**Big Data**

**Big Data:** Big Data is the term for huge amount of data i.e., the data beyond the storage capacity and which is beyond the process power is said to be a Big Data. Where Data is nothing but information.

The data can be Structure data, Semi structure data and unstructured data. Structure data is nothing but relational data. The data is in the form of tables i.e. rows and columns. The searching mechanism will be easy. Semi structure data is XML data. XML stands for extensible markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. Unstructured data can be any format other than structured data and semi structured data i.e. the data is in word, pdf, text, media logs like images, audios and videos, calls, tweets, net surf, .

**This Big Data is generated from different sectors like:**

1. CC Camera (Closed circuit for video surveillance)
2. Black Box Data(Air Lines)
3. Social Media Data(Face book and twitter)
4. Stock Exchange Data (share market)
5. Transport Data (vehicle speed, traffic)
6. Search Engine Data(Google Search and Yahoo Search)
7. Online Shopping Data (Amazon, Flipkart)
8. Hospitality Data(Maintaining patients data)

In 1990, we used to use 20GB to 40GB Hard Disk capacity with 64 to 128MB of RAM. At that time data transmission speed is upto 10kbps. Whereas in 2017, we are using 500GB to 2TB Hard Disk capacity with 4 to 16GB of RAM. AT this time data transmission speed is upto 100mbps.

**To get a better understanding of what Big Data is, it is often described using 5 Vs:**

1. Volume: Refers to the vast amounts of data generated every second. We are not talking Terabytes but Zettabytes or Brontobytes. If we take all the data generated in the world between the beginning of time and 2008, the same amount of data will soon be generated every minute. This makes most data sets too large to store and analyse using traditional database technology. New big data tools use distributed systems so that we can store and analyse data across databases that are dotted around anywhere in the world.

2. Variety: Refers to the different types of data we can now use. In the past we only focused on structured data that neatly fitted into tables or relational databases, such as financial data. In fact, 80% of the world’s data is unstructured (text, images, video, voice, etc.) With big data technology we can now analyse and bring together data of different types such as messages, social media conversations, photos, sensor data, video or voice recordings.

3. Velocity: Refers to the speed at which new data is generated and the speed at which data moves around. Just think of social media messages going viral in seconds. Technology allows us now to analyse the data while it is being generated (sometimes referred to as in-memory analytics), without ever putting it into databases.

4. 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: Then there is another V to take into account when looking at Big Data: Value! Having access to big data is no good unless we can turn it into value. Companies are starting to generate amazing value from their big data.

**To rectify the issues of Big Data there are challenges given below:**

1. Capturing data: Taking the data logs.
2. Curation: The main purpose of data curation is to ensure that data is reliably retrievable for future research purposes or reuse.
3. Storage: Where we are going to store the data is said to be storage.
4. Searching: Searching a piece of information in a data or to find the data.
5. Sharing: sharing the data
6. Transfer: transferring of data
7. Analysis: analyzing of data
8. Presentation: presenting the data which we want. It is nothing but retrieving of data.

**Benefits of Big Data:**

* Using the information kept in the social network like Face book, the marketing agencies are learning about the response for their campaigns, promotions, and other advertising mediums.
* Using the information in the social media like preferences and product perception of their consumers, product companies and retail organizations are planning their production.
* Using the data regarding the previous medical history of patients, hospitals are providing better and quick service.

**Data Technologies:**

Big data technologies are important in providing more accurate analysis, which may lead to more concrete decision-making resulting in greater operational efficiencies, cost reductions, and reduced risks for the business.

To harness the power of big data, you would require an infrastructure that can manage and process huge volumes of structured and unstructured data in real-time and can protect data privacy and security.

There are various technologies in the market from different vendors including Amazon, IBM, Microsoft, etc., to handle big data. While looking into the technologies that handle big data, we examine the following two classes of technology:

1. Operational Big Data
2. Analytical Big Data

1. Operational Big Data: These include systems like MongoDB that provide operational capabilities for real-time, interactive workloads where data is primarily captured and stored.

NoSQL Big Data systems are designed to take advantage of new cloud computing architectures that have emerged over the past decade to allow massive computations to be run inexpensively and efficiently. This makes operational big data workloads much easier to manage, cheaper, and faster to implement.

2. Analytical Big Data: These includes systems like Massively Parallel Processing (MPP) database systems and MapReduce that provide analytical capabilities for retrospective and complex analysis that may touch most or all of the data.

MapReduce provides a new method of analyzing data that is complementary to the capabilities provided by SQL, and a system based on MapReduce that can be scaled up from single servers to thousands of high and low end machines.

**Hadoop**

**Hadoop:**

* Hadoop is an open source framework developed by Apache Software Foundation.
* Hadoop framework can store large amount of data and variety of the data.
* Hadoop framework can process the large amount of data with very high speed.
* Hadoop framework will store the data in terms of files. The file system format which is followed by the hadoop framework is HDFS (Hadoop Distributed File System)
* To process the data which is present inside hadoop framework we use MapReduce programming models.
* To implement the Business Logics by using MapReduce, time convert and development cost also increases.
* To overcome the above problem, we can use hadoop eco system which lies on the top of the hadoop.

(Or)

* Hadoop is an open source, distributed, batch processing and fault-tolerance system. Which is capable of storing huge amount of data [TB, PB, ZB etc] along with processing on the same amount of data.
* Hadoop can easily perform parallel programming model.
* Hadoop framework consists of two main core components;

1. HDFS
2. MapReduce
3. HDFS: It is responsible for storing massive amount of data on the cluster. Cluster is nothing but group of systems with commodity hardware are connected with each other by the help of network is said to be cluster.
4. MapReduce: It is responsible for processing massive amount of data on the cluster.

* HDFS & MapReduce capabilities are its kernel for Hadoop.

**Hadoop’s History:**

* Hadoop was created by Doug Cutting & Michael J. Cafarella. Doug Cutting, who was working at Yahoo at that time, named it after his son’s toy elephant. It was originally developed to support distribution for the Nutch Search Engine Project.
* Hadoop is based on work done by Google in the late 1990’s and early 2000’s. Specially, on papers describing the Google File System (GFS) published in 2003 and MapReduce is published in 2004.

**How Does Hadoop Work?**

It is quite expensive to build bigger servers with heavy configurations that handle large scale processing, but as an alternative, you can tie together many commodity computers with single-CPU, as a single functional distributed system and practically, the clustered machines can read the dataset in parallel and provide a much higher throughput. Moreover, it is cheaper than one high-end server. So this is the first motivational factor behind using Hadoop that it runs across clustered and low-cost machines.

Hadoop runs code across a cluster of computers. This process includes the following core tasks that Hadoop performs:

* Data is initially divided into directories and files. Files are divided into uniform sized blocks of 128M and 64M (preferably 128M).
* These files are then distributed across various cluster nodes for further processing.
* HDFS, being on top of the local file system, supervises the processing.
* Blocks are replicated for handling hardware failure.
* Checking that the code was executed successfully.
* Performing the sort that takes place between the map and reduce stages.
* Sending the sorted data to a certain computer.
* Writing the debugging logs for each job.

**Advantages of Hadoop:**

* Hadoop framework allows the user to quickly write and test distributed systems. It is efficient, and it automatic distributes the data and work across the machines and in turn, utilizes the underlying parallelism of the CPU cores.
* Hadoop does not rely on hardware to provide fault-tolerance and high availability (FTHA), rather Hadoop library itself has been designed to detect and handle failures at the application layer.
* Servers can be added or removed from the cluster dynamically and Hadoop continues to operate without interruption.
* Another big advantage of Hadoop is that apart from being open source, it is compatible on all the platforms since it is Java based.

**Features of Hadoop:**

1. Horizontally Scaling: Hadoop framework supports unlimited amount horizontally scaling.
2. Nodes: Adding the nodes to the existing cluster is known as Horizontally Scaling because of unlimited amount.

NameNode: Name node is always configuring in master machine. There will be Single Name node. The job of name node is monitoring and giving job to a Data Node. Hence Namenode knows where data is stored and processing done by data node.

Secondary Namenode: Secondary Name node is always configuring in master machine. There will be Single Name node. Incase, name node fails then no need to worry because same data is present in secondary name node.

DataNode: Data node is always configuring in slave machine. There will be Multiple

Data node.

Note: Name node, Secondary name node and Data node are connected has VPN (Virtual

Private Network).

1. High qualibility of data even in node failure:

* In a hadoop frame work files will be divides in the form of blocks and those blocks will be replicated among all the data nodes of the hadoop cluster.
* In hadoop frame work blocks will be replicated according yo Rack awareness mechanism.
* The default block size in hadoop frame work is “64 MB”. If we want we can increase the block size according to the multiples of 64MB i.e. 128MB, 256MB.
* By default replication factor in hadoop frame work is “1” (original copy). If we want increase the replication factor. If replication factor is “n” than original copy (1+duplicate) copy “n-1”. The maximum replication factor is equally to the size of the hadoop cluster (number of data nodes).
* Even though any one of the data node is fail in hadoop cluster. Data is available for hadoop cluster because it maintains the replies cost of data.

1. Moving the completions logic to the data rather than moving the data to the completion logic:

* Hadoop framework will moves the completion logic to the data nodes by creating the cluster of the completion logic order to process the data.
* Because of above approach, we choice the parallel distribute processing.
* Because of parallel distributed processing in hadoop the time taken to process the data is very less.

1. Communication and Synchronization:

* While processing the data in hadoop cluster always communication and synchronization will take place among all the data.

1. Less network bandwidth:

* In order to provide communication among all the nodes of hadoop cluster [Name node to data node one data to another data node] less bandwidth is reasonable.

1. Hadoop framework provides the alternatives to the nodes failures:

* In a hadoop cluster, if any data node is failure. Block corrects the hadoop framework automatically. Recover that fails data blocks as failure block from the existing data nodes.
* If name node is failing in hadoop cluster, Secondary name node will give the support to the hadoop cluster.

1. Commodity Hardware:

* Commonly available hardware (or) commonly used is known as commodity hardware.
* We can install hadoop cluster in commodity hardware (or) specialization hardware.

1. Daemons of hadoop:

* Background services technically known as Daemons within the hadoop cluster.
* We have 5 daemons, they are

1. Name Node
2. Secondary Name Node
3. Data Node
4. Job Tracker
5. Task Tracker

**HDFS**

**DFS:**

* DFS stands for Distributed File System.
* System that permanently store data is said to be distributed file system.
* DFS support concurrency, distribution, replication access to file and remote servers.
* The data is divided into logical units (files, shards, chunks, blocks).
* DFS’s are working network based approach because of DFS’s are more complex than regular disks file systems.
* Example: File system tolerates node failure without suffering data loss.

**HDFS:**

* Hadoop is a distributed file system and it uses to store large amount of data like terabytes, petabytes, zetabytes etc.
* HDFS supports high throughput mechanism for accessing this large amount of information.
* In HDFS files are stored in sequential redundant manner over the multiple machines and this guaranteed the following ones.

1. Durability to failure
2. High availability to very parallel applications

**Advantages of HDFS:**

* HDFS store large amount of information.
* HDFS is simple and robust coherency model.
* HDFS is reliable for long period of time.
* HDFS is scalable and fast access to this information and it also possible to serve large number of clients by simply adding more machines to the cluster.
* HDFS should integrate well with Hadoop MapReduce allowing data to be read and computed upon locally when possible.
* HDFS providing streaming read performance.
* Data will be written to the HDFS – once and then read several times.
* The overhead of cashing is helps the data should simply be re-read from HDFS source.
* Fault-tolerance by detecting faults and applying quick, automatic recovery.
* Processing logic close to the data, rather than data close to the processing logic.
* Portability across heterogeneous commodity hardware and operating systems.
* Economy by distributing data and processing across clusters of commodity personal computers.
* Efficiency by distributing data and logic to process it in parallel on nodes where data is located.
* Reliability by automatically maintain multiple copies of data and automatically redeploying processing logic in the event of failures.

**Disadvantages of HDFS:**

In distributed file system, it is limited in its power. The files in a NFS volume all reside on a single machine. This will create some problems.

* It does not give any reliability guaranties, if that machine goes down. Example: By replacing the files to other machine.
* All the clients must go to this machine to retrieve their data. This can overload the server if a large number of clients must be handled.
* Clients need to copy the data to their local machines before they can operate on it.

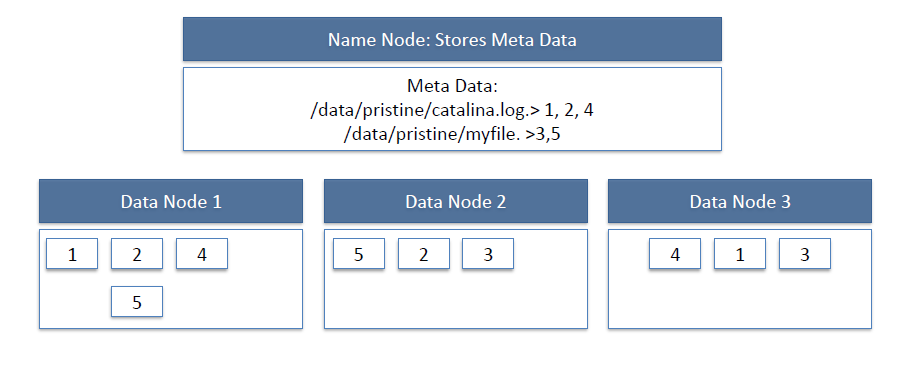
**Goals of HDFS:**

* Very large distributed file system: 10K nodes, 100 Million files. 10PB.
* Assumer commodity hardware: Files are replicated to handle hardware failure. Detect failures and recover from them.
* Optimized for batch processing: Data locations exposed so that computation can move to where data resides. It provides very high aggregate band width.

**HDFS is a block structured file system:**

Block:

* Block is the minimum unit of data that is said in HDFS, which is typically 64MB by default. However we can increase the block size in the multiples of 64MB.
* Each file is broken into blocks of a fixed size and these blocks are stored across a cluster of one or more machines with data storage capacity.
* Individual machines in the cluster are called the Data nodes.
* A file can be made of several blocks and not necessarily stored on the same machine.
* The target machine chose each block randomly on a block-by-block basis.
* So access permission to a file may need the cooperation of multiple machines and it supports file size for larger than a single machine DFS.
* Individual files sometimes need large space than a single hard drive could hold. If several machines must be involved in serving of a file, then a file could be rendered unavailable by the loss of any one of those machines. HDFS comebacks this problem by replacing each block across a number of machines (By default 3).



* In the above figure, the data nodes represent multiple files with replication factor of 2 and the same node maps the file names on to the block ids.
* In block structured file systems commonly use a block size on the order of 4 or 8KB.
* The default block size in HDFS is 64KB. This permits HDFS to decrease the amount of metadata storage required per file.
* In HDFS block structured file system, all the information’s are handled by the single machine called NameNode.
* The NameNode stores all the metadata for the file system. All the information’s like tracks file names, permissions and the locations of each block of each files etc can be stored in the main memory of the NameNodes machine and it permits fast access to the metadata.
* To open a file, the client first contacts NameNode and access a list of locations for the block that comprises the files and these locations identify the DataNodes which holds each block.
* Clients then read file data directly from the DataNode servers in parallel. So the NameNode is not directly involved in the bulk data transfer. Keeping its overhead to a minimum.
* NameNode information should be preserved even if the NameNode machine fails.
* NameNode failure is more severe for the cluster than DataNode failures.
* When individual DataNodes may crash and the entire cluster will continue to operate, the loss of the NameNode will render the cluster in accessible until it is manually restored.

**Features of HDFS:**

HDFS is file system designed for storing have some key characteristics. They are

1. Support for very large files.
2. Commodity Hardware.
3. Streaming data access.
4. High-latency data access.
5. Lots of small files.
6. Multiple writers, arbitrary file modifications.
7. Moving computation is then moving data.

**MapReduce**

**Introduction of MapReduce:**

* MapReduce published in 2004 by Google.
* Hadoop can run MapReduce programs written in various programming languages like java, ruby, python and c++.
* MapReduce is a parallel programming model for processing the huge amount of data.
* MapReduce making the structure data and out of some unstructured data….
* MapReduce provides automatic parallelization & distributions, fault-tolerance, I/O scheduling, monitoring, starters and updates.

**Motivation for MapReduce (why)?**

* Large scale data processing.
* MapReduce architecture provides automatic parallelism & distribution, fault-tolerance, I/O scheduling, monitoring & status updates.

**MapReduce History:**

* In 2004, MapReduce Paper released.
* In 2006, Lucen’s sub-project released.
* In 2008, Apache top-level project fastest sort of data.
* In 2012, MapReduce 2.0 (YARN) released. YARN stands for Yet Another Resource Negotiator.

**MapReduce:**

To take the advantage of parallel processing of Hadoop, the query must be in MapReduce form. The MapReduce is a paradigm which has two phases, the mapper phase and the reducer phase. In the Mapper the input is given in the form of key value pair. The output of the mapper is fed to the reducer as input. The reducer runs only after the mapper is over. The reducer too takes input in key value format and the output of reducer is final output.

## Steps in Map Reduce:

* Map takes a data in the form of pairs and returns a list of <key, value> pairs. The keys will not be unique in this case.
* Using the output of Map, sort and shuffle are applied by the Hadoop architecture. This sort and shuffle acts on these list of <key, value> pairs and sends out unique keys and a list of values associated with this unique key <key, list(values)>.
* Output of sort and shuffle will be sent to reducer phase. Reducer will perform a defined function on list of values for unique keys and Final output will<key, value> will be stored/displayed.

## How Many Maps:

The size of data to be processed decides the number of maps required. For example, we have 1000 MB data and block size is 64 MB then we need 16 mappers.

## Sort and Shuffle:

The sort and shuffle occur on the output of mapper and before the reducer. When the mapper task is complete, the results are sorted by key, partitioned if there are multiple reducers, and then written to disk. Using the input from each mapper <k2,v2> , we collect all the values for each unique key k2. This output from the shuffle phase in the form of <k2,list(v2)> is sent as input to reducer phase.

## ****How MapReduce works:****

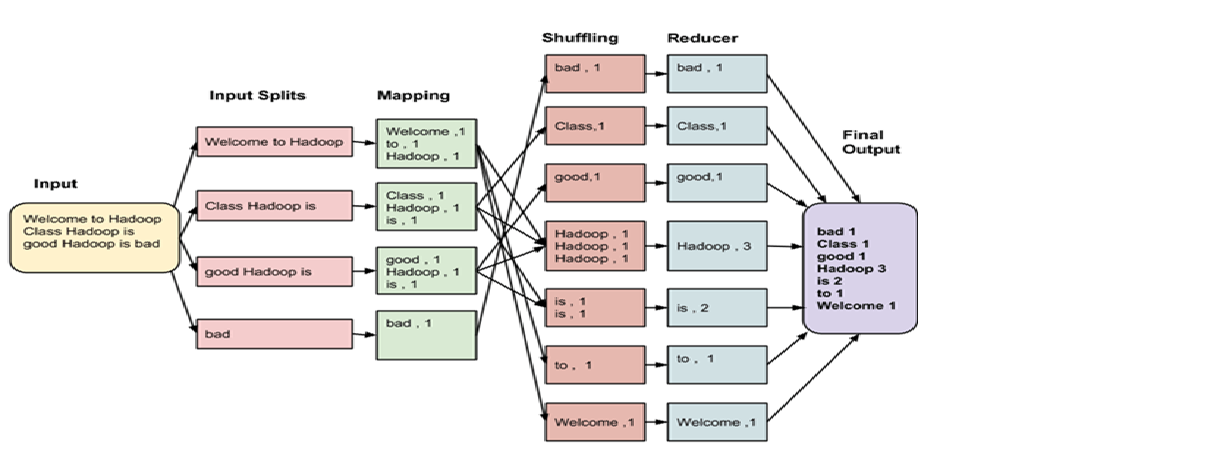
Let’s understand this with an example –

Consider you have following input data for your MapReduce Program

Welcome to Hadoop Class

Hadoop is good

Hadoop is bad



The data goes through following phases

**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 clubed 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.

# Counters & Joins:

# A counter in MapReduce is a mechanism used for collecting statistical information about the MapReduce job. This information could be useful for diagnosis of a problem in MapReduce job processing. Counters are similar to putting log message in the code for map or reduce.

# Typically, these counters are defined in a program (map or reduce) and are incremented during execution when a particular event or condition (specific to that counter) occurs. A very good application of counters is to track valid and invalid records from an input dataset.

## Two types of counters:

**1. Hadoop Built-In counters:**There are some built-in counters which exist per job. Below are built-in counter groups-

* **MapReduce Task Counters**- Collects task specific information (e.g., number of input records) during its execution time.
* **FileSystem Counters** - Collects information like number of bytes read or written by a task
* **FileInputFormat Counters** - Collects information of number of bytes read through FileInputFormat
* **FileOutputFormat Counters** - Collects information of number of bytes written through FileOutputFormat
* **Job Counters -** These counters are used by JobTracker. Statistics collected by them include e.g., number of task launched for a job.

**2. User Defined Counters:**

In addition to built-in counters, user can define his own counters using similar functionalities provided by programming languages. For example, in Java 'enum' are used to define user defined counters.

## ****MapReduce Join****

Joining two large dataset can be achieved using MapReduce Join. However, this process involves writing lots of code to perform actual join operation.

Joining of two datasets begin by comparing size of each dataset. If one dataset is smaller as compared to the other dataset then smaller dataset is distributed to every datanode in the cluster. Once it is distributed, either Mapper or Reducer uses smaller dataset to perform lookup for matching records from large dataset and then combine those records to form output records.

Depending upon the place where actual join is performed, this join is classified into-

**1. Map-side join -** When the join is performed by the mapper, it is called as map-side join. In this type, the join is performed before data is actually consumed by the map function. It is mandatory that the input to each map is in the form of a partition and is in sorted order. Also, there must be an equal number of partitions and it must be sorted by the join key.

**2. Reduce-side join -** When the join is performed by the reducer, it is called as reduce-side join. There is no necessity in this join to have dataset in a structured form (or partitioned).

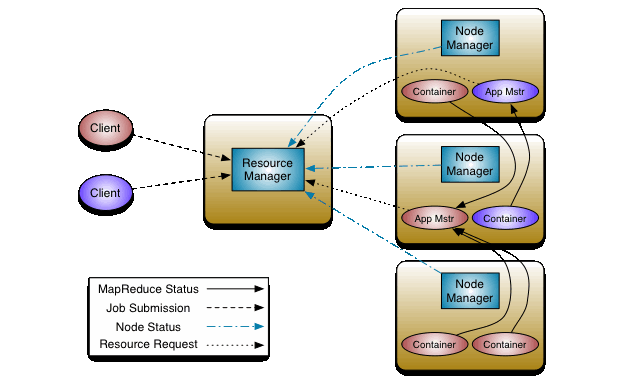
Here, map side processing emits join key and corresponding tuples of both the tables. As an effect of this processing, all the tuples with same join key fall into the same reducer which then joins the records with same join key.

**MapReduce 2.0 (YARN)**

The fundamental idea of YARN is to split up the functionalities of resource management and job scheduling/monitoring into separate daemons. The idea is to have a global ResourceManager (RM) and per-application ApplicationMaster (AM). An application is either a single job or a DAG of jobs.

The ResourceManager and the NodeManager form the data-computation framework. The ResourceManager is the ultimate authority that arbitrates resources among all the applications in the system. The NodeManager is the per-machine framework agent who is responsible for containers, monitoring their resource usage (CPU, memory, disk and network) and reporting the same to the ResourceManager/Scheduler.

The per-application ApplicationMaster is, in effect, a framework specific library and is tasked with negotiating resources from the ResourceManager and working with the NodeManager(s) to execute and monitor the tasks.



The ResourceManager has two main components: Scheduler and ApplicationsManager.

The Scheduler is responsible for allocating resources to the various running applications subject to familiar constraints of capacities, queues etc. The Scheduler is pure scheduler in the sense that it performs no monitoring or tracking of status for the application. Also, it offers no guarantees about restarting failed tasks either due to application failure or hardware failures. The Scheduler performs its scheduling function based on the resource requirements of the applications; it does so based on the abstract notion of a resource Container which incorporates elements such as memory, CPU, disk, network etc.

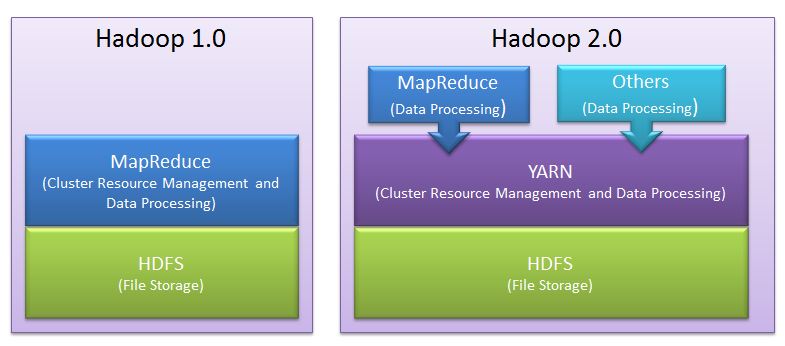
The Scheduler has a pluggable policy which is responsible for partitioning the cluster resources among the various queues, applications etc. The current schedulers such as the [CapacityScheduler](https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html) and the [FairScheduler](https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/FairScheduler.html) would be some examples of plug-ins.

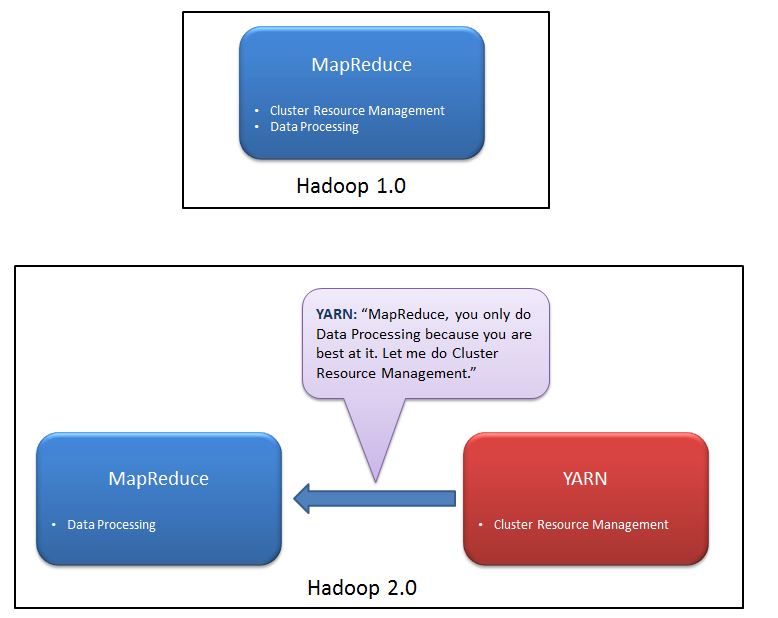
The ApplicationsManager is responsible for accepting job-submissions, negotiating the first container for executing the application specific ApplicationMaster and provides the service for restarting the ApplicationMaster container on failure. The per-application ApplicationMaster has the responsibility of negotiating appropriate resource containers from the Scheduler, tracking their status and monitoring for progress.

MapReduce in hadoop-2.x maintains API compatibility with previous stable release (hadoop-1.x). This means that all MapReduce jobs should still run unchanged on top of YARN with just a recompile.

YARN also supports the notion of resource reservation via the [ReservationSystem](https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/ReservationSystem.html), a component that allows users to specify a profile of resources over-time and temporal constraints (e.g., deadlines), and reserve resources to ensure the predictable execution of important jobs. The ReservationSystem tracks resources over-time, performs admission control for reservations, and dynamically instruct the underlying scheduler to ensure that the reservation is fullfilled.

**Introduction of new YARN layer in Hadoop 2.0:**

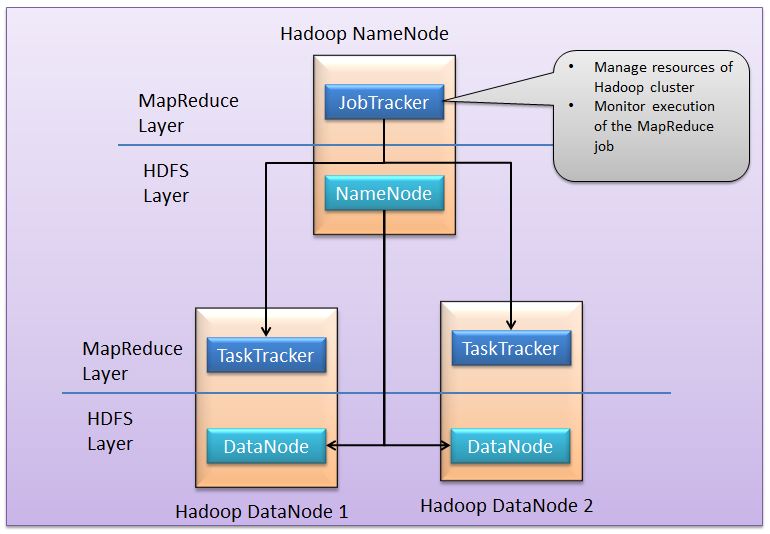
YARN (**Y**et **A**nother **R**esource **N**egotiator) is a new component added in Hadoop 2.0. Let’s have a look on how Hadoop architecture has changed from Hadoop 1.0 to Hadoop 2.0   
  
   
  
As shown, in Hadoop 2.0 a new layer has been introduced between HDFS and MapReduce.   
This is YARN framework which is responsible for doing Cluster Resource Management.   
  
**Cluster Resource Management:**

Cluster resource management means managing the resources of the Hadoop Clusters. And by resources we mean Memory, CPU etc.   
  
YARN took over this task of cluster management from MapReduce and MapReduce is streamlined to perform Data Processing only in which it is best.   


**Why YARN was needed?**

Before we understand the need of YARN, we should understand how cluster resource management was done in Hadoop 1.0 and what the problem in that approach was. 

Cluster Resource Management in Hadoop 1.0:

In Hadoop 1.0, there is tight coupling between Cluster Resource Management and MapReduce programming model.  Job Tracker, which does resource management, is part of, MapReduce Framework.   
  
   
In MapReduce framework, MapReduce job (MapReduce application) is divided between number of tasks called mappers and reducers. Each task runs on one of the machine (DataNode) of the cluster, and each machine has a limited number of predefined slots (map slot, reduce slot) for running tasks concurrently.   
  
Here, JobTracker is responsible for both managing the cluster's resources and driving the execution of the MapReduce job. It reserves and schedules slots for all tasks, configures, runs and monitors each task, and if a task fails, it allocates a new slot and reattempts the task. After a task finishes, the job tracker cleans up temporary resources and releases the task's slot to make it available for other jobs. 

Problems with this approach in Hadoop 1.0:

**It limits scalability:**JobTracker runs on single machine doing several task like

* Resource management
* Job and task scheduling and
* Monitoring

Although there are so many machines (DataNode) available; they are not getting used. This limits scalability.

**Availability Issue:**In Hadoop 1.0, JobTracker is single Point of availability. This means if JobTracker fails, all jobs must restart.

**Problem with Resource Utilization:**In Hadoop 1.0, there is concept of predefined number of map slots and reduce slots for each TaskTrackers. Resource Utilization issues occur because maps slots might be ‘full’ while reduce slots is empty (and vice-versa). Here the compute resources (DataNode) could sit idle which are reserved for Reduce slots even when there is immediate need for those resources to be used as Mapper slots.

**Limitation in running non-MapReduce Application:**In Hadoop 1.0, Job tracker was tightly integrated with MapReduce and only supporting application that obeys MapReduce program

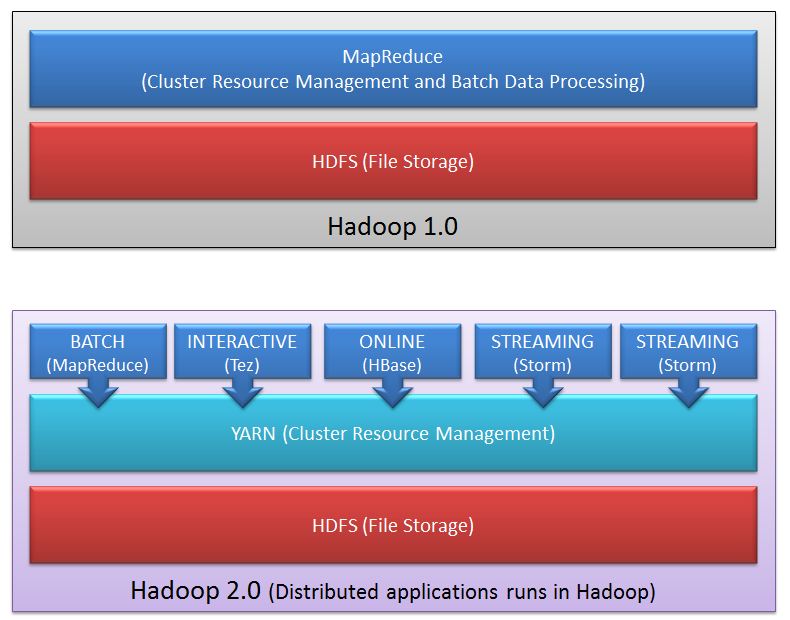
Framework can run on Hadoop.  Hadoop distributed file system (HDFS) makes it cheap to store large amounts of data, and its scalable MapReduce analysis engine makes it possible to extract insights from that data. MapReduce works on batch-driven data analysis, where the input data is partitioned into smaller batches that can be processed in parallel across many machines in the Hadoop cluster. But MapReduce, while powerful enough to express many data analysis algorithms, is not always the optimal choice of programming paradigm. It‘s often desirable to run other computation paradigms in the Hadoop cluster.

**Problem in performing real-time analysis:**MapReduce is batch driven. What if I want to do perform real time analysis instead of batch-processing (where results is available after several hours).   
  
There are many applications which need results in real time like fraud detection algorithm. There are real time engines like Apache Storm which can work better in this case. But in Hadoop 1.0, due to tight coupling these engines cannot run independently.

**Problem in running Message-Passing approach:**It is a stateful process that runs on each node of a distributed network. The processes communicate with each other by sending messages, and alter their state based on the messages they receive. This is not possible in MapReduce.

**Problem in running Ad-hoc query:**Many users like to query their big data using SQL. Apache Hive can execute a SQL query as a series of MapReduce jobs, but it has shortcomings in terms of performance.   
Recently, some new approaches such as Apache Tajo, Facebook's Presto and Cloudera's Impala drastically improve the performance, but they require to run services in other form than MapReduce form.   
It is not possible to run all such non Map Reduce jobs on Hadoop Cluster. Such jobs have to "disguise" themselves as mappers and reducers in order to be able to run on Hadoop 1.0.

Hadoop 2.0 solves all these problem with YARN:

   
  
YARN took over the task of cluster management from MapReduce and MapReduce is streamlined to perform Data Processing only in which it is best.   
  
YARN has central resource manager component which manages resources and allocates the resources to the application. Multiple applications can run on Hadoop via YARN and all application could share common resource management. 

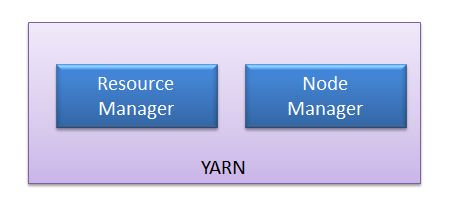
Advantage of YARN:

* 1. **Yarn does efficient utilization of the resource.**  
     There are no more fixed map-reduce slots. YARN provides central resource manager. With YARN, you can now run multiple applications in Hadoop, all sharing a common resource.
  2. **Yarn can even run application that do not follow MapReduce model.**  
     YARN decouples MapReduce's resource management and scheduling capabilities from the data processing component, enabling Hadoop to support more varied processing approaches and a broader array of applications. For example, Hadoop clusters can now run interactive querying and streaming data applications simultaneously with MapReduce batch jobs. This also streamlines MapReduce to do what is does best - process data.

**YARN is backward compatible.**  
This means that existing MapReduce job can run on Hadoop 2.0 without any change.

**No more JobTracker and TaskTracker needed in Hadoop 2.0**  
JobTracker and TaskTracker has totally disappeared. YARN splits the two major functionalities of the JobTracker i.e. resource management and job scheduling/monitoring into 2 separate daemons (components).

* Resource Manager
* Node Manager(node specific)

Central Resource Manager and node specific Node Manager together constitutes YARN.   
  


Difference between MR1 and MR2?

Earlier version of map- reduce framework in Hadoop 1.0 is called as **MR1**. The new version of MapReduce is known as **MR2**.   
  
No more JobTracker and TaskTracker needed in Hadoop 2. With the introduction of YARN in Hadoop2, the term JobTracker and TaskTracker disappeared. MapReduce is now streamlined to perform processing data.   
  
The new model is more isolated and scalable as compared to the earlier MR1 system. MR2 is one kind of distributed application that run MapReduce framework on top of YARN. MapReduce perform data processing via YARN. Other tools can also perform data processing via YARN. Hence Yarn execution model is more generic than earlier MapReduce model.   
MR1 was not able to do so. It would only run MapReduce applications.

**HIVE**

**What is 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.

Initially Hive was developed by Facebook, later the Apache Software Foundation took it up and developed it further as an open source under the name Apache Hive. It is used by different companies. For example, Amazon uses it in Amazon Elastic MapReduce.

### Hive is not

* A relational database
* A design for Online Transaction Processing (OLTP)
* A language for real-time queries and row-level updates

## Features of Hive

* It stores schema in a database and processed data into HDFS.
* It is designed for OLAP.
* It provides SQL type language for querying called HiveQL or HQL.
* It is familiar, fast, scalable, and extensible.

## Architecture of Hive

The following component diagram depicts the architecture of Hive:



This component diagram contains different units. The following table describes each unit:

|  |  |
| --- | --- |
| **Unit Name** | **Operation** |
| User Interface | Hive is a data warehouse infrastructure software that can create interaction between user and HDFS. The user interfaces that Hive supports are Hive Web UI, Hive command line, and Hive HD Insight (In Windows server). |
| Meta Store | Hive chooses respective database servers to store the schema or Metadata of tables, databases, columns in a table, their data types, and HDFS mapping. |
| HiveQL Process Engine | HiveQL is similar to SQL for querying on schema info on the Metastore. It is one of the replacements of traditional approach for MapReduce program. Instead of writing MapReduce program in Java, we can write a query for MapReduce job and process it. |
| Execution Engine | The conjunction part of HiveQL process Engine and MapReduce is Hive Execution Engine. Execution engine processes the query and generates results as same as MapReduce results. It uses the flavor of MapReduce. |
| HDFS or HBASE | Hadoop distributed file system or HBASE are the data storage techniques to store data into file system. |

## Working of Hive

The following diagram depicts the workflow between Hive and Hadoop.



The following table defines how Hive interacts with Hadoop framework:

|  |  |
| --- | --- |
| **Step No.** | **Operation** |
| 1 | **Execute Query**  The Hive interface such as Command Line or Web UI sends query to Driver (any database driver such as JDBC, ODBC, etc.) to execute. |
| 2 | **Get Plan**  The driver takes the help of query compiler that parses the query to check the syntax and query plan or the requirement of query. |
| 3 | **Get Metadata**  The compiler sends metadata request to Metastore (any database). |
| 4 | **Send Metadata**  Metastore sends metadata as a response to the compiler. |
| 5 | **Send Plan**  The compiler checks the requirement and resends the plan to the driver. Up to here, the parsing and compiling of a query is complete. |
| 6 | **Execute Plan**  The driver sends the execute plan to the execution engine. |
| 7 | **Execute Job**  Internally, the process of execution job is a MapReduce job. The execution engine sends the job to JobTracker, which is in Name node and it assigns this job to TaskTracker, which is in Data node. Here, the query executes MapReduce job. |
| 7.1 | **Metadata Ops**  Meanwhile in execution, the execution engine can execute metadata operations with Metastore. |
| 8 | **Fetch Result**  The execution engine receives the results from Data nodes. |
| 9 | **Send Results**  The execution engine sends those resultant values to the driver. |
| 10 | **Send Results**  The driver sends the results to Hive Interfaces. |

**APACHE PIG**

**APACHE PIG:**

Apache Pig is an abstraction over MapReduce. It is a tool/platform which is used to analyze larger sets of data representing them as data flows. Pig is generally used with Hadoop; we can perform all the data manipulation operations in Hadoop using Pig.

## What is Apache Pig?

Apache Pig is an abstraction over MapReduce. It is a tool/platform which is used to analyze larger sets of data representing them as data flows. Pig is generally used with Hadoop; we can perform all the data manipulation operations in Hadoop using Apache Pig.

To write data analysis programs, Pig provides a high-level language known as **Pig Latin**. This language provides various operators using which programmers can develop their own functions for reading, writing, and processing data.

To analyze data using **Apache Pig**, programmers need to write scripts using Pig Latin language. All these scripts are internally converted to Map and Reduce tasks. Apache Pig has a component known as **Pig Engine** that accepts the Pig Latin scripts as input and converts those scripts into MapReduce jobs.

## Why Do We Need Apache Pig?

Programmers who are not so good at Java normally used to struggle working with Hadoop, especially while performing any MapReduce tasks. Apache Pig is a boon for all such programmers.

* Using **Pig Latin**, programmers can perform MapReduce tasks easily without having to type complex codes in Java.
* Apache Pig uses **multi-query approach**, thereby reducing the length of codes. For example, an operation that would require you to type 200 lines of code (LoC) in Java can be easily done by typing as less as just 10 LoC in Apache Pig. Ultimately Apache Pig reduces the development time by almost 16 times.
* Pig Latin is **SQL-like language** and it is easy to learn Apache Pig when you are familiar with SQL.
* Apache Pig provides many built-in operators to support data operations like joins, filters, ordering, etc. In addition, it also provides nested data types like tuples, bags, and maps that are missing from MapReduce.

## Features of Pig:

Apache Pig comes with the following features −

* **Rich set of operators** − It provides many operators to perform operations like join, sort, filer, etc.
* **Ease of programming** − Pig Latin is similar to SQL and it is easy to write a Pig script if you are good at SQL.
* **Optimization opportunities** − The tasks in Apache Pig optimize their execution automatically, so the programmers need to focus only on semantics of the language.
* **Extensibility** − Using the existing operators, users can develop their own functions to read, process, and write data.
* **UDF’s** − Pig provides the facility to create **User-defined Functions** in other programming languages such as Java and invoke or embed them in Pig Scripts.
* **Handles all kinds of data** − Apache Pig analyzes all kinds of data, both structured as well as unstructured. It stores the results in HDFS.

## Apache Pig Vs MapReduce:

Listed below are the major differences between Apache Pig and MapReduce.

|  |  |
| --- | --- |
| **Apache Pig** | **MapReduce** |
| Apache Pig is a data flow language. | MapReduce is a data processing paradigm. |
| It is a high level language. | MapReduce is low level and rigid. |
| Performing a Join operation in Apache Pig is pretty simple. | It is quite difficult in MapReduce to perform a Join operation between datasets. |
| Any novice programmer with a basic knowledge of SQL can work conveniently with Apache Pig. | Exposure to Java is must to work with MapReduce. |
| Apache Pig uses multi-query approach, thereby reducing the length of the codes to a great extent. | MapReduce will require almost 20 times more the number of lines to perform the same task. |
| There is no need for compilation. On execution, every Apache Pig operator is converted internally into a MapReduce job. | MapReduce jobs have a long compilation process. |

## Apache Pig Vs SQL:

Listed below are the major differences between Apache Pig and SQL.

|  |  |
| --- | --- |
| **Pig** | **SQL** |
| Pig Latin is a **procedural** language. | SQL is a **declarative** language. |
| In Apache Pig, **schema** is optional. We can store data without designing a schema (values are stored as $01, $02 etc.) | Schema is mandatory in SQL. |
| The data model in Apache Pig is **nested relational**. | The data model used in SQL **is flat relational**. |
| Apache Pig provides limited opportunity for **Query optimization**. | There is more opportunity for query optimization in SQL. |

In addition to above differences, Apache Pig Latin −

* Allows splits in the pipeline.
* Allows developers to store data anywhere in the pipeline.
* Declares execution plans.
* Provides operators to perform ETL (Extract, Transform, and Load) functions.

## Apache Pig Vs Hive:

Both Apache Pig and Hive are used to create MapReduce jobs. And in some cases, Hive operates on HDFS in a similar way Apache Pig does. In the following table, we have listed a few significant points that set Apache Pig apart from Hive.

|  |  |
| --- | --- |
| **Apache Pig** | **Hive** |
| Apache Pig uses a language called **Pig Latin**. It was originally created at **Yahoo**. | Hive uses a language called **HiveQL**. It was originally created at **Facebook**. |
| Pig Latin is a data flow language. | HiveQL is a query processing language. |
| Pig Latin is a procedural language and it fits in pipeline paradigm. | HiveQL is a declarative language. |
| Apache Pig can handle structured, unstructured, and semi-structured data. | Hive is mostly for structured data. |

## Applications of Apache Pig:

Apache Pig is generally used by data scientists for performing tasks involving ad-hoc processing and quick prototyping. Apache Pig is used −

* To process huge data sources such as web logs.
* To perform data processing for search platforms.
* To process time sensitive data loads.

## Apache Pig – History:

In **2006**, Apache Pig was developed as a research project at Yahoo, especially to create and execute MapReduce jobs on every dataset. In **2007**, Apache Pig was open sourced via Apache incubator. In **2008**, the first release of Apache Pig came out. In **2010**, Apache Pig graduated as an Apache top-level project.

The language used to analyze data in Hadoop using Pig is known as **Pig Latin**. It is a highlevel data processing language which provides a rich set of data types and operators to perform various operations on the data.

To perform a particular task Programmers using Pig, programmers need to write a Pig script using the Pig Latin language, and execute them using any of the execution mechanisms (Grunt Shell, UDFs and Embedded). After execution, these scripts will go through a series of transformations applied by the Pig Framework, to produce the desired output.

Internally, Apache Pig converts these scripts into a series of MapReduce jobs, and thus, it makes the programmer’s job easy. The architecture of Apache Pig is shown below.



## Apache Pig Components:

As shown in the figure, there are various components in the Apache Pig framework. Let us take a look at the major components.

### Parser:

Initially the Pig Scripts are handled by the Parser. It checks the syntax of the script, does type checking, and other miscellaneous checks. The output of the parser will be a DAG (directed acyclic graph), which represents the Pig Latin statements and logical operators.

In the DAG, the logical operators of the script are represented as the nodes and the data flows are represented as edges.

### Optimizer:

The logical plan (DAG) is passed to the logical optimizer, which carries out the logical optimizations such as projection and pushdown.

### Compiler:

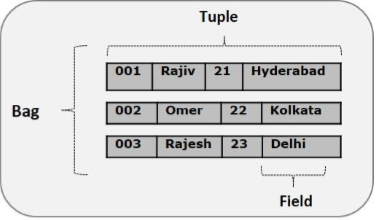
The compiler compiles the optimized logical plan into a series of MapReduce jobs.

### Execution engine:

Finally the MapReduce jobs are submitted to Hadoop in a sorted order. Finally, these MapReduce jobs are executed on Hadoop producing the desired results.

## Pig Latin Data Model:

The data model of Pig Latin is fully nested and it allows complex non-atomic datatypes such as **map** and **tuple**. Given below is the diagrammatical representation of Pig Latin’s data model.



### Atom:

Any single value in Pig Latin, irrespective of their data, type is known as an **Atom**. It is stored as string and can be used as string and number. int, long, float, double, chararray, and bytearray are the atomic values of Pig. A piece of data or a simple atomic value is known as a **field**.

Example − ‘raja’ or ‘30’

### Tuple:

A record that is formed by an ordered set of fields is known as a tuple, the fields can be of any type. A tuple is similar to a row in a table of RDBMS.

Example − (Raja, 30)

### Bag:

A bag is an unordered set of tuples. In other words, a collection of tuples (non-unique) is known as a bag. Each tuple can have any number of fields (flexible schema). A bag is represented by ‘{}’. It is similar to a table in RDBMS, but unlike a table in RDBMS, it is not necessary that every tuple contain the same number of fields or that the fields in the same position (column) have the same type.

Example − {(Raja, 30), (Mohammad, 45)}

A bag can be a field in a relation; in that context, it is known as **inner bag**.

Example − {Raja, 30, **{9848022338, raja@gmail.com,}**}

### Map:

A map (or data map) is a set of key-value pairs. The **key** needs to be of type chararray and should be unique. The **value** might be of any type. It is represented by ‘[]’

Example − [name#Raja, age#30]

### Relation:

A relation is a bag of tuples. The relations in Pig Latin are unordered (there is no guarantee that tuples are processed in any particular order).

**Sqoop**

**Sqoop Introduction:**

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

The traditional application management system, that is, the interaction of applications with relational database using RDBMS, is one of the sources that generate Big Data. Such Big Data, generated by RDBMS, is stored in **Relational Database Servers** in the relational database structure.

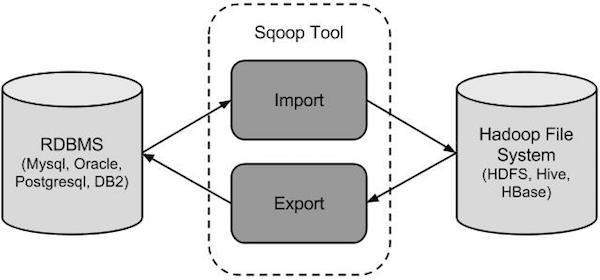
When Big Data storages and analyzers such as MapReduce, Hive, HBase, Cassandra, Pig, etc. of the Hadoop ecosystem came into picture, they required a tool to interact with the relational database servers for importing and exporting the Big Data residing in them. Here, Sqoop occupies a place in the Hadoop ecosystem to provide feasible interaction between relational database server and Hadoop’s HDFS.

**Sqoop:** “SQL to Hadoop and Hadoop to SQL”

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

## How Sqoop Works?

The following image describes the workflow of Sqoop.



## Sqoop Import:

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

## Sqoop Export:

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

As Sqoop is a sub-project of Hadoop, it can only work on Linux operating system.

**MySQL**

**MySQL Introduction:**

MySQL is the most popular Open Source Relational SQL Database Management System. MySQL is one of the best RDBMS being used for developing various web-based software applications. MySQL is developed, marketed and supported by MySQL AB, which is a Swedish company.

## What is a Database?

A database is a separate application that stores a collection of data. Each database has one or more distinct APIs for creating, accessing, managing, searching and replicating the data it holds.

Other kinds of data stores can also be used, such as files on the file system or large hash tables in memory but data fetching and writing would not be so fast and easy with those type of systems.

Nowadays, we use relational database management systems (RDBMS) to store and manage huge volume of data. This is called relational database because all the data is stored into different tables and relations are established using primary keys or other keys known as **Foreign Keys**.

A **Relational Database Management System (RDBMS)** is a software that −

* Enables you to implement a database with tables, columns and indexes.
* Guarantees the Referential Integrity between rows of various tables.
* Updates the indexes automatically.
* Interprets an SQL query and combines information from various tables.

## RDBMS Terminology:

Before we proceed to explain the MySQL database system, let us revise a few definitions related to the database.

* **Database** − A database is a collection of tables, with related data.
* **Table** − A table is a matrix with data. A table in a database looks like a simple spreadsheet.
* **Column** − One column (data element) contains data of one and the same kind, for example the column postcode.
* **Row** − A row (= tuple, entry or record) is a group of related data, for example the data of one subscription.
* **Redundancy** − Storing data twice, redundantly to make the system faster.
* **Primary Key** − A primary key is unique. A key value can not occur twice in one table. With a key, you can only find one row.
* **Foreign Key** − A foreign key is the linking pin between two tables.
* **Compound Key** − A compound key (composite key) is a key that consists of multiple columns, because one column is not sufficiently unique.
* **Index** − An index in a database resembles an index at the back of a book.
* **Referential Integrity** − Referential Integrity makes sure that a foreign key value always points to an existing row.

## MySQL Database:

MySQL is a fast, easy-to-use RDBMS being used for many small and big businesses. MySQL is developed, marketed and supported by MySQL AB, which is a Swedish company. MySQL is becoming so popular because of many good reasons −

* MySQL is released under an open-source license. So you have nothing to pay to use it.
* MySQL is a very powerful program in its own right. It handles a large subset of the functionality of the most expensive and powerful database packages.
* MySQL uses a standard form of the well-known SQL data language.
* MySQL works on many operating systems and with many languages including PHP, PERL, C, C++, JAVA, etc.
* MySQL works very quickly and works well even with large data sets.
* MySQL is very friendly to PHP, the most appreciated language for web development.
* MySQL supports large databases, up to 50 million rows or more in a table. The default file size limit for a table is 4GB, but you can increase this (if your operating system can handle it) to a theoretical limit of 8 million terabytes (TB).
* MySQL is customizable. The open-source GPL license allows programmers to modify the MySQL software to fit their own specific environments.

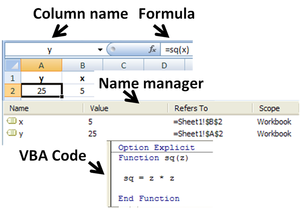
**EXCEL**

**Excel:**

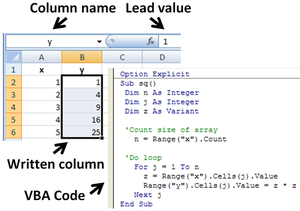
Microsoft Excel is a [spreadsheet](https://en.wikipedia.org/wiki/Spreadsheet) developed by [Microsoft](https://en.wikipedia.org/wiki/Microsoft) for [Windows](https://en.wikipedia.org/wiki/Microsoft_Windows), [macOS](https://en.wikipedia.org/wiki/MacOS), [Android](https://en.wikipedia.org/wiki/Android_(operating_system)) and [iOS](https://en.wikipedia.org/wiki/IOS). It features calculation, graphing tools, [pivot tables](https://en.wikipedia.org/wiki/Pivot_table), and a [macro](https://en.wikipedia.org/wiki/Macro_(computer_science)) programming language called [Visual Basic for Applications](https://en.wikipedia.org/wiki/Visual_Basic_for_Applications). It has been a very widely applied spreadsheet for these platforms, especially since version 5 in 1993, and it has replaced Lotus as the industry standard for spreadsheets. Excel forms part of [Microsoft Office](https://en.wikipedia.org/wiki/Microsoft_Office).

### Basic operation:

Microsoft Excel has the basic features of all spreadsheets, using a grid of cells arranged in numbered rows and letter-named columns to organize data manipulations like arithmetic operations. It has a battery of supplied functions to answer statistical, engineering and financial needs. In addition, it can display data as line graphs, histograms and charts, and with a very limited three-dimensional graphical display. It allows sectioning of data to view its dependencies on various factors for different perspectives (using [pivot tables](https://en.wikipedia.org/wiki/Pivot_table) and the scenario manager). It has a programming aspect, Visual Basic for Applications, allowing the user to employ a wide variety of numerical methods, for example, for solving differential equations of mathematical physics, and then reporting the results back to the spreadsheet. It also has a variety of interactive features allowing user interfaces that can completely hide the spreadsheet from the user, so the spreadsheet presents itself as a so-called application, or decision support system (DSS), via a custom-designed user interface, for example, a stock analyzer, or in general, as a design tool that asks the user questions and provides answers and reports. In a more elaborate realization, an Excel application can automatically poll external databases and measuring instruments using an update schedule, analyze the results, make a [Word](https://en.wikipedia.org/wiki/Microsoft_Word) report or [PowerPoint](https://en.wikipedia.org/wiki/Microsoft_PowerPoint) slide show, and e-mail these presentations on a regular basis to a list of participants. Excel was not designed to be used as a database.



Use of a user-defined function *sq(x)* in Microsoft Excel. The named variables *x* & *y* are identified in the *Name Manager*. The function *sq* is introduced using the *Visual Basic* editor supplied with Excel.



Subroutine in Excel calculates the square of named column variable *x* read from the spreadsheet, and writes it into the named column variable *y*.

#### History:

From its first version Excel supported end user programming of macros (automation of repetitive tasks) and user defined functions (extension of Excel's built-in function library). In early versions of Excel these programs were written in a macro language whose statements had formula syntax and resided in the cells of special purpose macro sheets (stored with file extension .XLM in Windows.) XLM was the default macro language for Excel through Excel 4.0. Beginning with version 5.0 Excel recorded macros in VBA by default but with version 5.0 XLM recording was still allowed as an option. After version 5.0 that option was discontinued. All versions of Excel, including Excel 2010 are capable of running an XLM macro, though Microsoft discourages their use.

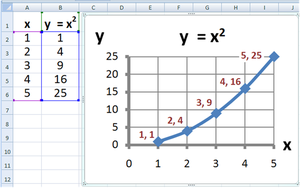
### Charts:

Excel supports [charts](https://en.wikipedia.org/wiki/Chart), [graphs](https://en.wikipedia.org/wiki/Graph_of_a_function), or [histograms](https://en.wikipedia.org/wiki/Histogram) generated from specified groups of cells. The generated graphic component can either be embedded within the current sheet, or added as a separate object.

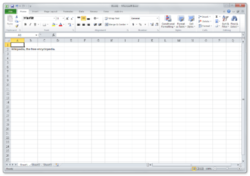
These displays are dynamically updated if the content of cells changes. For example, suppose that the important design requirements are displayed visually; then, in response to a user's change in trial values for parameters, the curves describing the design change shape and their points of intersection shift, assisting the selection of the best design.

### Number of rows and columns:

Versions of Excel up to 7.0 had a limitation in the size of their data sets of 16K (214 = 16384) rows. Versions 8.0 through 11.0 could handle 64K (216 = 65536) rows and 256 columns (28 as label 'IV'). Version 12.0 can handle 1M (220 = 1048576) rows, and 16384 (214 as label 'XFD') columns.



Created a graph using excel



Excel sheet

**ANALYSING OF H1B VISA APPLICANTS**

**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.

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 growth in applications.

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.

3) Which industry has the most number of Data Scientist positions?

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

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

6) 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.

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

8) Find the average Prevailing Wage for each Job for each Year (take part time and full time

Separate)

9) Which are top ten employers who have the highest success rate in petitions?

10) Which are the top 10 job positions which have the highest success rate in petitions?

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

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

Bottom of Form

The dataset has nearly 3 million records.

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

In the data, few columns are enclosed by double quotes and also we have comma’s in a single column and the column is enclosed by double quotes. So we have used hive csv serve to load the data. In the quoteChar, we have given **“(**double quote**).**So this will take the column value in between the double quotes.

Let’s create a table to load the h1b applicant’s data as shown below.

|  |  |
| --- | --- |
|  | CREATE TABLE h1b\_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 SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'    WITH SERDEPROPERTIES (    "separatorChar" = ",",    "quoteChar" = "\""    ) STORED AS TEXTFILE; |

**USE CASES IN MapReduce:**

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

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

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

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

public class Task1a {

public static class map1a extends Mapper<LongWritable, Text, Text, Text> {

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

throws IOException, InterruptedException {

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

String year = str[7];

String job\_title = str[4];

String petition =str[2];

if (str[4].equals("DATA ENGINEER")) {

con.write(new Text(year), new Text(petition));

}

}

}

public static class reducer1a extends Reducer<Text,Text,Text,IntWritable>

{

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

{

int count=0;

for(Text val:value)

{

count++;

}

con.write(key,new IntWritable(count));

}

}

public static void main(String[] args) throws IllegalArgumentException,

IOException, ClassNotFoundException, InterruptedException {

Configuration con = new Configuration();

Job job = Job.getInstance(con, "");

job.setJarByClass(Task1a.class);

job.setMapperClass(map1a.class);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(Text.class);

job.setReducerClass(reducer1a.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

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

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

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

}

}

Execution:

hadoop jar Task1a.jar Task1a /user/hive/warehouse/h1b.db/h1b /user/hduser/hadoop/Task1aoutput1;

hadoop fs -cat /user/hduser/hadoop/Task1aoutput1/p\*;

Output:

2011 18

2012 32

2013 41

2014 89

2015 160

2016 251

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

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.IntWritable;

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 Task1b {

public static class map1b extends Mapper<LongWritable, Text, Text, IntWritable> {

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

throws IOException, InterruptedException {

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

String job\_title = str[4];

String application = (str[6]);

con.write(new Text(job\_title), new IntWritable(1));

}

}

public static class reducer1b extends

Reducer<Text, IntWritable, NullWritable, Text> {

private TreeMap<Long, Text> repToRecordMap = new TreeMap<Long, Text>();

public void reduce(Text key, Iterable<IntWritable> values, Context context)

throws IOException, InterruptedException {

int sum = 0;

for (IntWritable val : values) {

sum=sum+val.get();

}

String mytotal = String.format("%d", sum);

String myValue = key.toString();

myValue = myValue + ',' + mytotal;

repToRecordMap.put(new Long(sum), new Text(myValue));

if (repToRecordMap.size() > 5) {

repToRecordMap.remove(repToRecordMap.firstKey());

}

}

protected void cleanup(Context context) throws IOException,

InterruptedException {

for (Text t : repToRecordMap.values()) {

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

}

}

}

public static void main(String[] args) throws IOException,

ClassNotFoundException, InterruptedException {

Configuration conf = new Configuration();

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

job.setJarByClass(Task1b.class);

job.setMapperClass(map1b.class);

job.setReducerClass(reducer1b.class);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(IntWritable.class);

job.setOutputKeyClass(NullWritable.class);

job.setOutputValueClass(IntWritable.class);

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

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

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

}

}

Execution:

hadoop jar Task1b.jar Task1b /user/hive/warehouse/h1b.db/h1b /user/hduser/hadoop/Task1boutput;

hadoop fs -cat /user/hduser/hadoop/Task1boutput/p\*;

Output:

SOFTWARE DEVELOPER,42907

SYSTEMS ANALYST,61965

COMPUTER PROGRAMMER,70570

SOFTWARE ENGINEER,121307

PROGRAMMER ANALYST,249038

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

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 Task2a {

public static class map2a extends

Mapper<LongWritable, Text, Text, Text> {

public void map(LongWritable key, Text value, Context context)

throws IOException, InterruptedException {

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

String year = (str[7]);

String job\_title = str[4];

String worksite = str[8];

if (job\_title.equals("DATA ENGINEER")) {

context.write(new Text(year + '\t' + worksite), new Text(

worksite));

}

}

}

public static class reducer2a extends Partitioner<Text, Text> {

@Override

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

String[] rec = key.toString().split("\t");

String year = rec[0];

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

return 0;

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

return 1;

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

return 2;

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

return 3;

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

return 4;

} else {

return 5;

}

}

}

public static class pet\_Reducer extends

Reducer<Text, Text, NullWritable, Text> {

private TreeMap<Long, Text> repToRecordMap = new TreeMap<Long, Text>();

public void reduce(Text key, Iterable<Text> values, Context context)

throws IOException, InterruptedException {

int count = 0;

for (Text val : values) {

count++;

}

String mytotal = String.format("%d", count);

String myValue = key.toString();

myValue = myValue + ',' + mytotal;

repToRecordMap.put(new Long(count), new Text(myValue));

if (repToRecordMap.size() > 5) {

repToRecordMap.remove(repToRecordMap.firstKey());

}

}

protected void cleanup(Context context) throws IOException,

InterruptedException {

for (Text t : repToRecordMap.values()) {

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

}

}

}

public static void main(String[] args) throws IOException,

ClassNotFoundException, InterruptedException {

Configuration conf = new Configuration();

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

job.setJarByClass(Task2a.class);

job.setMapperClass(map2a.class);

job.setPartitionerClass(reducer2a.class);

job.setReducerClass(pet\_Reducer.class);

job.setNumReduceTasks(6);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(Text.class);

job.setOutputKeyClass(NullWritable.class);

job.setOutputValueClass(Text.class);

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

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

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

}

}

Execution:

hadoop jar Task2a.jar Task2a /user/hive/warehouse/h1b.db/h1b /user/hduser/hadoop/Task2aoutput;

hadoop fs -cat /user/hduser/hadoop/Task2aoutput/p\*;

Output:

2011 SUPERIOR, COLORADO,1

2011 PLANO, TEXAS,2

2011 SAN FRANCISCO, CALIFORNIA,3

2012 WOODLAND HILLS, CALIFORNIA,1

2012 SEATTLE, WASHINGTON,2

2012 PONTIAC, MICHIGAN,3

2012 SAN FRANCISCO, CALIFORNIA,7

2013 WAYNE, PENNSYLVANIA,1

2013 NEWARK, NEW JERSEY,2

2013 NEW YORK, NEW YORK,4

2013 SAN FRANCISCO, CALIFORNIA,5

2013 MENLO PARK, CALIFORNIA,10

2014 OMAHA, NEBRASKA,4

2014 MOUNTAIN VIEW, CALIFORNIA,5

2014 NEW YORK, NEW YORK,9

2014 SAN FRANCISCO, CALIFORNIA,12

2014 MENLO PARK, CALIFORNIA,13

2015 SEATTLE, WASHINGTON,6

2015 SAN MATEO, CALIFORNIA,12

2015 MENLO PARK, CALIFORNIA,19

2015 NEW YORK, NEW YORK,25

2015 SAN FRANCISCO, CALIFORNIA,33

2016 REDWOOD CITY, CALIFORNIA,6

2016 SAN MATEO, CALIFORNIA,9

2016 SAN FRANCISCO, CALIFORNIA,33

2016 NEW YORK, NEW YORK,34

2016 MENLO PARK, CALIFORNIA,35

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

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 Task2b {

public static class map2b extends Mapper<LongWritable, Text, Text, Text> {

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

throws IOException, InterruptedException {

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

String case\_status = rec[1];

String worksite = rec[8];

String year = rec[7];

if (case\_status.equals("CERTIFIED")

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

con.write(new Text(year + '\t' + worksite), new Text(worksite));

}

}

}

public static class reducer2b extends Partitioner<Text, Text> {

@Override

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

String[] rec = key.toString().split("\t");

String year = rec[0];

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

return 0;

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

return 1;

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

return 2;

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

return 3;

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

return 4;

} else {

return 5;

}

}

}

public static class red\_loc extends Reducer<Text, Text, NullWritable, Text> {

private TreeMap<Long, Text> repToRecordMap = new TreeMap<Long, Text>();

public void reduce(Text key, Iterable<Text> values, Context context)

throws IOException, InterruptedException {

int count = 0;

for (Text val : values) {

count++;

}

String mytotal = String.format("%d", count);

String myValue = key.toString();

myValue = myValue + ',' + mytotal;

repToRecordMap.put(new Long(count), new Text(myValue));

if (repToRecordMap.size() > 5) {

repToRecordMap.remove(repToRecordMap.firstKey());

}

}

protected void cleanup(Context context) throws IOException,

InterruptedException {

for (Text t : repToRecordMap.values()) {

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

}

}

}

public static void main(String[] args) throws IOException,

ClassNotFoundException, InterruptedException {

Configuration conf = new Configuration();

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

job.setJarByClass(Task2b.class);

job.setMapperClass(map2b.class);

job.setPartitionerClass(reducer2b.class);

job.setReducerClass(red\_loc.class);

job.setNumReduceTasks(6);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(Text.class);

job.setOutputKeyClass(NullWritable.class);

job.setOutputValueClass(Text.class);

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

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

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

}

}

Execution:

hadoop jar Task2b.jar Task2b /user/hive/warehouse/h1b.db/h1b /user/hduser/hadoop/Task2boutput;

hadoop fs -cat /user/hduser/hadoop/Task2boutput/p\*;

Output:

2011 SAN JOSE, CALIFORNIA,4883

2011 SAN FRANCISCO, CALIFORNIA,4893

2011 CHICAGO, ILLINOIS,5410

2011 HOUSTON, TEXAS,8619

2011 NEW YORK, NEW YORK,24024

2012 ATLANTA, GEORGIA,5958

2012 CHICAGO, ILLINOIS,6131

2012 SAN FRANCISCO, CALIFORNIA,6763

2012 HOUSTON, TEXAS,11145

2012 NEW YORK, NEW YORK,26161

2013 ATLANTA, GEORGIA,6853

2013 SAN JOSE, CALIFORNIA,7243

2013 SAN FRANCISCO, CALIFORNIA,8003

2013 HOUSTON, TEXAS,12249

2013 NEW YORK, NEW YORK,25888

2014 ATLANTA, GEORGIA,8608

2014 SAN JOSE, CALIFORNIA,8914

2014 SAN FRANCISCO, CALIFORNIA,10779

2014 HOUSTON, TEXAS,14476

2014 NEW YORK, NEW YORK,30132

2015 SAN JOSE, CALIFORNIA,10471

2015 ATLANTA, GEORGIA,11137

2015 SAN FRANCISCO, CALIFORNIA,13815

2015 HOUSTON, TEXAS,16606

2015 NEW YORK, NEW YORK,34216

2016 CHICAGO, ILLINOIS,11998

2016 ATLANTA, GEORGIA,12382

2016 SAN FRANCISCO, CALIFORNIA,15029

2016 HOUSTON, TEXAS,15514

2016 NEW YORK, NEW YORK,37746

3)Which industry has the most number of Data Scientist positions?

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

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

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

public class Task3{

public static class map3 extends Mapper<LongWritable, Text, Text, Text> {

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

throws IOException, InterruptedException {

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

String job\_title = str[4];

String petition =str[3];

con.write(new Text(petition), new Text(job\_title));

}

}

public static class reducer3 extends Reducer<Text,Text,Text,IntWritable>

{

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

{

int count=0;

for(Text val:value)

{

count++;

}

con.write(key,new IntWritable(count));

}

}

public static void main(String[] args) throws IllegalArgumentException,

IOException, ClassNotFoundException, InterruptedException {

Configuration con = new Configuration();

Job job = Job.getInstance(con, "");

job.setJarByClass(Task3.class);

job.setMapperClass(map3.class);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(Text.class);

job.setReducerClass(reducer3.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

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

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

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

}

}

Execution:

hadoop jar Task3.jar Task3 /user/hive/warehouse/h1b.db/h1b /user/hduser/hadoop/Task3output;

hadoop fs -cat /user/hduser/hadoop/Task3output/p\*;

Output:

ACCOUNTANT 2

ACCOUNTANTS 132

ACCOUNTANTS AND AUDITORS 22114

ACTORS 3

ACTUARIES 1389

ACUPUNCTURISTS 1

ADMINISTRATIVE SERVICES MANAGERS 315

ADULT BASIC AND SECONDARY EDUCATION AND LITERACY 116

ADULT BASIC AND SECONDARY EDUCATION AND LITERACY T 6

ADULT BASIC SECOND EDUC AND LIT TEACH AND INSTRUC 2

ADVERTISING AND PROMORTIONS MANAGERS 1

ADVERTISING AND PROMOTIONS MANAGER 2

ADVERTISING AND PROMOTIONS MANAGERS 851

ADVERTISING SALES AGENTS 69

AEROSPACE ENGINEERING AND OPERATIONS TECHNICIANS 50

AEROSPACE ENGINEERS 627

AGENTS AND BUSINESS MANAGERS OF ARTISTS 1

AGENTS AND BUSINESS MANAGERS OF ARTISTS, 96

AGENTS AND BUSINESS MANAGERS OF ARTISTS, PERFORMER 2

AGRICULTURAL AND FOOD SCIENCE TECHNICIANS 131

AGRICULTURAL ENGINEERS 208

AGRICULTURAL INSPECTORS 88

AGRICULTURAL SCIENCES TEACHERS, POSTSECONDARY 203

AGRICULTURAL WORKERS, ALL OTHER 6

AIRCRAFT CARGO HANDLING SUPERVISSORS 1

AIRCRAFT MECHANICS AND SERVICE TECHNICIANS 16

AIRFIELD OPERATIONS SPECIALISTS 3

AIRLINE PILOTS, COPILOTS, AND FLIGHT ENGINEERS 25

AMUSEMENT AND RECREATION ATTENDANTS 3

ANESTHESIOLOGISTS 193

ANIMAL BREEDERS 40

ANIMAL SCIENTISTS 263

ANIMAL TRAINERS 15

ANTHROPOLOGIST AND ARCHEOLOGIST 1

ANTHROPOLOGISTS AND ARCHEOLOGISTS 26

ANTHROPOLOGY AND ARCHEOLOGY TEACHERS, 93

ANTHROPOLOGY AND ARCHEOLOGY TEACHERS, POSTSECONDAR 4

APPLICATION DEVELOPER 1

APPLICATION PROGRAMMER 81

APPRAISERS AND ASSESSORS OF REAL ESTATE 32

ARBITRATORS, MEDIATORS, AND CONCILIATORS 9

ARCHITECT, EXCEPT LANDSCAPE AND NAVAL 2

ARCHITECTS, EXCEPT LANDSCAPE AND NAVAL 2350

ARCHITECTURAL AND CIVIL DRAFTERS 1720

ARCHITECTURAL AND ENGINEERING MANAGERS 2687

ARCHITECTURAL AND ENGINEERS MANAGERS 1

ARCHITECTURAL DRAFTERS 18

ARCHITECTURE TEACHERS, POSTSECONDARY 223

ARCHIVISTS 54

AREA, ETHNIC, & CULTURAL TEACHERS, POSTSECONDARY 1

AREA, ETHNIC, AND CULTURAL STUDIES TEACHERS, 160

AREA, ETHNIC, AND CULTURAL STUDIES TEACHERS, POSTS 4

ART DIRECTORS 894

ART, DRAMA AND MUSIC TEACHERS, POSTSECONDARY 7

ART, DRAMA, AND MUSIC TEACHER, POSTSECONDARY 1

ART, DRAMA, AND MUSIC TEACHERS, POSTSECONDARY 635

ARTISTS AND RELATED WORKERS, ALL OTHER 47

ASSOCIATE DOCTOR OF CHIROPRACTIC 1

ASTRONOMERS 208

ATHLETES AND SPORTS COMPETITORS 3

ATHLETIC TRAINERS 146

ATMOSPHERIC AND SPACE SCIENTISTS 338

ATMOSPHERIC, EARTH, MARINE, & SPACE SCIENCES TEACH 1

ATMOSPHERIC, EARTH, MARINE, AND SPACE SCIENCES 174

ATMOSPHERIC, EARTH, MARINE, AND SPACE SCIENCES TEA 2

ATMOSPHERIC, EARTH, MARINE, AND SPACE SCIENCES TEACHERS, POSTSECONDARY 1

ATMOSPHERIC, EARTH, MARINE, SPACE SCIENCES TEACHER 1

ATTORNEY 3

AUDIO AND VIDEO EQUIPMENT TECHNICIANS 22

AUDIO-VISUAL AND MULTIMEDIA COLLECTIONS 27

AUDIOLOGISTS 51

AUDITORS 16

AUTOMOTIVE BODY AND RELATED REPAIRERS 3

AUTOMOTIVE ENGINEER 3

AUTOMOTIVE ENGINEERS 41

AUTOMOTIVE SERVICE TECHNICIANS AND MECHANICS 19

Able Seamen 2

Accountants 3826

Accountants and Auditors 30708

Actors 23

Actuaries 2097

Adhesive Bonding Machine Operators and Tenders 4

Administrative Law Judges, Adjudicators, and Heari 6

Administrative Services Managers 1046

Adult Basic and Secondary Education and Literacy T 224

Adult Literacy, Remedial Education, and GED Teache 50

Advertising Sales Agents 126

Advertising and Promotions Managers 2074

Aerospace Engineering and Operations Technicians 129

Aerospace Engineers 1331

Agents and Business Managers of Artists, Performer 244

Agricultural Crop Farm Managers 48

Agricultural Engineers 391

Agricultural Equipment Operators 4

Agricultural Inspectors 240

Agricultural Sciences Teachers, Postsecondary 240

Agricultural Technicians 25

Agricultural Workers, All Other 28

Agricultural and Food Science Technicians 255

Agricultural and Food Scientists 10

Aircraft Engine Specialists 3

Aircraft Mechanics and Service Technicians 59

Aircraft Structure, Surfaces, Rigging, and Systems 1

Airfield Operations Specialists 5

Airframe-and-Power-Plant Mechanics 2

Airline Pilots, Copilots, and Flight Engineers 31

All other air transportation workers 1

All other architects, surveyors,and cartographers 8

All other food processing workers 2

Amusement and Recreation Attendants 3

Anesthesiologists 373

Animal Breeders 283

Animal Scientists 475

Animal Trainers 28

Anthropologists 4

Anthropologists and Archeologists 32

Anthropology and Archeology Teachers, Postsecondar 194

Appraisers and Assessors of Real Estate 55

Appraisers, Real Estate 10

Arbitrators, Mediators, and Conciliators 11

Archeologists 5

Architects, Except Landscape and Naval 3756

Architectural Drafters 266

Architectural and Civil Drafters 1981

Architectural and Engineering Managers 4413

Architecture Teachers, Postsecondary 426

Archivists 122

Area, Ethnic, and Cultural Studies Teachers, Posts 320

Art Directors 1473

Art, Drama, and Music Teachers, Postsecondary 1185

Artists and Related Workers, All Other 135

Assemblers and Fabricators, All Other 2

Astronomers 440

Athletes and Sports Competitors 5

Athletic Trainers 422

Atmospheric and Space Scientists 619

Atmospheric, Earth, Marine, and Space Sciences Tea 287

Audio and Video Equipment Technicians 37

Audio-Visual Collections Specialists 10

Audio-Visual and Multimedia Collections Specialist 66

Audiologists 84

Auditors 1075

Automotive Body and Related Repairers 1

Automotive Master Mechanics 10

Automotive Service Technicians and Mechanics 10

Automotive Specialty Technicians 3

Aviation Inspectors 2

Avionics Technicians 12

BAKERS 10

BARBERS 1

BARTENDERS 1

BILL AND ACCOUNT COLLECTORS 3

BILLING AND POSTING CLERKS 14

BIOCHEMICAL ENGINEERS 3

……………………………..Output is too long…………………………………………

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

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 Task4 {

public static class map4 extends

Mapper<LongWritable, Text, Text, Text> {

public void map(LongWritable key, Text value, Context context)

throws IOException, InterruptedException {

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

String year = (str[7]);

String petition\_employer = str[3];

context.write(new Text(year + "\t" + str[2]), new Text(

petition\_employer));

}

}

public static class year\_partitionerclass extends Partitioner<Text, Text> {

@Override

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

String[] rec = key.toString().split("\t");

String year = rec[0];

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

return 0;

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

return 1;

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

return 2;

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

return 3;

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

return 4;

} else {

return 5;

}

}

}

public static class reducer4 extends

Reducer<Text, Text, NullWritable, Text> {

private TreeMap<Long, Text> repToRecordMap = new TreeMap<Long, Text>();

public void reduce(Text key, Iterable<Text> values, Context context)

throws IOException, InterruptedException {

int count = 0;

for (Text val : values) {

count++;

}

String mytotal = String.format("%d", count);

String myValue = key.toString();

myValue = myValue + ',' + mytotal;

repToRecordMap.put(new Long(count), new Text(myValue));

if (repToRecordMap.size() > 5) {

repToRecordMap.remove(repToRecordMap.firstKey());

}

}

protected void cleanup(Context context) throws IOException,

InterruptedException {

for (Text t : repToRecordMap.values()) {

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

}

}

}

public static void main(String[] args) throws IOException,

ClassNotFoundException, InterruptedException {

Configuration conf = new Configuration();

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

job.setJarByClass(Task4.class);

job.setMapperClass(map4.class);

job.setPartitionerClass(year\_partitionerclass.class);

job.setReducerClass(reducer4.class);

job.setNumReduceTasks(6);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(Text.class);

job.setOutputKeyClass(NullWritable.class);

job.setOutputValueClass(Text.class);

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

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

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

}

}

Execution:

hadoop jar Task4.jar Task4 /user/hive/warehouse/h1b.db/h1b /user/hduser/hadoop/Task4output;

hadoop fs -cat /user/hduser/hadoop/Task4output/p\*;

Output:

2011 COGNIZANT TECHNOLOGY SOLUTIONS U.S. CORPORATION,2721

2011 WIPRO LIMITED,3028

2011 DELOITTE CONSULTING LLP,3621

2011 MICROSOFT CORPORATION,4253

2011 TATA CONSULTANCY SERVICES LIMITED,5416

2012 IBM INDIA PRIVATE LIMITED,4074

2012 DELOITTE CONSULTING LLP,4727

2012 TATA CONSULTANCY SERVICES LIMITED,6735

2012 WIPRO LIMITED,7182

2012 INFOSYS LIMITED,15818

2013 ACCENTURE LLP,4994

2013 DELOITTE CONSULTING LLP,6124

2013 WIPRO LIMITED,6734

2013 TATA CONSULTANCY SERVICES LIMITED,8790

2013 INFOSYS LIMITED,32223

2014 ACCENTURE LLP,5498

2014 DELOITTE CONSULTING LLP,7017

2014 WIPRO LIMITED,8365

2014 TATA CONSULTANCY SERVICES LIMITED,14098

2014 INFOSYS LIMITED,23759

2015 ACCENTURE LLP,9605

2015 IBM INDIA PRIVATE LIMITED,10693

2015 WIPRO LIMITED,12201

2015 TATA CONSULTANCY SERVICES LIMITED,16553

2015 INFOSYS LIMITED,33245

2016 IBM INDIA PRIVATE LIMITED,9787

2016 WIPRO LIMITED,10607

2016 TATA CONSULTANCY SERVICES LIMITED,13134

2016 CAPGEMINI AMERICA INC,16725

2016 INFOSYS LIMITED,25352

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

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.\*;

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.Reducer.Context;

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

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

public class Task5 {

public static class map5 extends Mapper<LongWritable, Text, Text, Text> {

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

throws IOException, InterruptedException {

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

String year = rec[7];

String job\_title = rec[4];

String petitions = rec[3];

con.write(new Text(year + '\t' + job\_title), new Text(petitions));

}

}

public static class year\_partitionerclass extends Partitioner<Text, Text> {

@Override

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

String[] rec = key.toString().split("\t");

String year = rec[0];

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

return 0;

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

return 1;

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

return 2;

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

return 3;

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

return 4;

} else {

return 5;

}

}

}

public static class reducer5 extends Reducer<Text, Text, NullWritable, Text> {

private TreeMap<Long, Text> repToRecordMap = new TreeMap<Long, Text>();

public void reduce(Text key, Iterable<Text> values, Context context)

throws IOException, InterruptedException {

int count = 0;

for (Text val : values) {

count++;

}

String mytotal = String.format("%d", count);

String myValue = key.toString();

myValue = myValue + ',' + mytotal;

repToRecordMap.put(new Long(count), new Text(myValue));

if (repToRecordMap.size() > 10) {

repToRecordMap.remove(repToRecordMap.firstKey());

}

}

protected void cleanup(Context context) throws IOException,

InterruptedException {

for (Text t : repToRecordMap.values()) {

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

}

}

}

public static void main(String[] args) throws IOException,

ClassNotFoundException, InterruptedException {

Configuration conf = new Configuration();

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

job.setJarByClass(Task5.class);

job.setMapperClass(map5.class);

job.setPartitionerClass(year\_partitionerclass.class);

job.setReducerClass(reducer5.class);

job.setNumReduceTasks(6);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(Text.class);

job.setOutputKeyClass(NullWritable.class);

job.setOutputValueClass(Text.class);

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

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

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

}

}

Execution:

hadoop jar Task5.jar Task5 /user/hive/warehouse/h1b.db/h1b /user/hduser/hadoop/Task5output;

hadoop fs -cat /user/hduser/hadoop/Task5output/p\*;

Output:

2011 SENIOR CONSULTANT,2798

2011 SENIOR SOFTWARE ENGINEER,2935

2011 PHYSICAL THERAPIST,3377

2011 ASSISTANT PROFESSOR,3467

2011 COMPUTER SYSTEMS ANALYST,3698

2011 BUSINESS ANALYST,3891

2011 SYSTEMS ANALYST,8644

2011 COMPUTER PROGRAMMER,8998

2011 SOFTWARE ENGINEER,12763

2011 PROGRAMMER ANALYST,31799

2012 SENIOR CONSULTANT,3737

2012 ASSISTANT PROFESSOR,3801

2012 PHYSICAL THERAPIST,3871

2012 SOFTWARE DEVELOPER,3895

2012 COMPUTER SYSTEMS ANALYST,4706

2012 BUSINESS ANALYST,4752

2012 SYSTEMS ANALYST,9296

2012 COMPUTER PROGRAMMER,9629

2012 SOFTWARE ENGINEER,14437

2012 PROGRAMMER ANALYST,33066

2013 SENIOR CONSULTANT,4326

2013 SOFTWARE DEVELOPER,5026

2013 COMPUTER SYSTEMS ANALYST,5043

2013 BUSINESS ANALYST,5716

2013 TECHNOLOGY ANALYST - US,7683

2013 TECHNOLOGY LEAD - US,7853

2013 SYSTEMS ANALYST,8714

2013 COMPUTER PROGRAMMER,11271

2013 SOFTWARE ENGINEER,15680

2013 PROGRAMMER ANALYST,33880

2014 SENIOR CONSULTANT,4898

2014 TECHNOLOGY ANALYST - US,4913

2014 TECHNOLOGY LEAD - US,5057

2014 COMPUTER SYSTEMS ANALYST,6821

2014 BUSINESS ANALYST,7302

2014 SOFTWARE DEVELOPER,7337

2014 SYSTEMS ANALYST,10194

2014 COMPUTER PROGRAMMER,14950

2014 SOFTWARE ENGINEER,20500

2014 PROGRAMMER ANALYST,43114

2015 SENIOR SOFTWARE ENGINEER,6013

2015 TECHNOLOGY ANALYST - US,7014

2015 COMPUTER SYSTEMS ANALYST,7918

2015 TECHNOLOGY LEAD - US,8242

2015 BUSINESS ANALYST,8853

2015 SOFTWARE DEVELOPER,10441

2015 SYSTEMS ANALYST,12803

2015 COMPUTER PROGRAMMER,14054

2015 SOFTWARE ENGINEER,27259

2015 PROGRAMMER ANALYST,53436

2016 TECHNOLOGY LEAD - US,5410

2016 DEVELOPER,6084

2016 SENIOR SOFTWARE ENGINEER,6439

2016 COMPUTER SYSTEMS ANALYST,6900

2016 BUSINESS ANALYST,9167

2016 COMPUTER PROGRAMMER,11668

2016 SYSTEMS ANALYST,12314

2016 SOFTWARE DEVELOPER,14041

2016 SOFTWARE ENGINEER,30668

2016 PROGRAMMER ANALYST,53743

7) Create a bar graph to depict the number of applications for each year?

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.\*;

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

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.util.Waitable;

public class Task7 {

public static class map7 extends Mapper<LongWritable,Text,Text,Text>

{

public void map(LongWritable key,Text Value,Context con) throws IOException, InterruptedException

{

String[] rec=Value.toString().split("\t");

String year=rec[7];

String application=rec[3];

con.write(new Text(year),new Text(application));

}

}

public static class reducer7 extends Reducer<Text,Text,Text,IntWritable>

{

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

{

int count=0;

for(Text val:value)

{

count++;

}

con.write(key, new IntWritable(count));

}

}

public static void main(String[] args) throws IllegalArgumentException, IOException, ClassNotFoundException, InterruptedException

{

Configuration con=new Configuration();

Job job=Job.getInstance(con,"");

job.setJarByClass(Task7.class);

job.setMapperClass(map7.class);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(Text.class);

job.setReducerClass(reducer7.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

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

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

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

}

}

Execution:

hadoop jar Task7.jar Task7 /user/hive/warehouse/h1b.db/h1b /user/hduser/hadoop/Task7output;

hadoop fs -cat /user/hduser/hadoop/Task7output/p\*;

Output:

2011 358767

2012 415607

2013 442114

2014 519427

2015 618727

2016 647803

10) Which are the top 10 job positions which have the highest success rate in petitions?

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.IntWritable;

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.Mapper.Context;

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

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

public class Task10 {

public static class map\_grw extends

Mapper<LongWritable, Text, Text, IntWritable> {

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

throws IOException, InterruptedException {

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

String job\_title = str[4];

String application = (str[6]);

if (str[1].equals("CERTIFIED") || str[1].equals("CERTIFIED-WITHDRWAN")) {

con.write(new Text(job\_title), new IntWritable(1));

}

}

}

public static class reduce\_grwt extends

Reducer<Text, IntWritable, NullWritable, Text> {

private TreeMap<Long, Text> repToRecordMap = new TreeMap<Long, Text>();

public void reduce(Text key, Iterable<IntWritable> values,

Context context) throws IOException, InterruptedException {

int sum = 0;

for (IntWritable val : values) {

sum = sum + val.get();

}

String mytotal = String.format("%d", sum);

String myValue = key.toString();

myValue = myValue + ',' + mytotal;

repToRecordMap.put(new Long(sum), new Text(myValue));

if (repToRecordMap.size() > 10) {

repToRecordMap.remove(repToRecordMap.firstKey());

}

}

protected void cleanup(Context context) throws IOException,

InterruptedException {

for (Text t : repToRecordMap.values()) {

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

}

}

}

public static void main(String[] args) throws IOException,

ClassNotFoundException, InterruptedException {

Configuration conf = new Configuration();

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

job.setJarByClass(Task10.class);

job.setMapperClass(map\_grw.class);

job.setReducerClass(reduce\_grwt.class);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(IntWritable.class);

job.setOutputKeyClass(NullWritable.class);

job.setOutputValueClass(IntWritable.class);

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

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

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

}

}

Execution:

hadoop jar Task10.jar Task10 /user/hive/warehouse/h1b.db/h1b /user/hduser/hadoop/Task10output;

hadoop fs -cat /user/hduser/hadoop/Task10output/p\*;

Output:

SENIOR SOFTWARE ENGINEER,23547

TECHNOLOGY ANALYST - US,25985

TECHNOLOGY LEAD - US,28280

COMPUTER SYSTEMS ANALYST,31433

BUSINESS ANALYST,35204

SOFTWARE DEVELOPER,37771

SYSTEMS ANALYST,55744

COMPUTER PROGRAMMER,64018

SOFTWARE ENGINEER,102990

PROGRAMMER ANALYST,222730

**USE CASES IN HIVE:**

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

select count(\*) as rag,job\_title,year from h1b where job\_title =='DATA ENGINEER' group by job\_title,year order by year;

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

Select count(\*) as rag,job\_title from h1b group by job\_title order by rag desc limit 5;

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

select worksite,job\_title,year, count(\*) from h1b where job\_title=='DATA ENGINEER' group by worksite,job\_title,year;

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

select worksite, case\_status,year,count(worksite) as rag from h1b where case\_status == 'CERTIFIED' or case\_status == 'CERTIFIED-WITHDRAW' group by worksite,case\_status,year order by rag desc limit 5;

3) Which industry has the most number of Data Scientist positions?

select soc\_name,count(job\_title) as rag from h1b where job\_title== 'DATA SCIENTIST' group by soc\_name order by rag;

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

select count(case\_status) as rag,year,employer\_name from h1b group by employer\_name,year order by rag desc limit 5;

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

select job\_title,year from(select rank() over(partition by year order by rag desc)as rag1,job\_title,year from(select count(job\_title)as rag,job\_title,year from h1b group by job\_title,year)a)b where rag1<=10;

6) 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.

Select case\_status,year,(raghav\*100)/a.roc from (Select count(\*) as raghav ,case\_status,year from h1b group by case\_status,year) b,(select count(\*) as roc from h1b) a order by case\_status;

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

select year,count(soc\_name) as applications from h1b group by year;

8) Find the average Prevailing Wage for each Job for each Year ?

select avg(prevailing\_wage),job\_title,year from h1b group by job\_title,year,FULL\_TIME\_POSITION order by job\_title;

9. Which are top ten employers who have the highest success rate in petitions?

Select employer\_name,(rag\*100)/rag1 as top10 from (select count(\*) as rag1,1 as tojoin from h1b)b join (Select count(\*) as rag,1 as tojoin,employer\_name from h1b where case\_status="CERTIFIED" or case\_status="CERTIFIED WITHDRAWN" group by employer\_name) a on a.tojoin=b.tojoin order by top10 limit 10;

10) Which are the top 10 job positions which have the highest success rate in petitions?

Select job\_title,(top\*100)/rag1 as top10 from (Select count(\*) as top,job\_title from h1b where case\_status= "CERTIFIED" or case\_status="CERTIFIED WITHDRAWN" group by job\_title )b,(Select count(\*) as rag1 from h1b) a order by top10 limit 10;

**USE CASES IN PIG:**

h1b= LOAD '/user/hive/warehouse/h1b.db/h1b' using PigStorage('\t') AS

(s\_no:int,case\_status:chararray,employer\_name:chararray,soc\_name:chararray,job\_title:chararray,full\_time\_position:chararray,prevailing\_wage:double,year:int ,worksite1:chararray,longitute:double, latitute:double);

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

fil = filter h1b by job\_title == 'DATA ENGINEER';

disp = foreach fil generate soc\_name,year;

grupby = group disp by $1;

con = foreach grupby generate $0,COUNT(disp.soc\_name);

dump con;

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

disp = foreach h1b generate job\_title;

grupby = group disp by $0;

con = foreach grupby generate $0,COUNT(disp.$0);

ord = order con by $1 desc;

lmt = limit ord 5;

dump lmt;

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

fil = FILTER h1b by job\_title == 'DATA ENGINEER' ;

dis =foreach fil generate job\_title,year;

grp2013 = group dis by (job\_title,year);

cunt2013 = foreach grp2013 generate $0.year,$0.job\_title,COUNT(dis.$0) as TEngr;

yer = group cunt2013 by $0;

top5 = foreach yer { sorted = order cunt2013 by TEngr desc ; top = limit sorted 5 ; generate flatten(top); };

dump top5;

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

fil = FILTER h1b by case\_status == 'CERTIFIED' or case\_status == 'CERTIFIED-WITHDRAWN';

dis =foreach fil generate worksite1,year;

grp2013 = group dis by (worksite1,year);

cunt2013 = foreach grp2013 generate $0.year,$0.worksite1,COUNT(dis.$0) as TaprdVisa;

yer = group cunt2013 by $0;

top5 = foreach yer { sorted = order cunt2013 by TaprdVisa desc ; top = limit sorted 5 ; generate flatten(top); };

dump top5;

3) Which industry has the most number of Data Scientist positions?

fil = filter h1b by job\_title == 'DATA SCIENTIST';

dis = foreach fil generate soc\_name,job\_title;

grup = group dis by soc\_name;

con = foreach grup generate group,COUNT(dis.job\_title) as total;

odr = order con by total desc;

dump odr;

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

dis =foreach h1b generate employer\_name,year;

grp2013 = group dis by (employer\_name,year);

cunt2013 = foreach grp2013 generate $0.year,$0.employer\_name,COUNT(dis.$0) as petin;

yer = group cunt2013 by $0;

top5 = foreach yer { sorted = order cunt2013 by petin desc ; top = limit sorted 5 ; generate flatten(top); };

dump top5;

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

dis =foreach h1b generate job\_title,year;

grp2013 = group dis by (job\_title,year);

cunt2013 = foreach grp2013 generate $0.year,$0.job\_title,COUNT(dis.$0) as Tapplications;

yer = group cunt2013 by $0;

top10 = foreach yer { sorted = order cunt2013 by Tapplications desc ; top = limit sorted 5 ; generate flatten(top); };

dump top10;

6) 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.

totalrecgrup = group h1b ALL;

totalrec = foreach totalrecgrup generate COUNT(h1b.soc\_name)as totalApplications;

dis = foreach h1b generate year,case\_status;

grp = GROUP dis by year;

con = foreach grp generate $0,COUNT($1) as case\_application;

perc = foreach con generate $0,ROUND\_TO(((case\_application/(double)totalrec.totalApplications)\*100),2);

odr = order perc by $1 desc;

dump odr;

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

dis = foreach h1b generate year,soc\_name;

groupby = group dis by year;

con = foreach groupby generate $0,COUNT(dis.soc\_name);

dump con;

8) Find the average Prevailing Wage for each Job for each Year (take part time and full time separate)

fil = filter h1b by full\_time\_position == 'Y';

tme\_Y = group fil by (job\_title,year);

sum\_Y = foreach tme\_Y generate group,AVG(fil.prevailing\_wage) as AVG\_Y;

odr\_Y = order sum\_Y by $1 desc;

fil = filter h1b by full\_time\_position == 'N';

tme\_N = group fil by (job\_title,year);

sum\_N = foreach tme\_N generate group,AVG(fil.prevailing\_wage) as AVG\_N;

odr\_N = order sum\_N by $1 desc;

join1 = join sum\_Y by $0,sum\_N by $0;

for = foreach join1 generate $0,$1,$3;

odr = order for by $1 desc;

lim = limit odr 4;

dump lim;

9) Which are top ten employers who have the highest success rate in petitions?

totalrecgrup = group h1b ALL;

totalrec = foreach totalrecgrup generate COUNT(h1b.soc\_name)as totalApplications;

rec = foreach h1b generate employer\_name,case\_status;

fil\_suc = FILTER rec by case\_status == 'CERTIFIED' or case\_status == 'CERTIFIED-WITHDRAWN';

rec1 = foreach fil\_suc generate employer\_name,case\_status;

groupby = group rec1 by employer\_name;

coutofEmplye = foreach groupby generate $0,COUNT(rec1.case\_status) as cerftifiedApplications;

perc = foreach coutofEmplye generate $0,(cerftifiedApplications/(double)totalrec.totalApplications)\*100;

odr = order perc by $1 desc;

lim = LIMIT odr 10;

dump lim;

10) Which are the top 10 job positions which have the highest success rate in petitions?

totalrecgrup = group h1b ALL;

totalrec = foreach totalrecgrup generate COUNT(h1b.soc\_name)as totalApplications;

rec = foreach h1b generate job\_title,case\_status;

fil\_suc = FILTER rec by case\_status == 'CERTIFIED' or case\_status == 'CERTIFIED-WITHDRAWN';

rec1 = foreach fil\_suc generate job\_title,case\_status;

groupby = group rec1 by job\_title;

coutofEmplye = foreach groupby generate $0,COUNT(rec1.case\_status) as cerftifiedApplications;

d = foreach coutofEmplye generate $0,(((cerftifiedApplications/(double)totalrec.totalApplications)\*100));

odr = order d by $1 desc;

lim = LIMIT odr 10;

dump lim;