminated by ',' select job_title,ROUND(count(case_status)*100/3002048,2) as c from h2b_applications where case_status="CERTIFIED" or case_status="CERTIFIED_WITHDRAWN" group by job_title order by c desc limit 10;

- sqoop export --connect jdbc:mysql://localhost/college --username root --password 'root' --table h1b --export-dir /hanoi/000000_0 --input-fields-terminated-by ',' ;

```
Database changed
mysql> select  * from h1b;
+----------------------+-----------------+
| job_title            | prevailing_wage |
+----------------------+-----------------+
| PROGRAMMER ANALYST   |            3.81 |
| SOFTWARE DEVELOPERS  |            3.51 |
| Software Developers  |             3.5 |
|  Applications        |            3.14 |
|  ALL OTHER           |            2.95 |
|  All Other           |            2.76 |
```

SHELL SCRIPT

Using shell script hive,pig,mapreduce ,sqoopjobs can be done in single coding itself

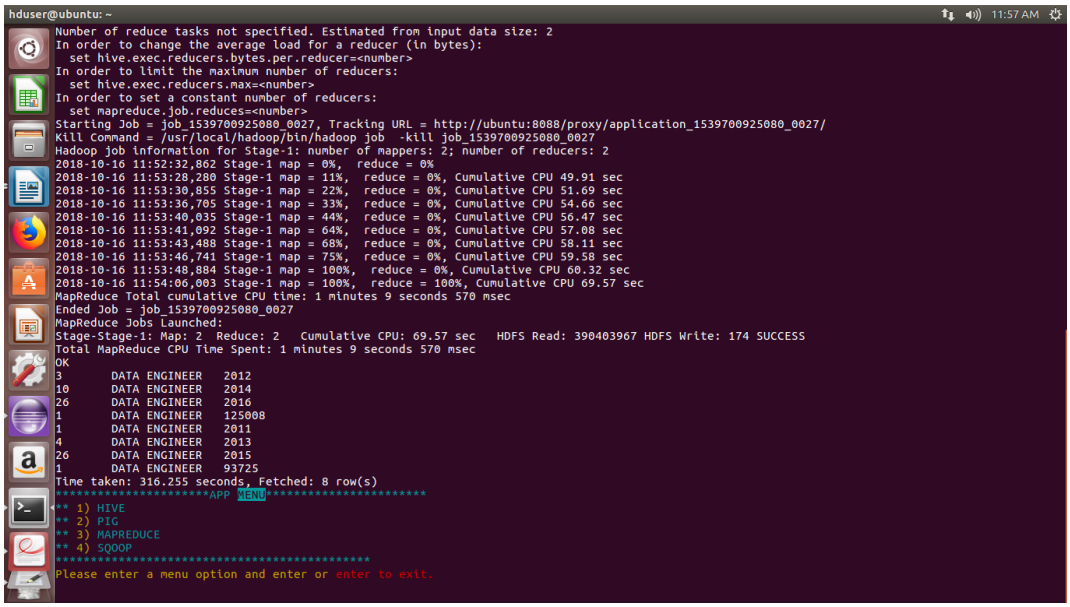
```
#!/bin/bash
show_menu()
{
   NORMAL=`echo "\033[m"`
   MENU=`echo "\033[36m"` #Blue
   NUMBER=`echo "\033[33m"` #yellow
   FGRED=`echo "\033[41m"`
   RED_TEXT=`echo "\033[31m"`
   ENTER_LINE=`echo "\033[33m"`
         echo  -e  "${MENU}**********************APP  MENU**********************${
NORMAL}"
   echo -e "${MENU}**${NUMBER} 1)${MENU} HIVE ${NORMAL}"
   echo -e "${MENU}**${NUMBER} 2)${MENU} PIG ${NORMAL}"
   echo -e "${MENU}**${NUMBER} 3)${MENU} MAPREDUCE ${NORMAL}"
   echo -e "${MENU}**${NUMBER} 4)${MENU} SQOOP ${NORMAL}"

   echo -e "${MENU}*********************************************${NORMAL}"
    echo -e "${ENTER_LINE}Please enter a menu option and enter or ${RED_TEXT}enter to exit. ${
NORMAL}"
   read opt
}
function option_picked()
{
   COLOR='\033[01;31m' # bold red
   RESET='\033[00;00m' # normal white
   MESSAGE="$1"  #modified to post the correct option selected
   echo -e "${COLOR}${MESSAGE}${RESET}"
}

function getpinCodeBank(){
```

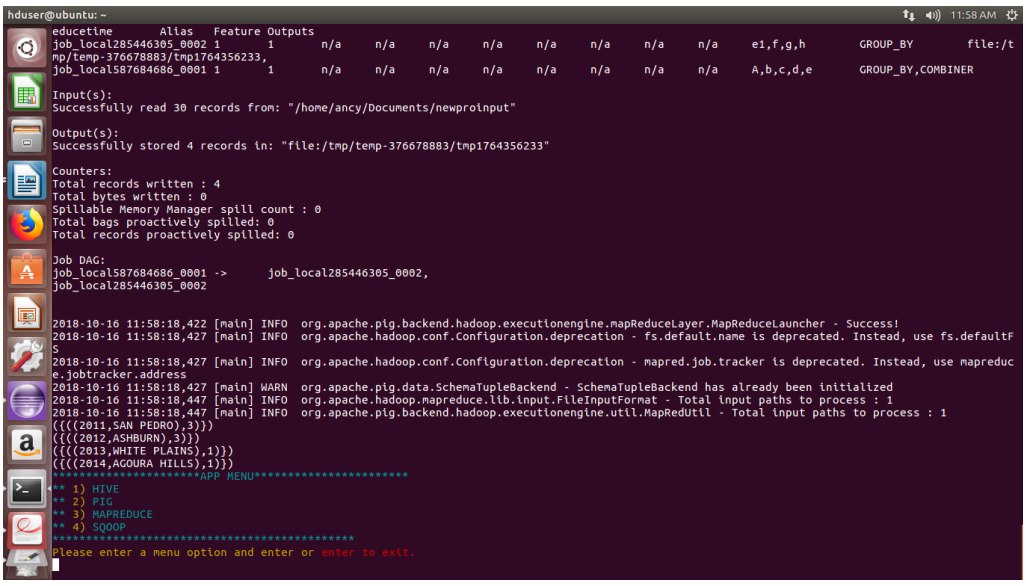
```
        echo "in getPinCodebank"
        echo $1
        echo $2
        #hive -e "Select * from AppData where PinCode = $1 AND Bank = '$2'"
}

clear
show_menu
while [ opt != " ]
   do
   if [[ $opt = "" ]]; then
        exit;
   else
      case $opt in
      1) clear;
            echo -e "${MENU}**${NUMBER} 1)${MENU} 1a Is the number of petitions with Data Engineer job title increasing over time? ${NORMAL}"
            echo -e "${MENU}**${NUMBER} 2)${MENU} 6 Find the percentage and the count of each case status on total applications for each year${NORMAL}"
            echo -e "${MENU}**${NUMBER} 3)${MENU} 9 Which are top ten employers who have the highest success rate in petitions?
${NORMAL}"
            echo -e "${MENU}**${NUMBER} 4)${MENU} 10 Which are the top 10 job positions which have the highest success rate in petitions? ${NORMAL}"
            read ques
          case $ques in


       1) clear;
                hive -e "select count(case_status),job_title,year from niit.h2b_applications where job_title='DATA ENGINEER' group by job_title,year"
                ;;


       2) clear;
            hive -e "select case_status,year,count(case_status)*100/3002048,count(case_status) from niit.h2b_applications group by        case_status,year limit 10"
                ;;


       3) clear;
                hive -e "select job_title,ROUND(count(case_status)*100/3002048,2) as c from niit.h2b_applications where case_status= 'CERTIFIED' or case_status='CERTIFIED_WITHDRAWN' group by job_title order by c desc limit 10"

;;
       4) clear;
```

```
                hive -e "select job_title,ROUND(count(case_status)*100/3002048,2) as c from niit.h2b_applications where case_status= 'CERTIFIED'or case_status='CERTIFIED_WITHDRAWN'group by job_title order by c desc limit 10"
                ;;

        *) clear;
                echo "DEFAULT"
  esac
show_menu;
;;


        2) clear;
           echo -e "${MENU}**${NUMBER} 1)${MENU} 2a Which part of the US has the most Data Engineer jobs for each year? ${NORMAL}"
            echo -e "${MENU}**${NUMBER} 2)${MENU} 3 Which industry has the most number of Data Scientist positions?
${NORMAL}"
            echo -e "${MENU}**${NUMBER} 3)${MENU} 7 Create a bar graph to depict the number of applications for each year${NORMAL}"
            echo -e "${MENU}**${NUMBER} 4)${MENU} 8 Find the average Prevailing Wage for each Job for each Year (take part time and full time separate)${NORMAL}"
            read ques1
          case $ques1 in

         1) clear;
                pig -x local /home/ancy/Documents/2a.pig
                ;;
        2) clear;
                pig -x local /home/ancy/Documents/3.pig

                ;;
         3) clear;
                pig -x local /home/ancy/Documents/7.pig

                        ;;

          4) clear;
                pig -x local /home/ancy/Documents/8.pig

                        ;;

        *) clear;
                echo "DEFAULT"
  esac
show_menu;
;;
```

```
                3) clear;
                    echo -e "${MENU}**${NUMBER} 1)${MENU} 1b Find top 5 job titles who are having highest growth in applications. ${NORMAL}"
                    echo -e "${MENU}**${NUMBER} 2)${MENU} 2b find top 5 locations in the US who have got certified visa for each year.
${NORMAL}"
                    echo -e "${MENU}**${NUMBER} 3)${MENU} 4 Which top 5 employers file the most petitions each year?${NORMAL}"
                    echo -e "${MENU}**${NUMBER} 4)${MENU} 5 Find the most popular top 10 job positions for H1B visa applications for each year?${NORMAL}"
            read ques2
          case $ques2 in

                1) clear;

    hadoop jar /home/ancy/Documents/jarfile/demo1bar.jar file:///home/ancy/Documents/newproinput \
1b
                    hadoop fs -cat /user/hduser/1b/p*
                    ;;
                2) clear;

hadoop jar /home/ancy/Documents/jarfile/demo2bar.jar file:///home/ancy/Documents/newproinput \2b
                    hadoop fs -cat /user/hduser/2b/p*
                    ;;
                3) clear;

 hadoop jar /home/ancy/Documents/jarfile/demo4jar.jar file:///home/ancy/Documents/newproinput \4b
                    hadoop fs -cat /user/hduser/4b/p*
                    ;;
                4) clear;
hadoop jar /home/ancy/Documents/jarfile/demo5jar.jar file:///home/ancy/Documents/newproinput \5b
                    hadoop fs -cat /user/hduser/4b/p*
                        ;;
                *) clear;
                        echo "DEFAULT"
esac
show_menu;
;;
                4) clear;
insert overwrite local directory '/hanoi'row format delimited fields terminated by ',' select job_title,ROUND(count(case_status)*100/3002048,2) as c from h2b_applications where case_status="CERTIFIED" or case_status="CERTIFIED_WITHDRAWN" group by job_title order by c desc limit 10;

sqoop export --connect jdbc:mysql://localhost/college --username root --password 'root' --table h1b --export-dir /hanoi/000000_0 --input-fields-terminated-by ','
;;
esac
show_menu;
```

```
;;


                \n exit;
        ;;

        *) clear;
        option_picked "Pick an option from the menu";
        show_menu;
        ;;
    esac
fi

done
```





Conclusion

Hence, in conclusion to this Big Data tutorial, we can say that Apache Hadoop is the most popular and powerful big data tool. Big Data stores huge amount of data in the distributed manner and processes the data in parallel on a cluster of nodes. It provides the world’s most reliable storage layer- HDFS. Batch processing engine MapReduce and Resource management layer- YARN. 4 daemons (NameNode, datanode, node manager, resource manager) run in Hadoop to ensure Hadoop functionality.