2

INTRODUCTION TO BIG DATA AND HADOOP

Unit Structure

2.0 OBJECTIVES

After going through this unit, you will be able to:

- Define data analytics on huge data sets
- Extract meaningful insights, such as hidden patterns and unknown correlations.
- Explain the steps in data pre-processing
- describe the dimensionality of data
- learn the data reduction and data compression techniques

2.1 INTRODUCTION TO HADOOP

Hadoop is a Java-based Apache open-source framework that allows for the distributed processing of large datasets across clusters of computers using simple programming models. The Hadoop framework application operates in a computing environment that allows for distributed storage and computation across computer clusters. Hadoop is intended to scale from a single server to thousands of machines, each of which provides local computation and storage.

Why use Hadoop:

- Apache Hadoop is not only a storage system but also data storage and processing platform.
- It is expandable (as we can add more nodes on the fly).
- Tolerant to faults (Even if nodes go down, data is processed by another node).
- It efficiently processes large amounts of data on commodity hardware in a cluster.
- Hadoop is used to process massive amounts of data.
- Commodity hardware is low-end hardware; it consists of inexpensive devices that are very cost-effective. As a result, Hadoop is very cost-effective.

Hadoop modules:

- **HDFS:** Hadoop Distributed File System Google released its GFS paper, and HDFS was built on top of it. It specifies that the files will be divided into blocks and stored in distributed architecture nodes.
- **Yarn:** Yet another Resource Negotiator that is used for job scheduling and cluster management.
- **Map Reduce:** is a framework that allows Java programmes to perform parallel computations on data using key-value pairs. The Map task converts input data into a data set that can be computed in Key value pair. The output of the Map task is consumed by the Reduce task, and the output of the Reducer produces the desired result.
- **Hadoop common:** These Java libraries are used to get things started.

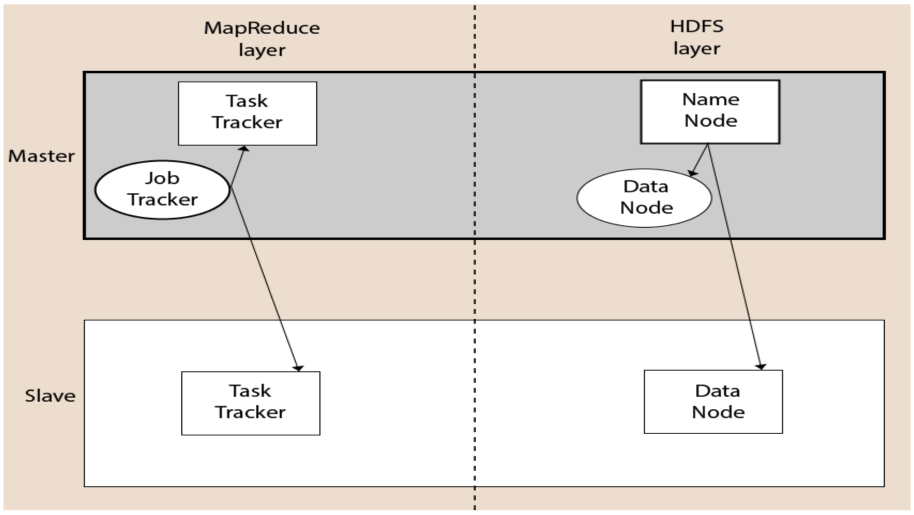


Figure 2.1 Framework of Hadoop

2.2 HISTORY OF HADOOP

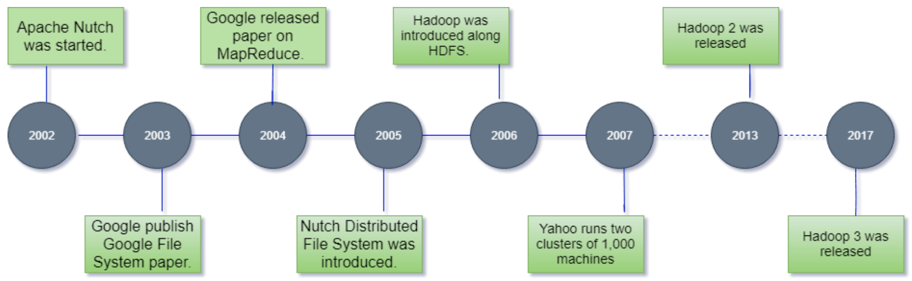

Apache Nutch was started.
Google released paper on MapReduce.
Hadoop was introduced along HDFS.
Hadoop 2 was released
2002
2003
2004
2005
2006
2007
2013
2017
Google publish Google File System paper.
Nutch Distributed File System was introduced.
Yahoo runs two clusters of 1,000 machines
Hadoop 3 was released

Figure 2.2 History of Hadoop

- Doug Cutting and Mike Cafarella founded Hadoop in 2002. Its origins can be traced back to the Google File System paper, which was published by Google. Let's focus on the history of Hadoop in the following steps:
- Doug Cutting and Mike Cafarella began working on Apache Nutch in 2002. It is a web crawler software project that is open source.
- They were dealing with large amounts of data while working on Apache Nutch. They have to spend a lot of money to store that data, which is the result of that project. This problem becomes one of the primary reasons for the development of Hadoop.
- Google introduced the GFS file system in 2003. (Google file system). It is a proprietary distributed file system designed to provide fast data access.
- Google published a white paper on Map Reduce in 2004. This method streamlines data processing on large clusters.
- Doug Cutting and Mike Cafarella introduced the NDFS file system in 2005. (Nutch Distributed File System). Map reduce is also included in this file system.
- Doug Cutting left Google in 2006 to join Yahoo. Dough Cutting introduces a new project Hadoop with a file system known as HDFS based on the Nutch project (Hadoop Distributed File System). This year saw the release of Hadoop's first version, 0.1.0.
- Doug Cutting named his Hadoop project after his son's toy elephant.
- Yahoo operates two 1000-machine clusters in 2007.
- Hadoop became the fastest system in 2008, sorting 1 terabyte of data on a 900-node cluster in 209 seconds.
- Hadoop 2.2 was released in 2013.
- Hadoop 3.0 was released in 2017.

2.3 HADOOP ARCHITECTURE

The Hadoop architecture consists of the file system, the MapReduce engine, and the HDFS (Hadoop Distributed File System). MapReduce engines are either MapReduce / MR1 or YARN / MR2.

A Hadoop cluster is made up of a single master and several slave nodes. Job Tracker, Task Tracker, Name Node, and Data Node comprise the master node, while Data Node and Task Tracker comprise the slave node.

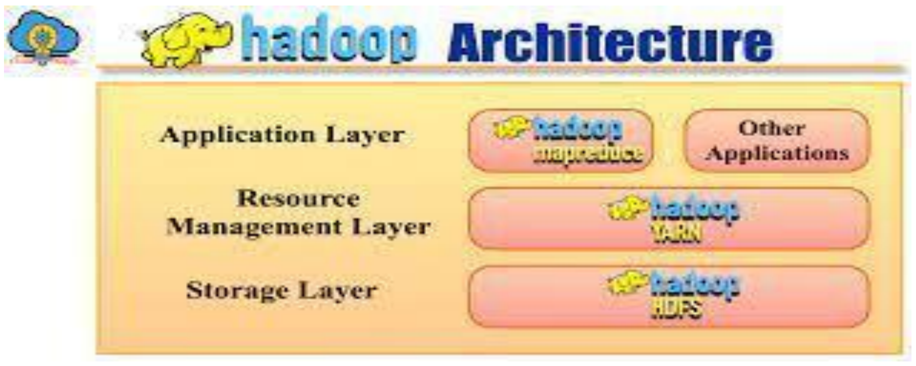


Figure 2.3 Hadoop Architecture

2.4 HADOOP DISTRIBUTED FILE SYSTEM

The Hadoop Distributed File System (HDFS) is a Hadoop distributed file system. It is built with master/slave architecture. This architecture consists of a single Name Node acting as the master and multiple Data Nodes acting as slaves.

Name Node and Data Node are both capable of running on commodity machines. HDFS is written in the Java programming language. As a result, the Name Node and Data Node software can be run on any machine that supports the Java programming language.

The Hadoop Distributed File System (HDFS) is a distributed file system based on the Google File System (GFS) that is designed to run on commodity hardware. It is very similar to existing distributed file systems. However, there are significant differences between it and other distributed file systems. It is highly fault-tolerant and is intended for use on low-cost hardware. It allows for high-throughput access to application data and is appropriate for applications with large datasets. In the HDFS cluster, there is only one master server.

- Because it is a single node, it may be the source of a single point failure.
- It manages the file system namespace by performing operations such as file opening, renaming, and closing.
- It simplifies the system's architecture.

Name Node:

- In the HDFS cluster, there is only one master server.
- Because it is a single node, it may be the source of a single point failure.
- It manages the file system namespace by performing operations such as file opening, renaming, and closing.
- It simplifies the system's architecture.

Data Node:

- There are several Data Nodes in the HDFS cluster.
- Each Data Node contains a number of data blocks.
- These data blocks are used to save information.
- It is Data Node's responsibility to handle read and write requests from file system clients.
- When instructed by the Name Node, it creates, deletes, and replicates blocks.

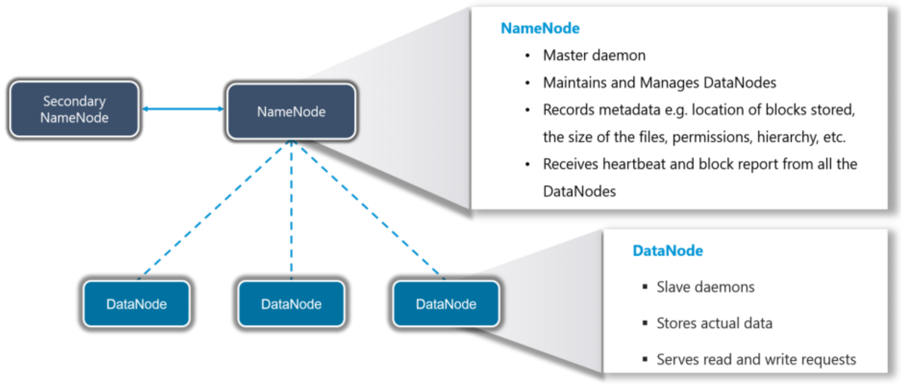

Secondary NameNode
NameNode
DataNode
DataNode
DataNode
NameNode
Master daemon
Maintains and Manages DataNodes
Records metadata e.g. location of blocks stored, the size of the files, permissions, hierarchy, etc.
Receives heartbeat and block report from all the DataNodes
DataNode
Slave daemons
Stores actual data
Serves read and write requests

Figure 2.4: HDFS

Job Tracker:

- Job Tracker's role is to accept MapReduce jobs from clients and process the data using Name Node.
- Name Node responds by sending metadata to Job Tracker.

Task Tracker:

- It performs the function of a slave node for Job Tracker.
- It receives the task and code from Job Tracker and applies it to the file. This procedure is also known as a Mapper.

Mapreduce Layer:

When the client application submits the MapReduce job to Job Tracker, the MapReduce is created. As a result, the Job Tracker forwards the request to the relevant Task Trackers. The Task Tracker occasionally fails or times out. That portion of the job is rescheduled in such a case.

2.4.1 Advantages and Disadvantages of HDFS:

Advantages of HDFS:

The advantages include its low cost, immutability, ability to store data consistently, tolerability of errors, scalability, block structure, capacity to handle massive amounts of data concurrently, and many others.

Disadvantages of HDFS:

The main problem is that it is not suitable for little amounts of data. It also has difficulties with possible stability, as well as being restricting and abrasive in character.

Apache Flumes, Apache Oozie, Apache HBase, Apache Sqoop, Apache Spark, Apache Storm, Apache Pig, Apache Hive, Apache Phoenix, and Cloudera Impala are also supported.

2.4.2 How Does Hadoop Work?

It is quite expensive to build larger servers with heavy configurations that handle large-scale processing, but as an alternative, you can connect many commodity computers with single-CPU as a single functional distributed system, and the clustered machines can read the dataset in parallel and provide much higher throughput. Furthermore, it is less expensive than purchasing a single high-end server. So, the fact that Hadoop runs on clustered and low-cost machines is the first motivator for using it.

- Initially, data is organized into directories and files.
- The files are divided into 128M and 64M uniformly sized blocks (preferably 128M).
- The processing is overseen by HDFS, which sits on top of the local file system.
- To handle hardware failure, blocks are replicated.
- Checking to see if the code was successfully executed.
- Executing the sort that occurs between the maps and reduce stages.
- Sending sorted data to a specific computer.
- For each job, create debugging logs.

2.4.3 Where is Hadoop Used?

Hadoop is used for:

- Search – Yahoo, Amazon, Zvents
- Log processing – Facebook, Yahoo
- Data Warehouse – Facebook, AOL
- Video and Image Analysis – New York Times, Eyealike

Till now, we have seen how Hadoop has made Big Data handling possible. But there are some scenarios where Hadoop implementation is not recommended.

When not to use Hadoop?

Following are some of those scenarios:

- Low Latency data access: Quick access to small parts of data
- Multiple data modification: Hadoop is a better fit only if we are primarily concerned about reading data and not modifying data.
- Lots of small files: Hadoop is suitable for scenarios, where we have few but large files.

After knowing the best suitable use-cases, let us move on and look at a case study where Hadoop has done wonders.

2.5 OVERVIEW OF YARN

Hadoop's resource management layer is Apache Yarn, which stands for "***Yet Another Resource Negotiator."*** The yarn was first introduced in Hadoop 2. x. Yarn enables various data processing engines, including as graph processing, interactive processing, stream processing, and batch processing, to operate and handle HDFS data (Hadoop Distributed File System). Yarn offers work schedule in addition to resource management.

Yarn extends Hadoop's capabilities to other developing technologies, allowing them to benefit from HDFS (the most stable and popular storage system on the globe) and economic cluster.

Apache yarn is also a Hadoop 2.x data operating system. This Hadoop 2.x architecture provides a general-purpose data processing platform that is not confined to MapReduce.

It enables Hadoop to handle systems other than MapReduce that are purpose-built for data processing. It enables the execution of many frameworks on the same hardware as Hadoop.

The Figure below demonstrates the architecture of Hadoop YARN.

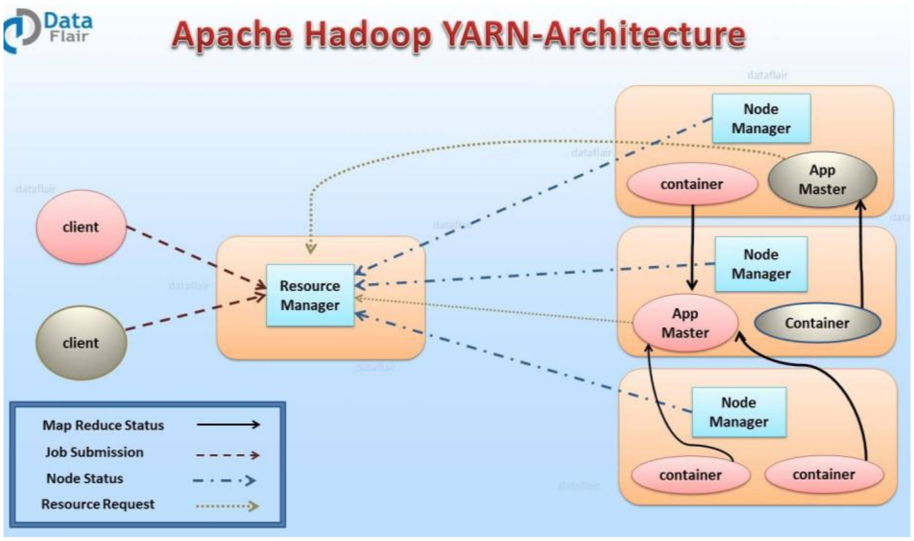


Figure 2.5 Apache Hadoop Yarn Architecture

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

- **Resource Manager (RM):**

It is Yarn's master daemon. The global allocation of resources (CPU and memory) across all apps is managed by RM. It resolves conflicts over system resources between competing programs. To study Yarn Resource Manager in depth, see to the Resource Manager guide. The main components of Resource Manager are:

✔ Application manager

✔ Scheduler

Application Manager:

It handles the running of Application Masters in the cluster, which means it is in charge of initiating application masters as well as monitoring and restarting them on various nodes in the event of a failure.

Scheduler:

The scheduler is in charge of distributing resources to the currently executing programme. The scheduler is a pure scheduler, which means it conducts no monitoring or tracking for the application and makes no assurances regarding resuming unsuccessful jobs due to application or hardware issues.

- **Node Manager (NM):**

It is Yarn's slave daemon. NM is in charge of monitoring container resource utilization and reporting it to the Resource Manager. On that computer, manage the user process. Yarn Node Manager additionally monitors the node on which it is executing. Long-running auxiliary services can also be plugged into the NM; these are application-specific services that are defined as part of the settings and loaded by the NM upon start-up. A shuffle is a common auxiliary service provided by NMs for MapReduce applications running on YARN.

- **Application Master (AM):**

Each application has its own application master. It interacts with the node manager to negotiate resources from the resource management. It is in charge of the application life cycle.

The AM obtains containers from the RM's Scheduler before contacting the associated NMs to begin the different tasks of the application.

2.6 FEATURES OF YARN

The following characteristics contributed to the popularity of ARN:

- **Scalability:** The scheduler in the YARN architecture's Resource management enables Hadoop to extend and manage thousands of nodes and clusters.
- **Compatibility:** Because YARN supports existing map-reduce applications without interruption, it is also compatible with Hadoop 1.0.
- **Cluster Usage:** Because YARN offers dynamic cluster utilization in Hadoop, optimal Cluster Utilization is possible.
- **Multi-tenancy:** It provides businesses with the benefit of multi-tenancy by allowing multiple engine access.

2.7 APPLICATION WORKFLOW IN HADOOP YARN

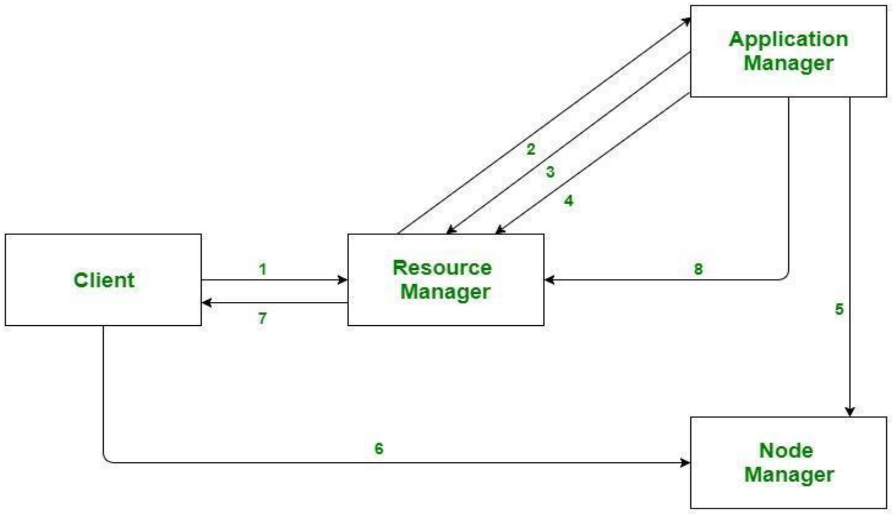


Figure 2.6: Application Workflow

MODULE II

3

HDFS

Unit Structure

3.0 INTRODUCTION TO HDFS

Data has grown exponentially over the period of time leading world towards big data. To store data which was majorly structured previously was easier as compared to data which is mixture of structured and unstructured also in huge volumes. To store and manage such data we require a network of systems i.e. distributed file systems. As they are network-based, all the complications of network programming add up, thus making distributed file systems more complex than regular disk filesystems. One of the vital challenges is making network fault tolerant that means if any of the node in network fails then it should not cause any data loss.

Hadoop is free. It is Java-based. Hadoop was initially released in 2011. It is part of the Apache project sponsored by the Apache Software Foundation (ASF). It was inspired by Google's MapReduce, a software framework in which an application is broken down into numerous small parts. It was named after its creator's (Doug Cutting) child's stuffed toy elephant Hadoop consists of the Hadoop kernel, MapReduce, the Hadoop Distributed File System (HDFS), and a number of related projects such as Apache Hive, HBase, and Zookeeper. The Hadoop framework is used by major players, including Google, Yahoo, and IBM, largely for applications involving search engines and advertising. The preferred operating systems are Windows and Linux, but Hadoop can also work with BSD and OS X.

Hadoop has introduced a distributed file system called as Hadoop Distributed File Systems which is abbreviated as HDFS.

HDFS provides redundant storage for big data by storing that data across a cluster of cheap, unreliable computers, thus extending the amount of available storage capacity that a single machine alone might have. However, because of the networked nature of a distributed file system, HDFS is more complex than traditional file systems. In order to minimize that complexity, HDFS is based off of the centralized storage architecture.[2]

In principle, HDFS is a software layer on top of a native file system such as ext4 orxfs, and in fact Hadoop generalizes the storage layer and can interact with local file systems and other storage types like Amazon S3. However, HDFS is the flagship distributed file system, and for most programming purposes it will be the primary file system you'll be interacting with. HDFS is designed for storing very large files with streaming data access, and as such, it comes with a few caveats:

- HDFS performs best with a modest number of very large files—for example, millions of large files (100 MB or more) rather than billions of smaller files that might occupy the same volume.
- HDFS implements the WORM pattern. Write once and read a lot. Random writes or file appends are prohibited.
- HDFS is optimized for streaming large file reads rather than random reads or selection.

HDFS is therefore best suited for storing raw input data for computations, intermediate results between computation steps, and final results for the entire operation. do.

Not suitable as a data backend for applications that require real-time updates, interactive data analysis, or record-based transaction support. Instead, HDFS users tend to create large heterogeneous data stores to aid in various calculations and analysis by writing data only once and reading it many times. These repositories are also referred to as "data lakes" because they simply store all data for known issues in a recoverable and fault-tolerant manner.

3.1 HDFS ARCHITECTURE

HDFS is particularly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS gives excessive throughput access to software records and is suitable for programs that have massive datasets. HDFS relaxes some POSIX requirements to permit streaming get right of entry to document system data. HDFS changed into firstly built as infrastructure for the Apache Nutch web search engine undertaking. HDFS is part of the Apache Hadoop core undertaking.

3.1.0 NameNode and DataNodes:

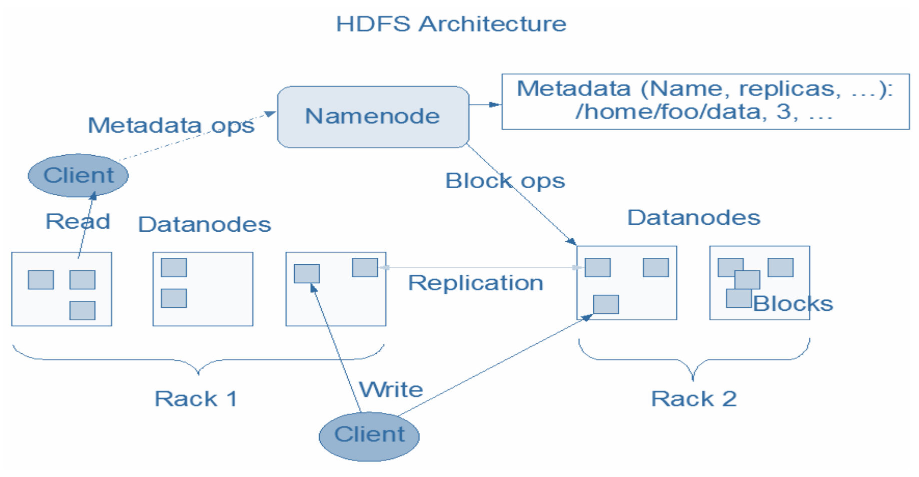


fig: 3.1 HDFS Architecture[1]

HDFS has a master/slave architecture.

An HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients. It maintains the file system tree and metadata for all the files and directories in the tree. This information is stored on the local disk in the form of two files: the namespace image and the edit log.

Along with it there are a some DataNodes, usually one per node in the cluster, which manage storage attached to the nodes that they run on. The Data Nodes are also called as worker nodes. HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes.

The NameNode also knows the DataNodes on which all the blocks for a given file are located, however, it does not store block locations, since this information is reconstructed from DataNodes when the system starts. The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes.

The DataNodes are responsible for serving read and write requests from the file system's clients. The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode.

It is important to make NameNode fault tolerant as without the NameNode the file system cannot be used. To make is fault tolerant the first step will be taking back-ups for the persistent state files and metadata present in it. Hadoop can be configured so that the NameNode writes its persistent state to multiple filesystems. These writes are synchronous and atomic. The usual configuration choice is to write to local disk as well as a remote NFS mount.

It is also possible to run a secondary NameNode, which has the same name but it still does not act as a NameNode. Its main role is to periodically merge the namespace image with the edit log to prevent the edit log from becoming too large. The secondary NameNode usually runs on a separate physical machine, since it requires plenty of CPU and as much memory as the NameNode to perform the merge. It keeps a copy of the merged namespace image, which can be used in the event of the NameNode failing. However, the state of the secondary NameNode lags that of the primary, so in the event of total failure of the primary, data loss is almost certain. The usual course of action in this case is to copy the NameNode's metadata files that are on NFS to the secondary and run it as the new primary.

3.1.1 The File System Namespace:

HDFS supports a traditional hierarchical file organization. A user or an application can create a directory and store files in that directory. The namespace hierarchy of a file system is similar to most other file systems that exist. You can create and delete files, move files from one directory to another, or rename files. HDFS supports user quotas and permissions. HDFS does not support hard or soft links. However, the HDFS architecture does not preclude the implementation of these features.

HDFS follows the file system naming convention, but some paths and names are reserved, such as /.reserved and .snapshot . Features such as transparent encryption and snapshots use reserved paths.

NameNode maintains the file system namespace. Any changes to the file system namespace or its properties are logged by the NameNode. Applications can specify the number of file replicas to maintain on HDFS. The number of copies of a file is called the replication factor of that file. This information is stored in NameNode.

3.1.2 Data Replication:

The HDFS block size is 64 MB by default, but many clusters use 128 MB (134,217,728 bytes) or even 256 MB (268,435,456 bytes) to ease memory pressure on the namenode and to give mappers more data to work on. Set this using the *dfs.block.size* property in *hdfs-site.xml*.

HDFS is designed to securely store very large files on computers in large clusters. Save each file as a block sequence. Blocks of files are duplicated for fault tolerance. Block size and replication factor are configurable for each file.

All blocks except the last block are the same size, but users can start a new block without filling the last block with the configured block size after variable length block support for append and hsync has been added.

Allows the application to specify the number of copies of a file. Replication factors can be specified during file creation and changed later.

A file in HDFS is written once (except appended and truncated) and has exactly one record at any time.

NameNode makes all decisions regarding block replication. Periodically receive Heartbeat and Blockreport messages from each data node in the cluster. Getting a heartbeat means the DataNode is working properly. Blockreport contains a list of all blocks in the DataNode.

1.1.3 Working with Distributed File System:

Basic File System Operations

To see the available commands in the fs shell, type:

```
hostname $ hadoop fs -help
```

```
Usage: hadoop fs [generic options]
```

As you can see, many of the familiar commands for interacting with the file system are there, specified as arguments to the hadoop fs command as flag arguments in theJava style—that is, as a single dash (–) supplied to the command. Secondary flags or options to the command are specified with additional Java style arguments delimited by spaces following the initial command. Be aware that order can matter when specifying such options. Consider a file named as shakespeare.txt

To copy the above file from one location to another type:

```
hostname $ hadoop fs –copyFromLocal shakespeare.txt
shakespeare.txt
```

In order to better manage the HDFS file system, create a hierarchical tree of directories just as you would on your local file system:

```
hostname $ hadoop fs -mkdir corpora
```

To list the contents of the remote home directory, use the ls command:

```
hostname $ hadoop fs –ls
```

```
drwxr-xr-x - analyst analyst 0 2015-05-04 17:58 corpora

-rw-r--r-- 3 analyst analyst 8877968 2015-05-04 17:52 shakespeare.txt
```

Other basic file operations like mv, cp, and rm will all work as expected on the remote file system. There is, however, no rmdir command; instead, use rm –R to recursively remove a directory with all files in it.

To read the contents of a file, use the cat command, then pipe the output to less in order view the contents of the remote file:

```
hostname $ hadoop fs –cat shakespeare.txt | less
```

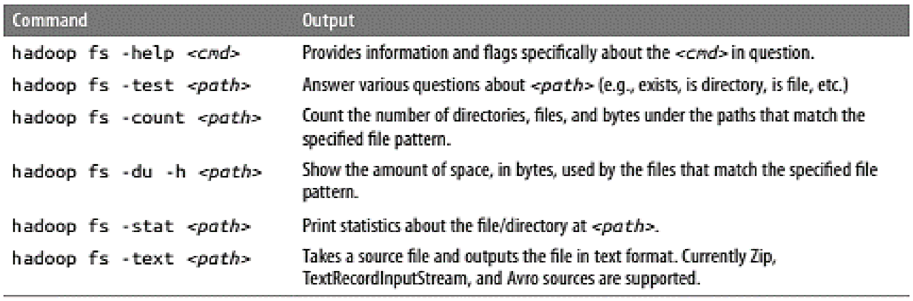
Alternatively, you can use the tail command to inspect only the last kilobyte of the file:

hostname $ hadoop fs –tail shakespeare.txt | less

To transfer entire files from the distributed file system to the local file system, use get or copyToLocal, which are identical commands. Similarly, use the moveToLocal command, which also removes the file from the distributed file system. Finally, the get merge command merges all files that match a given pattern or directory are copied and merged into a single file on the localhost. If files on the remote system are large, you may want to pipe them to a compression utility:

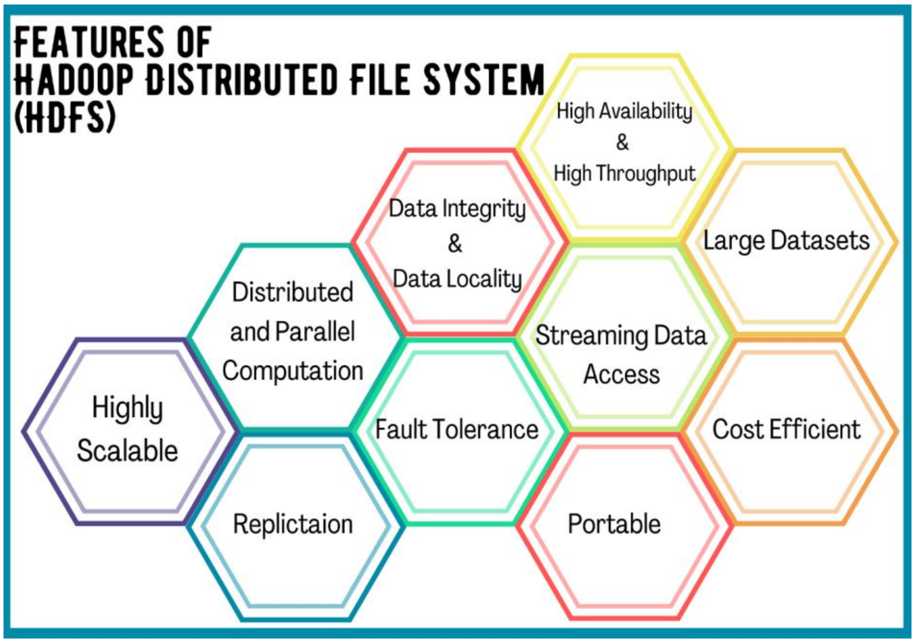
hostname $ hadoop fs –get shakespeare.txt ./shakespeare.from-remote.txt

Other useful utilities



Command	Output
`hadoop fs -help <cmd>`	Provides information and flags specifically about the *<cmd>* in question.
`hadoop fs -test <path>`	Answer various questions about *<path>* (e.g., exists, is directory, is file, etc.)
`hadoop fs -count <path>`	Count the number of directories, files, and bytes under the paths that match the specified file pattern.
`hadoop fs -du -h <path>`	Show the amount of space, in bytes, used by the files that match the specified file pattern.
`hadoop fs -stat <path>`	Print statistics about the file/directory at *<path>*.
`hadoop fs -text <path>`	Takes a source file and outputs the file in text format. Currently Zip, TextRecordInputStream, and Avro sources are supported.

3.2 FEATURES OF HDFS


FEATURES OF
HADOOP DISTRIBUTED FILE SYSTEM
(HDFS)
High Availability & High Throughput
Data Integrity & Data Locality
Large Datasets
Distributed and Parallel Computation
Streaming Data Access
Highly Scalable
Fault Tolerance
Cost Efficient
Replictaion
Portable

The major features of Hadoop Distributed File System (HDFS) are:

1. Distributed and Parallel Computation:

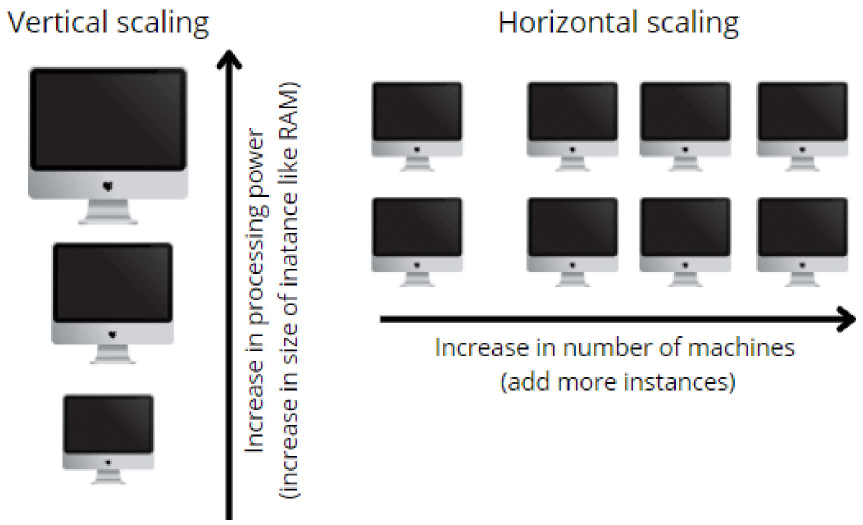
This is one of the most important features of the Hadoop Distributed File System (HDFS) and makes Hadoop a very powerful tool for big data storage and processing. Here, input data is divided into blocks called data blocks and stored on different nodes in the HDFS cluster.

2. Highly scalable:

There are basically two types of scaling: vertical and horizontal.

- Increases the hardware capacity of the system with scale-up (**vertical scaling)**. That is, add more memory, RAM, and CPU power to an existing system, or buy a new system with more memory, RAM, and CPU. However, there are many problems with this vertical scaling or scaling. First, there is always a limit to what you can increase hardware performance. So you simply cannot keep increasing the amount of memory, RAM or CPU on your machine. Second, when scaling, you must first stop the machine and then add resources to the existing machine. Reboot the system when you increase the performance of the device. Downtime that brings the system down is an issue.

- When you scale out (**horizontal scaling**), you add more nodes to an existing HDFS cluster instead of increasing the system's hardware capacity. And most importantly, you can add more machines on the go. without stopping the system. So, with horizontal scaling there is no downtime. Here you can run more machines in parallel to achieve the main Hadoop goal.


Vertical scaling
Increase in processing power
(increase in size of inatance like RAM)
Horizontal scaling
Increase in number of machines
(add more instances)

3. Replication:

Data Replication is one of the most important and unique features of HDFS. In HDFS replication of data is done to solve the problem of data loss in unfavorable conditions like crashing of a node, hardware failure, and so on.

The data is replicated across a number of machines in the cluster by creating replicas of blocks. The process of replication is maintained at regular intervals of time by HDFS and HDFS keeps creating replicas of user data on different machines present in the cluster. Hence whenever any machine in the cluster gets crashed, the user can access their data from other machines that contain the blocks of that data. Hence there is no possibility of a loss of user data.

4. Fault tolerance:

HDFS is highly fault tolerant and reliable. HDFS creates replicas of file blocks depending on the replication factor and stores them on different machines.

If one of the machines containing data blocks fails, another data node containing copies of these data blocks becomes available. This guarantees data loss and makes the system stable even under adverse conditions.

Hadoop 3 introduces erasure coding for fault tolerance. Erasure Coding in HDFS improves storage efficiency by providing the same level of resiliency and data reliability as traditional replication-based HDFS deployments.

5. Streaming Data Access:

The write-once/read-many design is intended to facilitate streaming reads.

Hadoop Streaming acts as an interface between Hadoop and the program written

6. Portable:

HDFS is designed in such a way that it can easily portable from platform to another.

7. Cost effective:

In HDFS architecture, the DataNodes, which stores the actual data are inexpensive commodity hardware, thus reduces storage costs.

8. Large Datasets:

HDFS can store data of any size (ranging from megabytes to petabytes) and of any formats (structured, unstructured).

9. High Availability:

Hadoop's high availability feature ensures data availability even if a NameNode or DataNode fails. HDFS creates copies of data blocks, so if one of the data nodes goes down, users can access the data on another data node that contains a copy of the same data block.

Also, if the active NameNode fails, the passive node takes over the responsibility of the active NameNode. In this way, data can be accessed and used by users during machine failure.

10. High Throughput:

HDFS stores data in a distributed fashion, allowing parallel processing of data across clusters of nodes. This reduces processing time and ensures high throughput.

11. Data Integrity:

Data integrity refers to the correctness of data. HDFS ensures data integrity by continuously checking data against checksums computed as files are written.

When reading a file, if the checksum does not match the original checksum, the data is said to be corrupted. The client then chooses to receive the data block from another DataNode that has a copy of that block. NameNode discards bad blocks and creates additional new copies.

12. Data Locality:

Data locality means moving computation logic to the data rather than moving data to the computational unit. Data Locality means it co-locate the data with the computing nodes.

In the traditional system, the data is brought at the application layer and then gets processed. But in the present scenario, due to the massive volume of data, bringing data to the application layer degrades the network performance.

In HDFS, we bring the computation part to the Data Nodes where data resides. Hence, with Hadoop HDFS, we are not moving computation logic to the data, rather than moving data to the computation logic. This feature reduces the bandwidth utilization in a system.

3.3 RACK AWARENESS

What is a rack?

The Rack is the collection of around 40-50 DataNodes connected using the same network switch. If the network goes down, the whole rack will be unavailable. A large Hadoop cluster is deployed in multiple racks.

What is Rack Awareness in Hadoop HDFS?

Hadoop components are not supported. For example, HDFS block placement provides fault tolerance by using rack awareness to place replicas of one block in another rack. This ensures data availability in the event of a network switch or partition failure within the cluster.

Hadoop master daemon obtains the cluster worker rack ID by calling an external script or Java class as specified in the configuration file. If you use Java classes or external scripts for your topology, the output must conform to the Java interface **org.apache.hadoop.net.DNSToSwitchMapping**. The interface expects a one-to-one correspondence to be maintained and the topology information in the format of ‘/myrack/myhost’, where ‘/’ is the topology delimiter, ‘myrack’ is the rack identifier, and ‘myhost’ is the individual host. Assuming a single /24 subnet per rack, one could use the format of ‘/193.168.100.0/193.168.100.5’ as a unique rack-host topology mapping.

To use the java class for topology mapping, the class name is specified by the **net.topology.node.switch.mapping.impl** parameter in the configuration file. For example, NetworkTopology.java is included in the Hadoop distribution and can be customized by the Hadoop administrator. Using Java classes instead of external scripts has performance benefits because Hadoop does not have to fork external processes when new worker nodes are registered.

When implementing an external script, it is specified as the **net.topology.script.file.name** parameter in the configuration file. Unlike Java classes, external topology scripts are not included in the Hadoop distribution and are provided by the administrator. Hadoop sends multiple IPs to ARGV when forking topology scripts. The number of IP addresses sent to the topology script is controlled by the **net.topology.script.number.args** setting, which defaults to 100. For **net.topology.script.number.args** is changed to 1, the topology script will branch for each IP address represented by the data node and/or node manager.

If **net.topology.script.file.name** or **net.topology.node.switch.mapping.impl** is not set, the rack ID '/defaultrack' is returned for all forwarded IP addresses. Although this behaviour seems desirable, it can cause problems with HDFS block replication because the default behaviour is to write one replicated block outside the rack, which cannot be done because there is only one rack named "/defaultrack".

Why Rack Awareness?

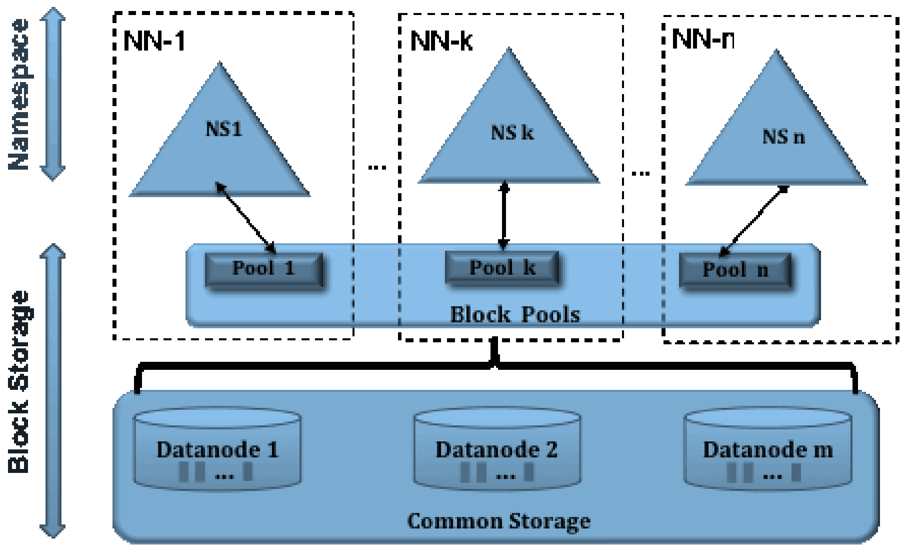
The reasons for the Rack Awareness in Hadoop are:

1. To reduce the network traffic while file read/write, which improves the cluster performance.

2. To achieve fault tolerance, even when the rack goes down (discussed later in this article).

3. Achieve high availability of data so that data is available even in unfavourable conditions.

4. To reduce the latency, that is, to make the file read/write operations done with lower delay.



The NameNode keeps a reference to every file and block in the filesystem in memory, which means that on very large clusters with many files, memory becomes the limiting factor for scaling.

HDFS Federation, introduced in the 0.23 release series, allows a cluster to scale by adding NameNodes, each of which manages a portion of the filesystem namespace. For example, one NameNode might manage all the files rooted under */user*, say, and a second NameNode might handle files under */share*.

Under federation, each NameNode manages a *namespace volume*, which is made up of the metadata for the namespace, and a *block pool* containing all the blocks for the files in the namespace. Namespace volumes are independent of each other, which means NameNodes do not communicate with one another, and furthermore the failure of one NameNode does not affect the availability of the namespaces managed by other NameNodes. However, block pool storage is not partitioned, so data nodes register with all NameNodes in the cluster and store blocks from multiple block pools. To access a HDFS federated cluster, the client uses a client-side mount table to map file path to a name node. This is controlled in the configuration by the *ViewFileSystem* and *viewfs://* URIs.

3.5 LIST OF REFERENCES

- [1]Apache Hadoop 3.3.2 – HDFS Architecture
- [2] This was first described in the 2003 paper by Ghemawat, Gobioff, and Leung, “The Google File System”.
- Apache Hadoop 3.3.2 – Rack Awareness
- Tom White, “HADOOP: The definitive Guide” O Reilly 2012, Third Edition, ISBN: 978-1-449-31152-0
- What is HDFS? Design Features of HDFS - VTUPulse

4

MAP REDUCE & ALGORITHMS

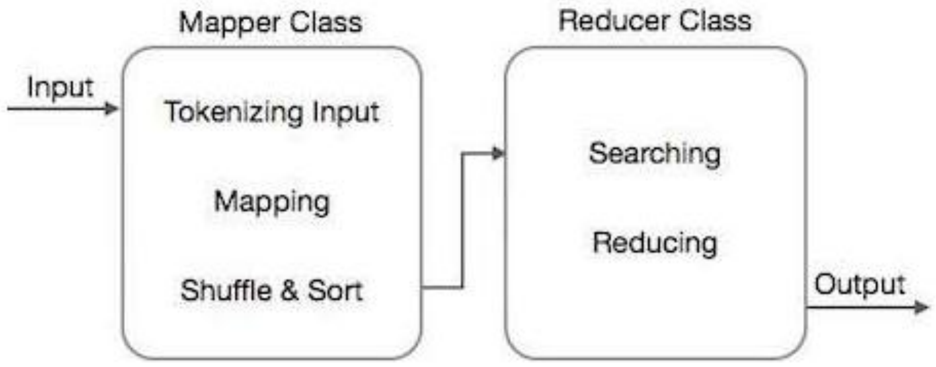
Unit Structure

4.0 INTRODUCTION TO MAP REDUCE

Businesses and governments need to analyse and process vast amounts of data in a very short amount of time. The processing has to be done on large amounts of data and is very time consuming if done on a single machine. So, the idea is to split the data into smaller chunks, send it to a cluster of machines that can be processed concurrently, and then merge the results. The huge growth of data generated by social networks and other blogging sites such as "Friends" on the social network site has increased the amount of graphic data with millions of nodes and edges. This created a new software stack. This new software stack brings parallelism to the using multiple publicly available hardware connected via Ethernet or a switch. Hadoop is a platform for large-scale distributed batch processing. Hadoop can be deployed on a single machine if it can process data, but its main purpose is to efficiently distribute large amounts of data across a set of machines for processing. Hadoop includes a Distributed File System (DFS) that splits the input data and sends pieces of the input data to multiple machines in a particular cluster for storage. The focus of this new software stack is on MapReduce, an advanced programming system. This helps to compute the problem in parallel using all connected machines, so that the output result is obtained in an efficient manner. DFS also provides up to 3x data replication to prevent data loss in case of media failure.

MapReduce is an easy-to-understand paradigm, but not easy to implement, especially in distributed systems. Scheduled according to the workload system.

- Tasks must be monitored and managed to ensure that all errors that occur are handled properly and that tasks continue to run in the event of a partial system failure.
- Inputs must be distributed across the cluster.
- Staged processing of input data should be done on a distributed system, preferably on the same system where the data resides.
- Reduction steps must be performed by collecting the intermediate results of multiple map steps and sending them to the appropriate machine.
- The final output must be available to other users, other applications, or other MapReduce jobs.



Hadoop MapReduce Applications:

Following are some practical applications of MapReduce programs.

E-commerce:

E-commerce companies such as Walmart, Ebay, and Amazon use MapReduce to analyze shopping behavior. MapReduce provides valuable information that is used as a basis for developing product recommendations. Some of the information used includes site history, e-commerce directories, purchase history, and interaction logs.

Social Networks:

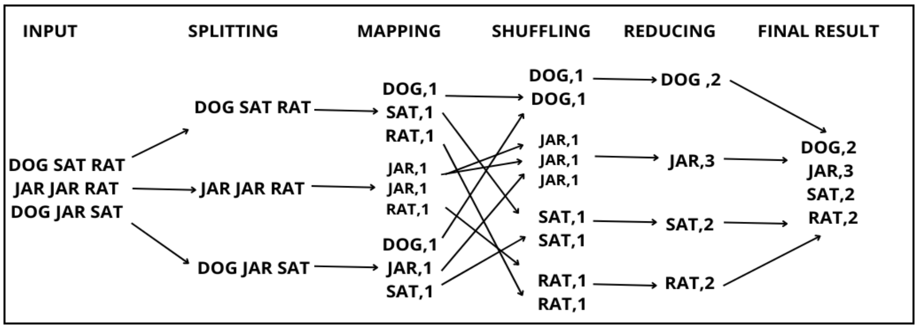
The MapReduce programming tool can evaluate certain information on social media platforms such as Facebook, Twitter and LinkedIn. You can rate important information such as who liked your status and who viewed your profile.

Entertainment:

Netflix uses MapReduce to analyze click and online customer logs. This information helps the company to offer movies based on the interests and behaviors of its customers.

4.1 MAP REDUCE ARCHITECTURE

The MapReduce framework improves job scheduling and monitoring. Failed tasks are re-executed by the framework. This framework is easy to use even for programmers with little experience in distributed processing. MapReduce can be implemented using a variety of programming languages, including Java, Hive, Pig, Scala, and Python.



Map Reduce Architecture

The applications that use MapReduce have the below advantages:

- They have been provided with convergence and good generalization performance.
- Data can be handled by making use of data-intensive applications.
- It provides high scalability.
- Counting any occurrences of every word is easy and has a massive document collection.
- A generic tool can be used to search tool in many data analysis.
- It offers load balancing time in large clusters.
- It also helps in the process of extracting contexts of user location, situations, etc.
- It can access large samples of respondents quickly.

4.1.1 The Map Task:

Split the input data set into chunks in a map operation. The Map Task processes these fragments in parallel. Map to use as input to the reduction problem.

4.1.2 The Reduce Task:

Reducers process the intermediate data from the maps into smaller tuples, that reduces the tasks, leading to the final output of the framework.

A developer can always set the number of the reducers to zero. That will completely disable the reduce step.

4.1.3 Grouping by Key:

In the MapReduce process, the mapper only understands the key-value pairs in the data, so before passing the data to the mapper, you must first transform the data into key-value pairs. In Hadoop MapReduce,key-value pairs are generated as follows using InputSplit and Record Reader.

By default, RecordReader uses TextInputFormat to convert data into key/value pairs. The RecordReader interacts with the InputSplit until it has finished reading the file.

In MapReduce, the map function processes a specific key-value pair and produces a specific number of key-value pairs, whereas the Reduce function processes the values grouped by the same key and produces a different set of key-value pairs as output. The output type of the map must match the input type of reduce as shown below.

Map: (K1, V1) > List(K2, V2)

Decrease: {(K2, List(V2 }) > List K3, V3)

Hadoop's generated key-value pairs depend on the dataset and desired output It varies, and typically key-value pairs are specified in four places: Map input, Map output, Reduce input, and Reduce output.

- **Map Input:**

Map-input by default will take the line offset as the key and the content of the line will be the value as Text.

- **Map Output:**

Map basic responsibility is to filter the data and provide the environment for grouping of data based on the key.

- o **Key:** It will be the field/ text/ object on which the data has to be grouped and aggregated on the reducer side.
- o **Value:** It will be the field/ text/ object which is to be handled by each individual reduce method.

- **Reduce Input:**

The output of Map is the input for reduce, so it is same as Map-Output.

- **Reduce Output:**

It depends on the required output.

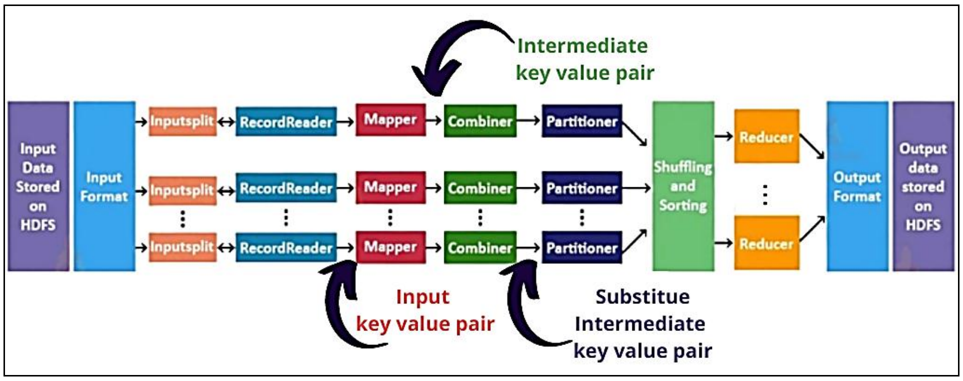
4.1.4 Partitioner and Combiners:

A partitioner partitions the key-value pairs in the Map intermediate output. Separate data using user-defined conditions that act like hash functions. The total number of sections equals the number of Reducer jobs per job.

A Combiner is also called and semi-reducer, is an optional step used to optimize the MapReduce process. Used to reduce output at node level. At this point, you can merge the cloned output from the map output into a single output file. The merge step speeds up the shuffle step by improving job performance.

Combiner class is used between Map class and Reduce class to reduce the amount of data transfer between Map and Reduce. In general, the output of the map job is large and the data passed to the reduce job is large.

4.1.5 Detail of Map Reduce Execution:



The data flow in MapReduce is the combination of different processing phases of such as Input Files, InputFormat in Hadoop, InputSplits, RecordReader, Mapper, Combiner, Partitioner, Shuffling and Sorting, Reducer, RecordWriter, and OutputFormat. Hence all these components play an important role in the Hadoop mapreduce working.

MapReduce processes the data in various phases with the help of different components. The steps of job execution in Hadoop.

Input Files:

The data for MapReduce job is stored in Input Files. Input files reside in HDFS. The input file format is random. Linebased log files and binary format can also be used.

InputFormat:

After that InputFormat defines how to divide and read these input files. It selects the files or other objects for input. InputFormat creates InputSplit.

InputFormat class calls the getSplits function and computes splits for each file and then sends them to the jobtracker.

InputSplits:

It represents the data that will be processed by an individual Mapper. For each split, one map task is created. Therefore, the number of map tasks is equal to the number of InputSplits. Framework divide split into records, which mapper process.

RecordReader:

It communicates with the InputSplit. And then transforms the data into key-value pairs suitable for reading by the Mapper. The RecordReader uses TextInputFormat by default to convert data into key/value pairs. Interact with InputSplit until the file is finished reading. Allocates an offset in bytes for each line in the file. These key/value pairs are then sent to the resolver for further processing.

Mapper:

It processes the input records generated by the RecordReader and generates intermediate key-value pairs. The intermediate output is completely different from the input pair. The result of the transformer is a complete group of key-value pairs. The Hadoop infrastructure does not store the output of the converter in HDFS. The mapper doesn't store it because the data is temporary and creates multiple non-write-free copies on HDFS. The mapper then passes the output to the combiner for further processing.

Combiner:

The Combiner is a Minireducer that performs local aggregation of mapper outputs. This minimizes data transfer between mappers and reducers. So when the combiner function completes, the platform passes the output to the splitter for further processing.

Partitioner:

Partitioner occurs when working with more than one reducer. Capture the output of the combiner and perform the split.

Output splits based on keys in MapReduce. Using a hash function, a key (or a subset of a key) creates partitions.

MapReduce splits each output of the combiner according to the key value. Then records with similar key values belong to the same section. After that, each section is sent to the reducer.

MapReduce splitting ensures that the map output is evenly distributed across the reducer.

Shuffle and Sort:

After splitting, the output is shuffled into collapsed nodes. Shuffling is the physical transfer of data over a network. This is because all converters terminate and shuffle their output at the reducer node. The framework then joins and sorts these intermediate results. This serves as an input to the phase reduction.

Reducer:

The reducer then takes as input the intermediate set of key-value pairs generated by the converter. It then runs the reducer function on each to produce an output. The gear output is the decisive output. The platform then saves the output to HDFS.

How many Reduces?

The right number of reduces seems to be 0.95 or 1.75 multiplied by (<no. of nodes> * <no. of maximum containers per node>).

With 0.95 all of the reduces can launch immediately and start transferring map outputs as the maps finish. With 1.75 the faster nodes will finish their first round of reduces and launch a second wave of reduces doing a much better job of load balancing.

Increasing the number of reduces increases the framework overhead, but increases load balancing and lowers the cost of failures.

The scaling factors above are slightly less than whole numbers to reserve a few reduce slots in the framework for speculative-tasks and failed tasks.

RecordWriter:

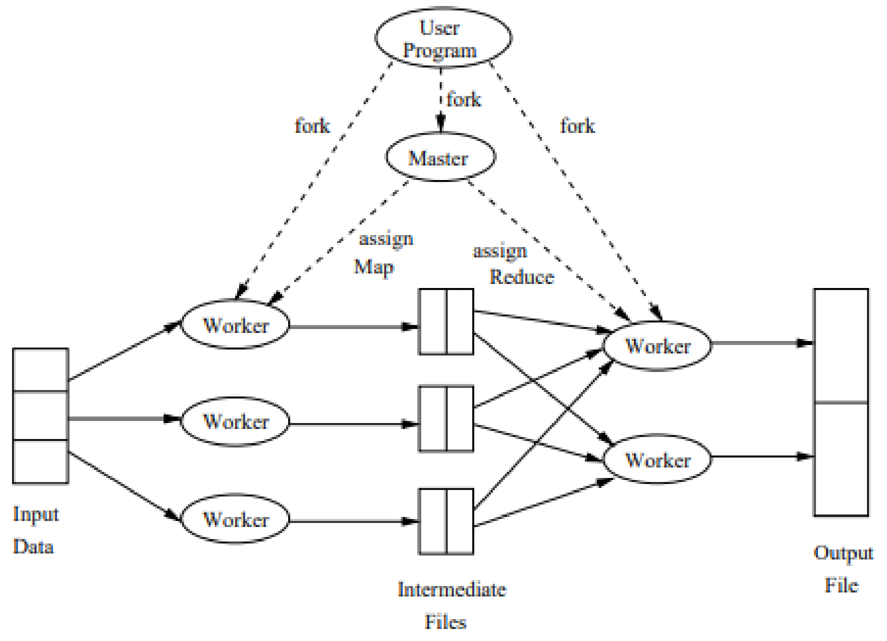
Writes this output key-value pair to the output file from the reducer stage.

OutputFormat:

OutputFormat specifies how the RecordReader writes these output key/value pairs to the output file. So, the instance served by Hadoop writes files to HDFS. So, the OutputFormat instance writes the reducer's deterministic output to HDFS.

Typically, a worker processes either a Map job (Map worker) or a Reduce job (Reduce worker), but not both jobs at the same time.

A master has many responsibilities. One is to create multiple Map jobs and Reduce jobs that the user program selects. These tasks are assigned to the workflow by the master. It makes sense to create one Map job for each chunk of the input file, but you may want to create fewer Reduce jobs. The reason for limiting the number of reduce jobs is that intermediate files must be created for each map job, and if there are too many reduce jobs, the number of intermediate files increases.



The Master keeps track of the state of each Map and Reduce job (idle, running in a specific worker process, or finished). The workflow reports to the master when the task is complete, and the master assigns a new task to that workflow.

Each map job is assigned one or more chunks of an input file and runs user-written code. The Map task creates a file for each Reduce task on the local disk of the worker running the Map task. The master receives information about the location and size of each of these files and the shrink operation of those files. When the master assigns a collapse action to a job flow, all the files that make up the input are passed to that job. The Reduce job runs user-written code and writes the results to a file that is part of the surrounding distributed file system.

4.2 ALGORITHM USING MAP REDUCE

MapReduce is a distributed data processing algorithm introduced by Google. The MapReduce algorithm is primarily based on a functional programming model. The MapReduce algorithm is useful for reliably and efficiently processing large amounts of data in parallel in a cluster environment.

Divides input tasks into smaller, more manageable subtasks and executes them in parallel. The MapReduce algorithm is based on sending processing nodes (local machines) to where the data resides.

The MapReduce algorithm works by splitting the process into three steps.

- Matching Stage
- Sorting and Moving Stage
- Reducing Stage

4.2.1 Matrix and Vector Multiplication by Map Reduce:

The original purpose for which the Google MapReduce implementation was created was to perform the very large matrix-vector multiplication required to compute the PageRank. It can be seen that matrix-vector and matrix-matrix computations are well suited for MapReduce-style computing. Another important class of operations where MapReduce can be effectively used is relational algebra operations.

Suppose we have an n × n matrix M, in which the entries in row i and column j are represented by m_{ij}. Suppose we have a vector v of length n whose jth element is equal to v_j. Then the matrix-vector product is a vector x of length n where the i-th element xi is

$$x_i = \sum_{j=1}^{n} m_{ij} u_j$$

If n = 100, we don't want to use DFS or MapReduce for this calculation.

However, this kind of calculation is the basis of the web page rankings performed by search engines, and n is in the tens of millions, so it can be used for any map operation.

Matrix M and vector v are stored in DFS files. Assume that the row column coordinates of each matrix element can be found either by their location in a file, or because they are stored in triples (i, j, m_{ij}) with explicit coordinates. We also assume that the position of the element v_j in the vector v will be found in a similar way.

Map Function: The Map function is written to be applied to a single element M. However, if v has not yet been read into the main memory of the compute node executing the Map operation, then v will first read the whole, then will be available to all applications of the "Map" function performed in this "Map" operation. Each map operation operates on a slice of matrix M. A key-value pair (i, $m_{ij}v_j$) is generated from each element of the m_{ij} matrix. Thus, all members of the sum that make up the xi component of the matrix-vector product receive the same key.

Reduce function: The reduce function simply sums up all values associated with a given key i. The result is a pair (i, x_i).

4.2.2 Computing Selection and Projection by Map Reduce:

Computing Selection by Map Reduce:

Selection does not require all the features of MapReduce. This can be done most conveniently only on the map part, but it can also be done only on the reduced part. Here is a MapReduce implementation of σC(R) allocation.

The Map function: check that C is satisfied for each tuple t in R. If so, create key-value pairs (t, t). That is, both the key and the value are t.

The Reduce Function: Reduce function is an identity. It simply passes of each key-value pair as output.

Computing Projection by Map Reduce:

Projection is performed similarly to selection, because projection may cause the same tuple to appear several times, the Reduce function must eliminate duplicates. We may compute πS (R) as follows.

The Map Function: For each tuple t in R, construct a tuple t′ by eliminating from t those components whose attributes are not in S. Output the key-value pair (t′, t′).

The Reduce Function: For each key t′ produced by any of the Map tasks, there will be one or more key-value pairs (t′, t′). The reduce function converts (t', [t', t',..., t']) to (t',t') so that for that key t' we get exactly one pair (t', t'). create.

Note that shrink operation is deduplication. Because these operations are associative and commutative, the combiner associated with each map operation can remove locally generated duplicates. But you still need a Reduce job to remove two identical tuples coming from different Map jobs.

4.2.3 Computing Grouping and Aggregation by Map Reduce:

As with joins, a minimal grouping and aggregation example with one grouping property and one aggregation is described. Let R(A, B, C) be a relation to which the operators γA, θ(B)(R) are applicable. Map does group and Reduce does aggregation.

The Map function: Creates a key-value pair (a, b) for each tuple (a, b, c).

The Reduce Function: Each key represents a group. Aggregate operator θ list [b1, b2,.,bn] value B associated with key a. Output is the (a, x) pair. where x is the result of applying θ to the list. For example, if θ is equal to SUM, then x = b1 + b2 + · · · + bn, and if θ is equal to MAX, then x is b1, b2, . . . ,billion.

If there are multiple grouping attributes, the key is a list of tuple values for all these attributes. If there is more than one aggregation, the reduction function applies each to the list of values associated with the given key and returns a tuple of keys with the component for all properties. Grouping (each aggregation result if more than one).

4.3 SELF-LEARNING TOPIC: CONCEPT OF SORTING AND NATURAL JOINS

Concept of Sorting:

Sorting is one of the basic MapReduce algorithms to process and analyze data. MapReduce implements sorting algorithm to automatically sort the output key-value pairs from the mapper by their keys.Sorting methods are

MODULE III

5

INTRODUCTION TO NOSQL

Unit Structure

5.0 OBJECTIVES

This chapter would make you understand the following concepts:

- What is NoSQL and how NoSQL is an important technology in the big data landscape.
- Features of NoSQL which makes it so popular.
- CAP Theorem and visual guide to NoSQL and CAP theorem.
- NoSQL Business drivers like volume, velocity, agility and variability
- Advantages of NoSQL
- Sharding and Replication along with its advantages
- NoSQL Data Architecture patterns – Key value store, Column Database, Graph Data store, Document-based store
- Use of NoSQL in Industry
- What is Big Data?
- Relation between NoSQL and Big Data
- Understanding various problems which are faced due to Big Data in different industries with the help of multiple use cases.

5.1 WHAT IS NOSQL?

We say that today is the age of Big Data. The sheer volume of data being generated today is exploding. The rate of data creation or generation is mind boggling. Mobile phones, social media, imaging technologies which are used for medical diagnosis, non-traditional IT devices like RFID readers, automated trackers, monitoring systems, GPS navigation systems —all these are among the fastest growing sources of data. Now keeping up with this huge influx of data is difficult, but what is more challenging is storing, analysing and archiving vast amounts of this generated data.

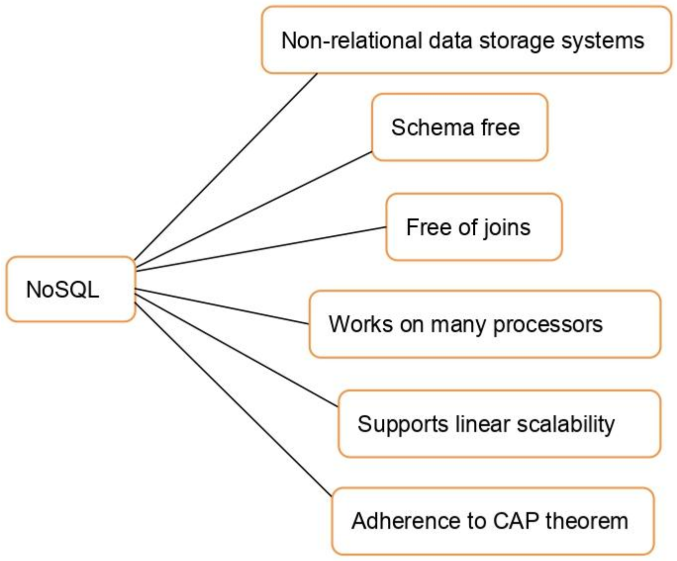
Along with the speedy data growth, data is also becoming increasingly semi-structured and sparse. It means the traditional data management technologies which were designed around upfront schema definition and relational references are no longer useful. This has led to the rise of a newer class of database technologies. The big data technology landscape majorly consists of two important technologies – NoSQL and Hadoop.

The term NoSQL was first used by Carlo Strozzi in 1998 to name his lightweight, open-source relational database that did not expose the standard SQL interface. NoSQL is actually an acronym that stands for ‘Not Only SQL’. Today NoSQL is used as an umbrella term for all the data stores and databases that do not follow traditional well established RDBMS principles. NoSQL is a set of concepts that allows the rapid and efficient processing of data sets with primary focus on performance, reliability and agility. NoSQL databases are widely used in big data and other real-time web applications. These are non-relational, open source and distributed databases. They are getting hugely popular because of their ability to scale out or scale horizontally and their proficiency at dealing with a rich variety of structured, semi-structured and unstructured data.

5.2 FEATURES OF NOSQL

Following are the various features of NoSQL which makes it a popular technology:

1. **Non-relational data storage system:** They do not adhere to the relational data model. NoSQL systems store and retrieve data from many formats like key-value pairs or document-oriented or column-oriented or graph-based databases.
2. **Schema free:** NoSQL databases are gaining popularity because of their support for flexibility to the schema. They easily allow data not to strictly adhere to any schema structure at the time of storage.
3. **Free of joins:** NoSQL databases allow users to extract data using simple interfaces without joins.



4. **Works on many processors:** NoSQL systems allow users to store the database on multiple processors and still maintain high-speed performance.

5. **Supports linear scalability:** Scalability is the ability of a system to increase throughput with the addition of resources to address increase in load. There is a consistent increase in the performance after adding multiple processors. NoSQL allows easy scaling to adapt to the data volume and complexity of cloud applications.

6. **Adherence to CAP theorem:** NoSQL databases do not offer support for ACID properties (Atomicity, Consistency, Isolation and Durability). On the contrary, they abide by Brewer's CAP theorem (Consistency, Availability and Partition tolerance) and are often seen compromising on consistency in favour of availability and partition tolerance.

5.3 CAP THEOREM

Eric Brewer first introduced the CAP theorem in year 2000 at a symposium. This theorem is widely adopted today by large web companies like Amazon as well as the NoSQL community. The acronym CAP stands for Consistency, Availability and Partition Tolerance.

- **Consistency:** Making available the same single updated readable version of the data to all the clients. Here, Consistency is concerned with multiple clients reading the same data from replicated partitions and getting consistent results.

- **High Availability:** System remains functional even if some nodes fail. High Availability means that the system is designed and implemented in such a way that it continues its read and write operation even if nodes in a cluster crash or some hardware or software parts are down due to upgrades. Internal communication failures between replicated data should not prevent updates.

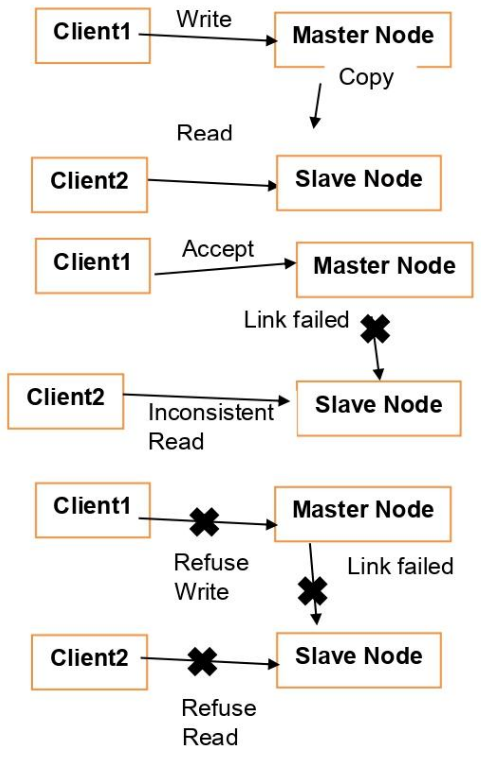
- **Partition tolerance:** Partition tolerance is the ability of the system to continue functioning in the presence of network partitions. This situation arises when network nodes cannot connect to each other (temporarily or permanently). System remains operations on system split or communication malfunction. A single node failure should not cause the entire system to collapse.

Distributed systems can only guarantee two of the features, not all three. Satisfying all three features at the same time is impossible. CAP theorem summarises that if you cannot limit the number of faults and requests can be directed to any server and you insist on serving every request then you cannot possibly remain consistent. It means you must always give up something – either consistency or availability or tolerance to failure and reconfiguration. So, CAP theorem can help the organization's decide what properties (Consistency, Availability or Partition Tolerance) are most important for their business.

If consistency and high availability are simultaneously required then a faster single processor can be the best choice. E.g., Amazon's Dynamo is available and partition tolerant but not strictly consistent whereas Google's Bigtable chooses consistency and availability over partition tolerance.

Choose consistency over availability whenever your business requirements demand atomic read and write operations. Choose availability over consistency when your business requirements allow some flexibility around when the data in the system is in sync.

The below given diagram helps you decide the relative merits of availability versus consistency when a network fails.

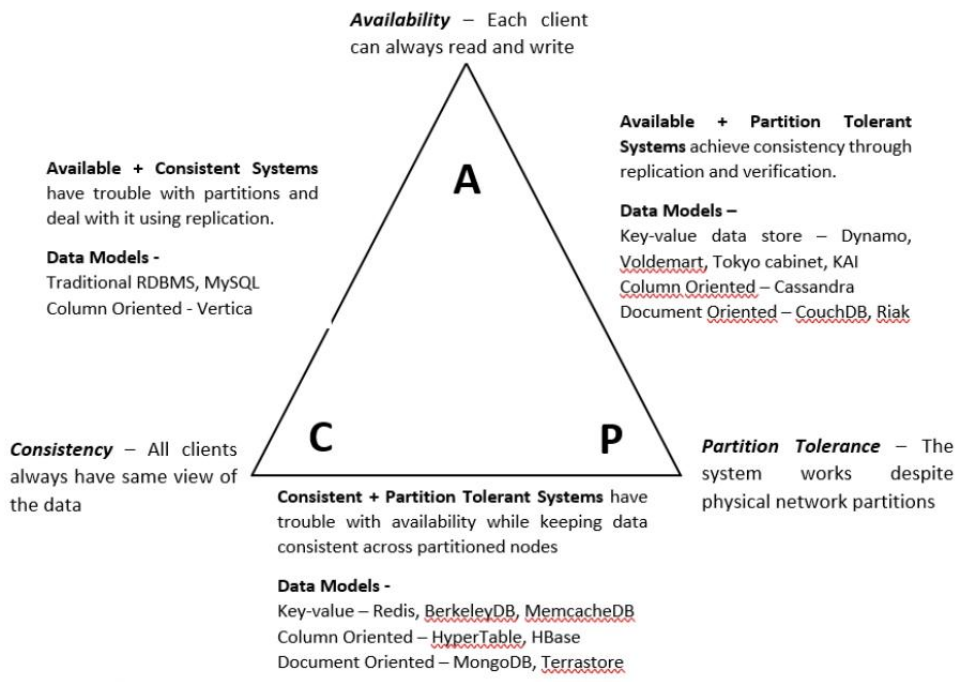


Normal Operation
Under normal operation a client write operation will be carried on the master node and then will be replicated over the network onto a slave node.

High Availability option
In case of link failure, you accept a write and risk inconsistent reads from the slave.

Consistency option
You choose consistency over availability and so block the client's write operation on the master node until the link between the data centres is restored back.

Following is the visual guide to NoSQL and CAP theorem:

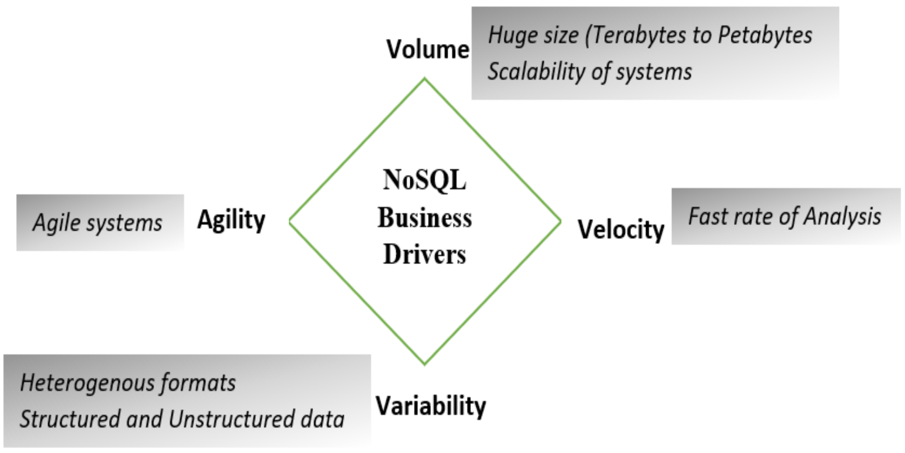

Availability – Each client can always read and write
A
C
P
Available + Consistent Systems have trouble with partitions and deal with it using replication.
Data Models -
Traditional RDBMS, MySQL
Column Oriented - Vertica
Available + Partition Tolerant Systems achieve consistency through replication and verification.
Data Models –
Key-value data store – Dynamo, Voldemart, Tokyo cabinet, KAI
Column Oriented – Cassandra
Document Oriented – CouchDB, Riak
Consistency – All clients always have same view of the data
Partition Tolerance – The system works despite physical network partitions
Consistent + Partition Tolerant Systems have trouble with availability while keeping data consistent across partitioned nodes
Data Models -
Key-value – Redis, BerkeleyDB, MemcacheDB
Column Oriented – HyperTable, HBase
Document Oriented – MongoDB, Terrastore

5.4 NOSQL BUSINESS DRIVERS

Today, we are living in an age of rampant data growth. As the size of data grows and source of data creation becomes increasingly diverse, we are faced with following challenges:

- Efficiently storing and accessing large amounts of diverse data is becoming difficult. Fault tolerance and backups make things even more complex.
- Manipulating large datasets involves running immensely parallel processes. Recovering the system gracefully from failures and providing results in a reasonably short period of time is complex.
- Managing the continuously evolving schema and metadata for semi-structured and unstructured data produced by diverse sources is another complex issue to deal with.

In today's world many organizations which supported single-CPU relational data models are rapidly making changes in their businesses based on the data they receive. The organizations realised that the way and means of storing and retrieving large amounts of data need newer approaches beyond their current techniques. NoSQL and related big data solutions were the first step in that direction. The demands of Volume, Velocity, Variability and Agility play a key role in developing cracks in the popularity of single-processor relational systems and emergence of NoSQL technologies.



- **Volume:** We have seen the growth in volume of data from bits to bytes to gigabytes to zettabytes. Traditional relational database infrastructure cannot cope up with these huge volumes of data. Just consider Facebook, since 2016, there are over 2 trillion posts and 270 billion photos uploaded. Enormous volume of data plays an important role for pushing the organizations to look for alternatives to their current RDBMS setup for querying big data. In earlier days, performance concerns were solved by using faster processors. But with the increase in volume of data produced daily, there was a need for horizontal scaling. So eventually the organizations moved to parallel processing from serial processing. In parallel processing data problems are split into separate paths and sent to separate processors to divide and conquer the load.

- **Velocity:** At what speed is new data generated? Rate at which this data is generated is much faster. We have moved from the days of batch processing to real time processing. Single processor RDBMS systems cannot meet the demands of real-time inputs and online queries made by users. When single-processor RDBMS is used as backend for processing online queries, the random bursts in online traffic slows down the response for every user. So, Velocity plays a major role in NOSQL movement.

- **Variability:** How diverse are different types of data? Data now is extremely diverse. Variety deals with wide range of data types and sources of data. So, rigid database schema imposed by RDBMS cannot deal with such heterogenous type of data.

- **Agility:** Traditional databases require a schema before writing data. Moreover, time is needed to get the data into the database and this process cannot be considered as agile. Today, the technologies focus more on agility meaning how fast can you extract value from mountains of data and how quickly can you translate that information into action.

5.5 ADVANTAGES OF NOSQL

Advantages of NoSQL are as follows:

1. **Scalability:** NoSQL technology was designed in Internet and Cloud computing era. So, it implements scale-out architecture. Here scalability is achieved by spreading the data and the load to a large cluster of computers. New computers and storage space can be easily added to the cluster. Whenever data volume increases or traffic grows scalability can be easily achieved in NoSQL. It allows distribution of database across 100+ nodes in multiple data centres. It can sustain over 100,000+ database read and write operations per second. It allows storage of 10,000 lakh documents in the database. In addition, many NoSQL databases can be easily upgraded or changed with zero downtime.

2. **No predefined schema:** Relational databases have predefined schema. To use RDBMS, a data model must be designed and data has to be transformed and then loaded into the database. NoSQL databases are gaining popularity as they allow the data to be stored in ways that are easier to understand. Fewer transformations are required when the data is stored and later retrieved for usage. Moreover, varied forms of data – structured, semi-structured and unstructured can be easily stored and retrieved. As NoSQL databases are flexible, they can easily adapt to newer forms of data. This also helps the developers who find it easier to develop various types of applications using data in native formats.

3. **Economical to use:** NoSQL databases are relatively cheaper to install and manage as compared to traditional datastores. It provides all the benefits of scale, high availability, fault tolerance at lower operational costs.

4. **Distributed and fault tolerant architecture:** In NoSQL, data can be replicated to multiple nodes and can be partitioned. Sharding allows different pieces of data to be distributed across multiple servers. NoSQL databases support auto-sharding. It means that they can natively and automatically distribute data across multiple servers without requiring the application to be even aware of the composition of the server pool. Servers can be added or removed without any application downtime. It means that data and query load are automatically balanced across servers in case a server becomes non-functional. No application disruption is experienced as non-functional server is quickly and transparently replaced. NoSQL also allows replication of data. Here, multiple copies of data are stored across the cluster and data centres. This ensures high availability and fault tolerance.

5.6 SHARDING AND REPLICATION

As data generated and stored withing an organization increases, there may come a point where the current infrastructure becomes inadequate and some mechanism to break/divide the data into reasonable chunks is needed. Organizations may use auto-sharding which includes breaking the database into chunks called as shards and spreading the shards across a number of distributed servers. Each shard is an independent database and collectively they constitute a logical database. Data shards may also be replicated for reasons of reliability and load balancing. On older systems this might mean taking the system down for few hours while the database is reconfigured manually and data is copied from old system onto the new system. But NoSQL systems do all this automatically (e.g., MongoDB). Many NoSQL databases provide automatic partitioning and balancing of data among nodes. Sharding is highly automated process in both big data and fault tolerant systems.

The most two important advantages of Sharding are as follows:

- Sharding reduces the amount of data that each node needs to store and manage. E.g., if the dataset was 2TB in size and we distribute this over 4 shards then each shard will have to store only 512 GB data. As the cluster size increases, the amount of data that each shard has to store and manage will decrease.

- Sharding also reduces the number of operations that each node has to handle. E.g., if we have to perform write operation, then the application has to access only that shard which stores the data related to the write operation.

As the number of servers grows, the probability of any one server being down remains the same. So, you may decide to replicate the data to a backup or mirrored system. Replication Factor means the number of copies of the given dataset stored across the network. In this case, when there are any changes to the master copy of the data, you must also update the backup copies immediately. It means that the backup copies must remain in sync with the master copy. So, you must have a method of data replication. Syncing these databases can reduce the system performance.

5.7 NOSQL DATA ARCHITECTURE PATTERNS

Types of NoSQL databases/data stores:

NoSQL applications use a variety of databases. Each NoSQL type of data store has unique attributes and uses.

1. **Key-value store:** It maintains a big hash table of keys and values. It is a simple data storage system that uses a key to access a value. The key of a key/value pair is a unique value in the set and can be easily looked up to access the data. Within a key-value pair, there are two related data elements. The first is a constant used to define the dataset. The other is a value which is a variable belonging to the dataset.

For e.g., a mobile phone can be the key, while its colour, model, brand can be the values.

Another example can be of the bank. A bank can use a database to store key-value pairs that contain their customers information. Customer’s name can be the key while his/her account number, account type, branch, balance can be the values.

Key/value pairs can be distributed and held in a cluster of nodes. Its typical usage includes Image stores, session management for web applications, user sessions for massive multi-player online games, online shopping carts etc.

A powerful key/value store is Oracle’s Berkeley DB which is a pure storage engine where both key and value are an array of bytes.

Another type of key/value store in regular use is a cache. A cache provides an in-memory snapshot of the most frequently used data in an application. A popular open source, high performance object caching system used in web applications is Memcached.

Another popular open source in-memory NoSQL key/value store is Redis (REmote DIctionary Server) which is primarily used as an application cache or quick-response database. Microsoft uses Redis to power MSN portal that gets 2 billion hits with a latency of less than 10ms on peak days.

Amazon’s Dynamo is also a key/value pair which emphasize availability over consistency in distributed deployments. It forms the backbone for Amazon’s distributed fault-tolerant and highly available system. Cassandra (used by Facebook, Reddit, Twitter, Spotify, eBay), Riak and Voldemart (used by LinkedIn) provide open-source Amazon Dynamo capabilities.

2. **Document Databases:** It maintains data in collections constituted of documents. Here hierarchical data structures are directly stored in the database. Document databases treat a document as a whole and do not split a document into its constituent name/value pairs. The document is stored in JSON or XML formats.

For example: Sample document in Document database

```
{ “Name” : “ABC”,
  “Address” : “Indore”,
  “Email” : abc@xyz.com,
  “Contact” : “98765”
}
```

Document databases are simple and scalable, thus making them useful for mobile applications that require fast iterations. These data models are also used in video streaming platforms, blogs and similar services.

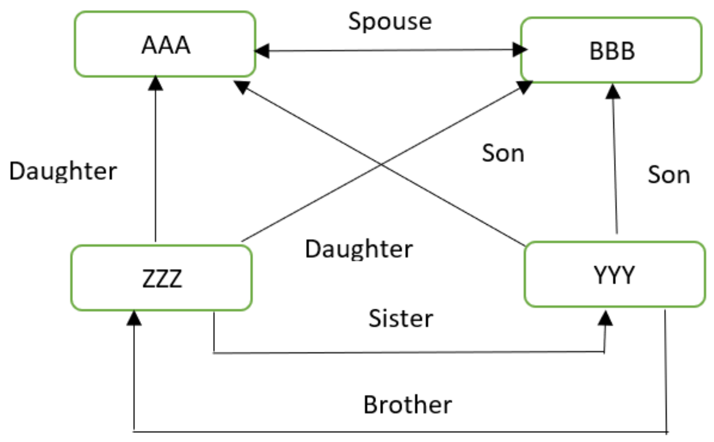
Most popular open-source document databases are MongoDB (used by GitHub), Amazon DocumentDB and Apache CouchDB (used by Apple, LinkedIn).

3. **Graph database:** It is also called as Network database. These datastores are based on graph theory. A graph database can have ACID properties. A graph stores data in nodes. It is designed to identify and work with the connections between data points. The fundamental data model of a graph database is very simple – nodes connected by edges. The entity is stored as a node with the relationship as edges. An edge gives a relationship between nodes. Every node and edge have a unique identifier.

The simplest possible graph is a single node. This would be a record with named values called properties/attributes. Theoretically there is no upper limit on the number of properties in a node, but practically we distribute the data into multiple nodes, organized with explicit relationships. When we store a graph structure in RDBMS, we can actually store single type of relation and adding another relation type requires changing the schema structure. A graph database is schema free and we can scale up to any level by adding a different type of entities and relations (nodes and edges).

It is typically used in mining data about customers on social networks, fraud detection, product preferences, eligibility rules etc.

Two popular Graph databases are Neo4j (ACID-compliant graph database used by Walmart to improve the speed and effectiveness of their online recommendation platform.) and FlockDB (used in Twitter – stores the adjacency lists for followers on Twitter). FlockDB is simply nodes and edges with no mechanism for additional attributes whereas Neo4j allows you to attach Java objects as attributes to nodes and edges in a schema-free way.



4. **Column store:** It's a sparse matrix system that uses a row and a column as keys. Here each storage block has data from only one column. It avoids consuming space when storing nulls by simply not storing a column when a value does not exist for a column. A column family is a collection of rows, which contains any number of columns

for each row. The row must be unique within a column family but the same row can be used in another column family.

CustomerID	Column Family : Identity
100	First Name : AAA Last Name : ZZZ
101	First Name : BBB Middle Name : KKK Last Name : YYY
102	Title : Dr. First Name : CCC
CustomerID	Column Family : Contact Info
100	Mobile : 555555 Email : a@eg.com
102	Email : b@asia.com
103	Mobile : 12345

The benefit of storing data in the column datastore is fast searching and accessing of the data. These type of datastores are typically used in data warehousing as they handle analytical queries very efficiently.

HBase (used by Facebook, Yahoo!) is a popular open-source, sorted ordered column family store. Data stored in HBase can be manipulated using MapReduce infrastructure. MapReduce tools can easily use HBase as the source and/or sink of the data.

Following table summarises comparison among different NoSQL databases and RDBMS based on various important features:

	Performance	**Scalability**	**Flexibility**	**Complexity**	**Functionality**
Key-Value Data Store	High	High	High	None	Variable
Column Oriented Data Store	High	High	Moderate	Low	Minimal
Document Data Store	High	Variable	High	Low	Variable
Graph Data Store	Variable	Variable	High	High	Graph Theory
Relational Databases	Variable	Variable	Low	Moderate	Relational Algebra

Following table summarises popular NoSQL products and its usage:

	Key-Value Data store	**Column Oriented Data store**	**Document Data Store**	**Graph Data Store**
Popular Products	Oracle's Berkley DB, Redis, Riak, Amazon's Dynamo	HBase, HyperTable	MongoDB, Amazon DocumentDB, Apache CouchDB	Neo4j, FlockDB, InfiniteGrpah
Usage	Shopping carts, web user data analysis	Analyze huge web user actions, sensor feeds	Real-time analytics, logging, document archive management	Network modelling, recommendation system
Mostly used by	Amazon, LinkedIn	Facebook, Twitter, eBay, Netflix	GitHub, Apple	Wallmart

5.8 USE OF NOSQL IN INDUSTRY

NoSQL is used in variety of applications in industries. Some uses are listed below:

1. **Session Management:** NoSQL is used for storing web application session information which is very large in size. Such session data is of unstructured format so it is advisable to store it in a schema-less document.

2. **Storing User Profile:** It is necessary to store user's profile to enable online transactions, to understand user preferences and carry out user authentication. In recent years users of web and mobile applications have grown two-fold so to handle such deluge NoSQL can be easily used to increase the capacity by adding servers which makes scaling cost effective.

3. **Content and Metadata store:** Many organizations require storage for storing huge amount of digital content, articles, e-books etc. in order to merge a variety of learning tools in a single platform. The content-based applications require frequent access to metadata so it should have minimum response time. NoSQL provides flexibility to store different types of contents and also provides faster access to data.

4. **Mobile Applications:** Users of smartphones are increasing at an enormous pace. Mobile applications face issues related to growth and volume. With relational databases, it becomes really difficult to expand as number of users increase. NoSQL allows us to expand without worries. Moreover, as NoSQL allows data to be stored in

MODULE IV

7

HADOOP ECOSYSTEM: HIVE AND PIG

Unit Structure

7.0 OBJECTIVES

- Complete knowledge of Apache Hive.
- Able to learn work the Hive project independently.
- Understand Hive Architecture.
- Learn Hive to explore and analyse huge datasets.
- Learn HQL language.
- Able to learn different operations of Hive.

7.1 INTRODUCTION OF HIVE

The term 'Big Data' refers to massive datasets with a high volume, high velocity, and a diverse mix of data that is growing by the day. Processing

Big Data using typical data management solutions is tough. As a result, the Apache Software Foundation developed Hadoop, a framework for managing and processing large amounts of data. Hive is a Hadoop data warehouse infrastructure solution that allows you to process structured data. It is built on top of Hadoop to summarise Big Data and facilitate querying and analysis. Initially developed by Facebook, Hive was eventually taken up by the Apache Software Foundation and developed as an open-source project under the name Apache Hive. It is utilised by a variety of businesses.

7.2 ARCHITECTURE

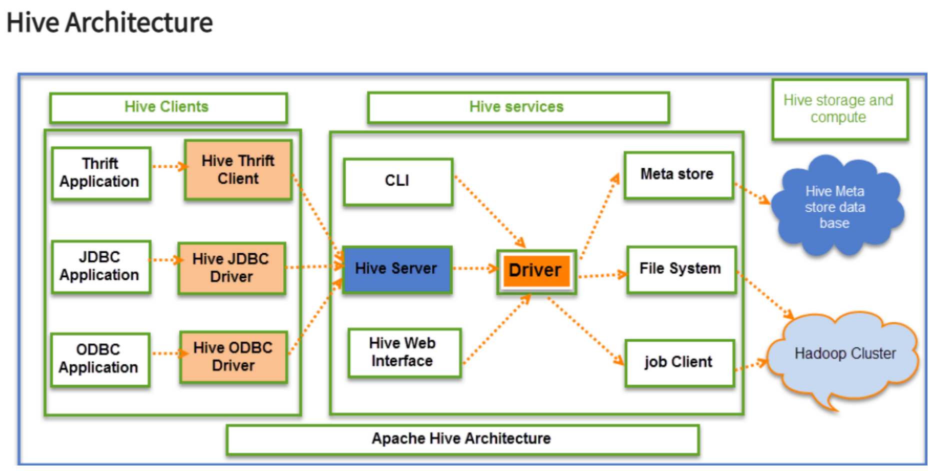


Figure 1: Architecture of HIVE

Important components of HIVE:

- HIVE clients
- HIVE Services
- Storage and computing

HIVE Clients:

Hive offers a variety of drivers for interacting with various types of applications. It will provide a Thrift client for communication in Thrift-based applications. JDBC Drivers are available for Java-based applications. ODBC drivers are available for any sort of application. In the Hive services, these Clients and Drivers communicate with the Hive server.

HIVE Services:

Hive Services can be used by clients to interact with Hive. If a client wishes to do any query-related actions in Hive, it must use Hive Services to do so. The command line interface (CLI) is used by Hive to perform DDL (Data Definition Language) operations. As indicated in the architectural diagram above, all drivers communicate with Hive server and the main driver in Hive services. The main driver is present in Hive services, and it interfaces with all types of JDBC, ODBC, and other client-specific applications. Driver will process requests from various apps and send them to the meta store and field systems for processing.

Storage and Computing:

Hive services like Meta store, File system, and Job Client communicate with Hive storage and carry out the following tasks.

- The Hive "Meta storage database" stores the metadata of tables generated in Hive.
- Query results and data loaded into tables will be stored on HDFS in a Hadoop cluster.

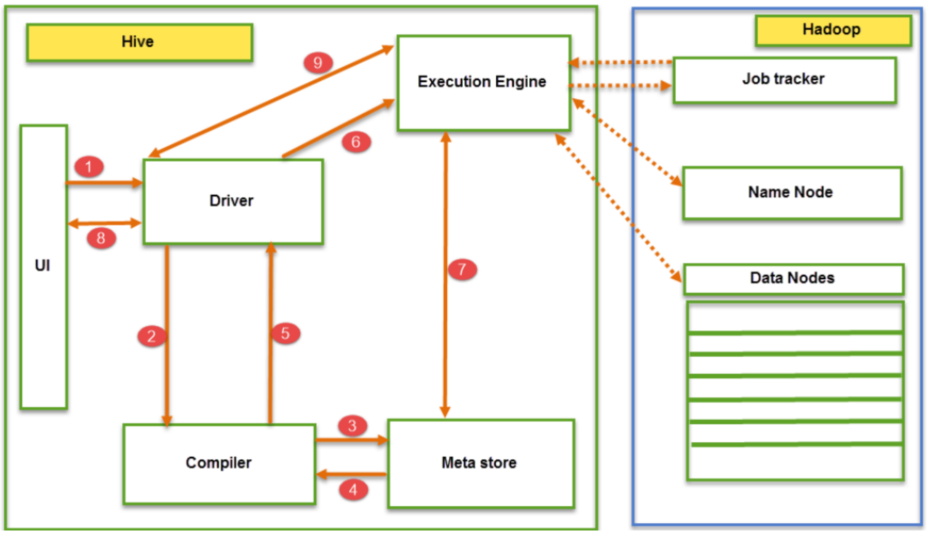


Figure 2: Job Execution Flow

From the above screenshot we can understand the Job execution flow in Hive with Hadoop the data flow in Hive behaves in the following pattern;

1. Executing Query from the UI (User Interface)
2. The driver is interacting with Compiler for getting the plan. (Here plan refers to query execution) process and its related metadata information gathering
3. The compiler creates the plan for a job to be executed. Compiler communicating with Meta store for getting metadata request
4. Meta store sends metadata information back to compiler
5. Compiler communicating with Driver with the proposed plan to execute the query
6. Driver Sending execution plans to Execution engine
7. Execution Engine (EE) acts as a bridge between Hive and Hadoop to process the query. For DFS operations.

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

8. Fetching results from driver
9. Sending results to Execution engine.

Once the results fetched from data nodes to the EE, it will send results back to driver and to UI (front end). Hive Continuously in contact with Hadoop file system and its daemons via Execution engine. The dotted arrow in the Job flow diagram shows the Execution engine communication with Hadoop daemons.

7.3 WAREHOUSE DIRECTORY AND META-DATA

When you create a table in Hive, by default Hive will manage the data, which means that Hive moves the data into its warehouse directory. Alternatively, you may create an external table, which tells Hive to refer to the data that is at an existing location outside the warehouse directory. The Hive component of the Hadoop eco-system processes all of the structure information of the many tables and partitions in the warehouse, including column, column type information, and the accompanying HDFS files where data is stored. That's where all of Hive's metadata is stored. Metadata which meta-store stores contain things like:

1. IDs of database
2. IDs of Tables and index
3. Time of creation of the index
4. Time of creation of tables
5. Input format used for tables
6. Output format used for tables

Apache Hive metadata is stored in Meta-store, which is a central repository. It uses a relational database to hold metadata for Hive tables (such as schema and location) and partitions. It uses the meta-store service API to give clients access to this data.

7.4 HIVE QUERY LANGUAG

The Hive Query Language (HiveQL) is a query language for processing and analysing structured data in Apache Hive. It shields users from Map Reduce programming's complexities. To make learning easier, it borrows notions from relational databases, such as tables, rows, columns, and schema. Hive provides a CLI for Hive query writing using Hive Query Language (HiveQL). HiveQL is Hive’s query language, a dialect of SQL. It is heavily influenced by MySQL, so if you are familiar with MySQL, you should feel at home using Hive. Hive’s SQL dialect, called HiveQL, is a mixture of SQL-92, MySQL, and Oracle’s SQL dialect. The level of SQL-92 support has improved over time, and will likely continue to get better. HiveQL also provides features from later SQL standards, such as window functions (also known as analytic functions) from SQL:2003. Some of Hive’s non-standard extensions to SQL were inspired by MapReduce, such as multitable inserts (see Multitable insert) and the TRANSFORM, MAP, and REDUCE clauses. When starting Hive for the first time, we can check that it is working by listing its tables — there should be none. The command must be terminated with a semicolon to tell Hive to execute it:

```
hive> SHOW TABLES;
```

```
OK Time taken: 0.473 seconds
```

Like SQL, HiveQL is generally case insensitive (except for string comparisons), so show tables; works equally well here. The Tab key will autocomplete Hive keywords and functions. For a fresh install, the command takes a few seconds to run as it lazily creates the meta-store database on your machine.

7.4.1 Loading Data Into Hive:

We can load data into hive table in three ways. Two of them are DML operations of Hive. Third way is using HDFS command. If we have data in RDBMS system like Oracle, Mysql, DB2 or SQL Server we can import it using SQOOP tool. Now to implement loading of data into hive we will practise some commands. Create a table called employee which has data as following:

Employee (eno, ename, salary, dno)

11, Balaji,100200,15

12, Radhika,120000,25

13, Nityanand,150000,15

14, Sai Nirupam,120000,35

Using Insert command:

We can load data into a table using Insert command in two ways. One Using Values command and other is using queries. Using Values command, we can append more rows of data into existing table. For example, to existing above employee table we can add extra row 15, Bala,150000,35 like below:

Insert into table employee values (15,'Bala',150000,35)

After this You can run a select command to see newly added row.

By using Queries:

You can also save the results of a query in a table. As an example, assuming you have an emp table, you can upload data into the employee database as seen below.

Insert into table employee Select * from emp where dno=45;

By using Load:

You can load data into a hive table using Load statement in two ways. One is from local file system to hive table and other is from HDFS to Hive table.

By using HDFS:

If you have data in a local file, you can easily upload it with HDFS commands. To find the location of a table, use the describe command, as shown below.

describe formatted employee;

It will display Location of the table, Assume You got location as /data/employee, you can upload data into table by using one of below commands.

hadoop fs -put /path/to/localfile /Data/employee

hadoop fs -copyFromLocal /path/to/localfile /Data/employee

hadoop fs -moveFromLocal /path/to/localfile /Data/employee

7.4.2 HIVE BUILT-IN FUNCTIONS

Return Type	Signature	Description
BIGINT	round(double a)	It returns the rounded BIGINT value of the double.

BIGINT	floor(double a)	It returns the maximum BIGINT value that is equal or less than the double.
BIGINT	ceil(double a)	It returns the minimum BIGINT value that is equal or greater than the double.
Double	rand(), rand(int seed)	It returns a random number that changes from row to row.
String	concat(string A, string B,...)	It returns the string resulting from concatenating B after A.
String	substr(string A, int start)	It returns the substring of A starting from start position till the end of string A.
String	substr(string A, int start, int length)	It returns the substring of A starting from start position with the given length.
String	upper(string A)	It returns the string resulting from converting all characters of A to upper case.
string	ucase(string A)	Same as above.
string	lower(string A)	It returns the string resulting from converting all characters of B to lower case.
string	lcase(string A)	Same as above.
string	trim(string A)	It returns the string resulting from trimming spaces from both ends of A.
string	ltrim(string A)	It returns the string resulting from trimming spaces from the beginning (left hand side) of A.
string	rtrim(string A)	rtrim(string A) It returns the string resulting from trimming spaces from the end (right hand side) of A.
string	regexp_replace(string A, string B, string C)	It returns the string resulting from replacing all substrings in B that match the Java regular expression syntax with C.
Int	size(Map<K.V>)	It returns the number of elements in the map type.

Int	size(Array<T>)	It returns the number of elements in the array type.
value of <type>	cast(<expr> as <type>)	It converts the results of the expression expr to <type> e.g. cast('1' as BIGINT) converts the string '1' to it integral representation. A NULL is returned if the conversion does not succeed.
string	from_unixtime(int unixtime)	convert the number of seconds from Unix epoch (1970-01-01 00:00:00 UTC) to a string representing the timestamp of that moment in the current system time zone in the format of "1970-01-01 00:00:00"
string	to_date(string timestamp)	It returns the date part of a timestamp string: to_date("1970-01-01 00:00:00") = "1970-01-01"
Int	year(string date)	It returns the year part of a date or a timestamp string: year("1970-01-01 00:00:00") = 1970, year("1970-01-01") = 1970
Int	month(string date)	It returns the month part of a date or a timestamp string: month("1970-11-01 00:00:00") = 11, month("1970-11-01") = 11
Int	day(string date)	It returns the day part of a date or a timestamp string: day("1970-11-01 00:00:00") = 1, day("1970-11-01") = 1
string	get_json_object(string json_string, string path)	It extracts json object from a json string based on json path specified, and returns json string of the extracted json object. It returns NULL if the input json string is invalid.

Some Examples of HIVE Built in functions:

Round() function:

```
hive> SELECT round(2.6) from temp;
```

On successful execution of query, you get to see the following response:

```
3.0
```

Floor() function:

hive> SELECT floor(2.6) from temp;

On successful execution of the query, you get to see the following response:

2.0

Ceil() function:

hive> SELECT ceil(2.6) from temp;

On successful execution of the query, you get to see the following response:

3.0

Aggregate Functions:

The following built-in aggregate functions are supported by Hive. These functions are used in the same way as SQL aggregate functions.

Return Type	Signature	Description
BIGINT	count(*), count(expr),	count(*) - Returns the total number of retrieved rows.
DOUBLE	sum(col), sum(DISTINCT col)	It returns the sum of the elements in the group or the sum of the distinct values of the column in the group.
DOUBLE	avg(col), avg(DISTINCT col)	It returns the average of the elements in the group or the average of the distinct values of the column in the group.
DOUBLE	min(col)	It returns the minimum value of the column in the group.
DOUBLE	max(col)	It returns the maximum value of the column in the group.

7.4.3 Joins in Hive:

- In Joins, only Equality joins are allowed.
- However, in the same query more than two tables can be joined.
- Basically, to offer more control over ON Clause for which there is no match LEFT, RIGHT, FULL OUTER joins exist in order.
- Also, note that Hive Joins are not Commutative
- Whether they are LEFT or RIGHT joins in Hive, even then Joins are left-associative.

SYNTAX:

Following is a syntax of Hive Join – HiveQL Select Clause.

```
join_table:
 table_reference JOIN table_factor [join_condition]
  | table_reference {LEFT|RIGHT|FULL} [OUTER] JOIN table_reference
  join_condition
  | table_reference LEFT SEMI JOIN table_reference join_condition
  | table_reference CROSS JOIN table_reference [join_condition]
```

Types of Joins:

- Inner join in Hive
- Left Outer Join in Hive
- Right Outer Join in Hive
- Full Outer Join in Hive

Inner Join:

Basically, to combine and retrieve the records from multiple tables we use Hive Join clause. Moreover, in SQL JOIN is as same as OUTER JOIN. Moreover, by using the primary keys and foreign keys of the tables JOIN condition is to be raised. Furthermore, the below query executes JOIN the CUSTOMER and ORDER tables. Then further retrieves the records:

```
hive> SELECT c.ID, c.NAME, c.AGE, o.AMOUNT

FROM CUSTOMERS c JOIN ORDERS o

ON (c.ID = o.CUSTOMER_ID);
```

Left Outer Join:

On defining HiveQL Left Outer Join, even if there are no matches in the right table it returns all the rows from the left table. To be more specific, even if the ON clause matches 0 (zero) records in the right table, then also this Hive JOIN still returns a row in the result. Although, it returns with NULL in each column from the right table. In addition, it returns all the values from the left table. Also, the matched values from the right table, or NULL in case of no matching JOIN predicate. However, the below query shows LEFT OUTER JOIN between CUSTOMER as well as ORDER tables:

```
hive> SELECT c.ID, c.NAME, o.AMOUNT, o.DATE

FROM CUSTOMERS c

LEFT OUTER JOIN ORDERS o

ON (c.ID = o.CUSTOMER_ID);
```

Right Outer Join:

Basically, even if there are no matches in the left table, HiveQL Right Outer Join returns all the rows from the right table. To be more specific, even if the ON clause matches 0 (zero) records in the left table, then also this Hive JOIN still returns a row in the result. Although, it returns with NULL in each column from the left table. In addition, it returns all the values from the right table. Also, the matched values from the left table or NULL in case of no matching join predicate.

notranslate"> hive> SELECT c.ID, c.NAME, o.AMOUNT, o.DATE FROM CUSTOMERS c RIGHT OUTER JOIN ORDERS o ON (c.ID = o.CUSTOMER_ID);

Full Outer Join:

The primary goal of this HiveQL Full outer Join is to merge the records from both the left and right outside tables in order to satisfy the Hive JOIN requirement. In addition, this connected table either contains all of the records from both tables or fills in NULL values for any missing matches on either side.

```
hive> SELECT c.ID, c.NAME, o.AMOUNT, o.DATE

FROM CUSTOMERS c

FULL OUTER JOIN ORDERS o

ON (c.ID = o.CUSTOMER_ID);
```

7.4.4 Partitioning in Hive:

Table partitioning is the process of splitting table data into sections based on the values of specific columns, such as date or country, and then separating the input records into distinct files/directories based on the date or country.

Partitioning can be done on the basis of many columns, resulting in a multi-dimensional structure on directory storage. For example, in addition to separating log entries by date column, we can also partition single-day records into country-specific separate files by inserting the country column in the partitioning. In the examples, we'll see more about this. Partitions are defined using the PARTITIONED BY clause and a set of column definitions for partitioning at the time of table formation. Partitions are defined using the PARTITIONED BY clause and a set of column definitions for partitioning at the time of table formation.

Advantages:

- Partitioning is used for distributing execution load horizontally.
- As the data is stored as slices/parts, query response time is faster to process the small part of the data instead of looking for a search in the entire data set.

- For example, in a large user table where the table is partitioned by country, then selecting users of country ‘IN’ will just scan one directory ‘country=IN’ instead of all the directories.

Limitations:

Having too many partitions in table creates large number of files and directories in HDFS, which is an overhead to NameNode since it must keep all metadata for the file system in memory only. Partitions may optimize some queries based on WHERE clauses, but may be less responsive for other important queries on grouping clauses. In Map reduce processing, huge number of partitions will lead to huge no of tasks (which will run in separate JVM) in each map reduce job, thus creates lot of overhead in maintaining JVM start up and tear down. For small files, a separate task will be used for each file. In worst scenarios, the overhead of JVM start up and tear down can exceed the actual processing time.

7.4.5 Querying Data:

Sorting and Aggregating:

Sorting data in Hive can be achieved by using a standard ORDER BY clause. ORDER BY performs a parallel total sort of the input (like that described in Total Sort). When a globally sorted result is not required — and in many cases it isn’t you can use Hive’s nonstandard extension, SORT BY, instead. SORT BY produces a sorted file per reducer. In some cases, you want to control which reducer a particular row goes to — typically so you can perform some subsequent aggregation. This is what Hive’s DISTRIBUTE BY clause does. Here’s an example to sort the weather dataset by year and temperature, in such a way as to ensure that all the rows for a given year end up in the same reducer partition:

```
hive> FROM records2

 > SELECT year, temperature

> DISTRIBUTE BY year

 > SORT BY year ASC, temperature DESC;

1949 111

1949 78

1950 22

1950 0

1950 -11
```

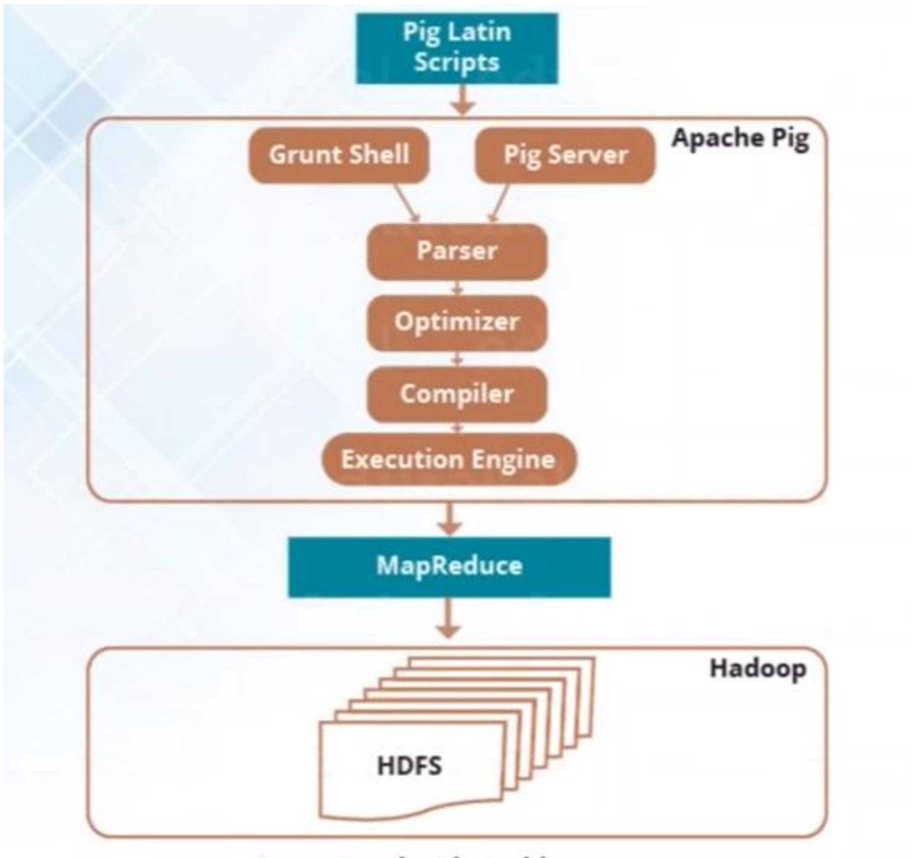
A follow-on query (or a query that nests this query as a subquery; see Subqueries) would be able to use the fact that each year’s temperatures were grouped and sorted (in descending order) in the same file. If the columns for SORT BY and DISTRIBUTE BY are the same, you can use CLUSTER BY as a shorthand for specifying both.

7.5 PIG

7.5.1 Background:

Apache Pig raises the level of abstraction for processing large datasets. MapReduce allows you, as the programmer, to specify a map function followed by a reduce function, but working out how to fit your data processing into this pattern, which often requires multiple MapReduce stages, can be a challenge. With Pig, the data structures are much richer, typically being multivalued and nested, and the transformations you can apply to the data are much more powerful. They include joins, for example, which are not for the faint of heart in MapReduce. Pig is made up of two pieces: The language used to express data flows, called Pig Latin. The execution environment to run Pig Latin programs. There are currently two environments: local execution in a single JVM and distributed execution on a Hadoop cluster. A Pig Latin program is made up of a series of operations, or transformations, that are applied to the input data to produce output. Taken as a whole, the operations describe a data flow, which the Pig execution environment translates into an executable representation and then runs. Under the covers, Pig turns the transformations into a series of MapReduce jobs, but as a programmer you are mostly unaware of this, which allows you to focus on the data rather than the nature of the execution. Pig is a scripting language for exploring large datasets. One criticism of MapReduce is that the development cycle is very long. Writing the mappers and reducers, compiling and packaging the code, submitting the job(s), and retrieving the results is a time-consuming business, and even with Streaming, which removes the compile and package step, the experience is still involved. Pig's sweet spot is its ability to process terabytes of data in response to a half-dozen lines of Pig Latin issued from the console. Indeed, it was created at Yahoo! to make it easier for researchers and engineers to mine the huge datasets there. Pig is very supportive of a programmer writing a query, since it provides several commands for introspecting the data structures in your program as it is written. Even more useful, it can perform a sample run on a representative subset of your input data, so you can see whether there are errors in the processing before unleashing it on the full dataset. Pig was designed to be extensible. Virtually all parts of the processing path are customizable: loading, storing, filtering, grouping, and joining can all be altered by userdefined functions (UDFs). These functions operate on Pig's nested data model, so they can integrate very deeply with Pig's operators. As another benefit, UDFs tend to be more reusable than the libraries developed for writing MapReduce programs.

7.5.2 Pig Architecture:



Apache Pig Architecture

1. **Parser:** The parser handles any pig scripts or instructions in the grunt shell. Parse will run checks on the scripts, such as checking the syntax, type checking, and a variety of other things. These checks will produce results in the form of a Directed Acyclic Graph (DAG), which contains pig Latin sentences and logical operators. Our logical operators of the scripts are nodes, and data flows are edges, therefore the DAG will have nodes that are connected to distinct edges.

2. **Optimizer:** After parsing and DAG generation, the DAG is sent to the logical optimizer, which performs logical optimizations such as projection and pushdown. By removing extraneous columns or data and pruning the loader to only load the relevant column, projection and pushdown increase query performance.

3. **Compiler:** The compiler compiles the optimised logical plan provided above and generates a series of Map-Reduce tasks. Essentially, the compiler will transform pig jobs into MapReduce jobs and exploit optimization opportunities in scripts, allowing the programmer to avoid manually tuning the software. Pig's compiler can reorder the execution sequence to enhance performance if the execution plan remains the same as the original programme because it is a data-flow language.

4. **Execution Engine:** Finally, all of the MapReduce jobs generated by the compiler are sorted and delivered to Hadoop. Finally, Hadoop executes the MapReduce job to produce the desired output.

5. Pig has two execution modes, which are determined by the location of the script and the availability of data.

- **Local Mode:** For limited data sets, local mode is the best option. Pig is implemented on a single JVM because all files are installed and run-on localhost, preventing parallel mapper execution. Pig will also peek into the local file system while importing data.

- **MapReduce Mode (MR Mode):** In MapReduce, the mode programmer needs access and setup of the Hadoop cluster and HDFS installation. In this mode data on which processing is done is exists in the HDFS system. After execution of pig script in MR mode, pig Latin statement is converted into Map Reduce jobs in the back-end to perform the operations on the data

7.5.3 Latin Basics:

Pig Latin is the language used to analyse data in Hadoop using Apache Pig. In this chapter, we are going to discuss the basics of Pig Latin such as Pig Latin statements, data types, general and relational operators, and Pig Latin UDF's.

Pig Statements:

While processing data using Pig Latin, statements are the basic constructs.

- These statements work with relations. They include expressions and schemas.

- Every statement ends with a semicolon (;).

- We will perform various operations using operators provided by Pig Latin, through statements.

- Except LOAD and STORE, while performing all other operations, Pig Latin statements take a relation as input and produce another relation as output.

- As soon as you enter a Load statement in the Grunt shell, its semantic checking will be carried out. To see the contents of the schema, you need to use the Dump operator. Only after performing the dump operation, the MapReduce job for loading the data into the file system will be carried out.

7.5.4 Pig Execution Modes:

There are three execution modes for pig which are

- **Interactive Mode (Grunt shell):** You can run Apache Pig in interactive mode using the Grunt shell. In this shell, you can enter the Pig Latin statements and get the output (using Dump operator).
- **Batch Mode (Script):** You can run Apache Pig in Batch mode by writing the Pig Latin script in a single file with **.pig** extension.
- **Embedded Mode (UDF):** Apache Pig provides the provision of defining our own functions (**U**ser **D**efined **F**unctions) in programming languages such as Java, and using them in our script.

7.5.5 Pig Processing – Loading and Transforming Data:

Apache Pig, in general, runs on top of Hadoop. It's a statistical tool for analysing huge datasets in the Hadoop File System. To use Apache Pig to analyse data, we must first load the data into Apache Pig. This chapter explains how to load data from HDFS into Apache Pig.

LOAD operator:

The LOAD operator of Pig Latin can be used to load data into Apache Pig from a file system (HDFS/ Local). The "=" operator divides the load statement into two sections. On the left, we must specify the name of the relation in which we want to store the data, and on the right, we must specify how we will store the data. The Load operator's syntax is seen below.

Relation_name = LOAD 'Input file path' USING function as schema;

- **relation_name:** We have to mention the relation in which we want to store the data.
- **Input file path:** We have to mention the HDFS directory where the file is stored. (In MapReduce mode)
- **Function:** We have to choose a function from the set of load functions provided by Apache Pig (BinStorage, JsonLoader, PigStorage, TextLoader).
- **Schema:** We have to define the schema of the data. We can define the required schema as follows:

(column1: data type, column2 : data type, column3 : data type);

Store Operator:

You can store the loaded data in the file system using the store operator. This chapter explains how to store data in Apache Pig using the Store operator.

Syntax:

STORE Relation_name INTO ' required_directory_path ' [USING function];

Loading and Storing Data, we have seen how to load data from external storage for processing in Pig. Storing the results is straightforward, too. Here's an example of using PigStorage to store tuples as plain-text values separated by a colon character: grunt> STORE A INTO 'out' USING PigStorage(':');

```
grunt> cat out

Joe:cherry:2

Ali:apple:3

 Joe:banana:2

Eve:apple:7
```

7.5.6 Pig Built-In Functions:

Type	Examples
EVAL functions	AVG, COUNT, COUNT_STAR, SUM, TOKENIZE, MAX, MIN, SIZE etc
LOAD or STORE functions	Pigstorage(), Textloader, HbaseStorage, JsonLoader, JsonStorage etc
Math functions	ABS, COS, SIN, TAN, CEIL, FLOOR, ROUND, RANDOM etc
String functions	TRIM, RTRIM, SUBSTRING, LOWER, UPPER etc
DateTime function	GetDay, GetHour, GetYear, ToUnixTime, ToString etc

Eval Functions:

AVG(col): computes the average of the numerical values in a single column of a bag

CONCAT(string expression1, string expression2): Concatenates two expressions of identical type

COUNT(DataBag bag): Computes the number of elements in a bag excluding null values

COUNT STAR (DataBag bag1, DataBag bag 2): Computes the number of elements in a bag including null values.

DIFF(DataBag bag1, DataBag bag2): It is used to compare two bags, if any element in one bag is not present in the other bag are returned in a bag

IsEmpty(DataBag bag), IsEmpty(Map map): It is used to check if the bag or map is empty

Max(col): Computes the maximum of the numeric values or character in a single column bag

MIN(col): Computes the minimum of the numeric values or character in a single column bag

DEFINE pluck pluckTuple(expression1): It allows the user to specify a string prefix, and filters the columns which begins with that prefix

SIZE(expression): Computes the number of elements based on any pig data

SUBSTRACT(DataBag bag1, DataBag bag2): It returns the bag which does not contain bag1 element in bag2

SUM: Computes the sum of the values in a single-column bag

TOKENIZE(String expression[,'field delimiter'): It splits the string and outputs a bag of words.

Filtering:

The FILTER operator is used to filter tuples from a relation depending on a criterion.

Syntax:

grunt> Relation2_name = FILTER Relation1_name BY (condition);

For example:

StudentInfo.txt

1,Rajiv,Reddy,21,9848022337,Hyderabad

2,siddarth,Battacharya,22,9848022338,Kolkata

3,Rajesh,Khanna,22,9848022339,Delhi

4,Preethi,Agarwal,21,9848022330,Pune

5,Trupthi,Mohanthy,23,9848022336,Bhuwaneshwar

6,Archana,Mishra,23,9848022335,Chennai

7,Komal,Nayak,24,9848022334,trivendram

8,Bharathi,Nambiayar,24,9848022333,Chennai

Loading of file **StudentInfo.txt**

grunt> student_details = LOAD 'hdfs://localhost:9000/pig_data/StudentInfo.txt' USING PigStorage(',') as (id:int, firstname:chararray, lastname:chararray, age:int, phone:chararray, city:chararray);

Let’s use the Filter operator to get the details of the students who belong to the city Chennai.

filter_data = FILTER student_details BY city == 'Chennai';

7.5.7 Grouping and Sorting:

To group data in one or more relations, use the GROUP operator. It gathers information using the same key.

grunt> Group_data = GROUP Relation_name BY age;

The ORDER BY operator is used to sort the contents of a relation according to one or more fields.

Syntax:

grunt> Relation_name2 = ORDER Relatin_name1 BY (ASC|DESC);

7.5.8 Installation of Pig and Pig Latin Commands:

Before you start using Apache Pig, you must have Hadoop and Java installed on your PC. As a result, before installing Apache Pig, install java and Hadoop.

First of all, download the latest version of Apache Pig from the following website − https://pig.apache.org/

Install Apache Pig:

After downloading the Apache Pig software, install it in your Linux environment by following the steps given below.

STEP 1:

Create a directory with the name Pig in the same directory where the installation directories of Hadoop, Java, and other software were installed. (In our tutorial, we have created the Pig directory in the user named Hadoop).

```
$ mkdir Pig
```

STEP 2:

Extract the downloaded tar files as shown below.

```
$ cd Downloads/

$ tar zxvf pig-0.15.0-src.tar.gz

$ tar zxvf pig-0.15.0.tar.gz
```

STEP 3:

Move the content of pig-0.15.0-src.tar.gz file to the Pig directory created earlier as shown below.

```
$ mv pig-0.15.0-src.tar.gz/* /home/Hadoop/Pig/
```

Configure Apache Pig:

After installing Apache Pig, we have to configure it. To configure, we need to edit two files − bashrc and pig.properties.

.bashrc file

In the .bashrc file, set the following variables:

- PIG_HOME folder to the Apache Pig's installation folder,
- PATH environment variable to the bin folder, and
- PIG_CLASSPATH environment variable to the etc (configuration) folder of your Hadoop installations (the directory that contains the core-site.xml, hdfs-site.xml and mapred-site.xml files).

```
export PIG_HOME = /home/Hadoop/Pig

export PATH = $PATH:/home/Hadoop/pig/bin

export PIG_CLASSPATH = $HADOOP_HOME/conf
```

pig.properties file

In the conf folder of Pig, we have a file named pig.properties. In the pig.properties file, you can set various parameters as given below.

```
pig -h properties
```

7.6 LIST OF REFERENCES

- Tom White, "HADOOP: The definitive Guide" O Reilly 2012, Third Edition,ISBN: 978-1-449-31152-0
- Chuck Lam, "Hadoop in Action", Dreamtech Press 2016, First Edition ,ISBN:139788177228137
- Shiva Achari," Hadoop Essential " PACKT Publications, ISBN 978-1-78439-668-8
- RadhaShankarmani and M. Vijayalakshmi ,"Big Data Analytics "Wiley Textbook Series, Second Edition, ISBN 9788126565757

Web References:

- https://hadoop.apache.org/docs/stable/
- https://pig.apache.org/

11

DATA VISUALIZATION

Unit Structure

11.0 OBJECTIVE

Data visualization is the graphical representation of information and data. By using visual elements like charts, graphs, and maps, data visualization tools provide an accessible way to see and understand trends, outliers, and patterns in data. In the world of Big Data, data visualization tools and technologies are essential to analyse massive amounts of information and make data-driven decisions.

11.1 INTRODUCTION

Data visualization is actually a set of data points and information that are represented graphically to make it easy and quick for user to understand. Data visualization is good if it has a clear meaning, purpose, and is very easy to interpret, without requiring context. Tools of data visualization provide an accessible way to see and understand trends, outliers, and patterns in data by using visual effects or elements such as a chart, graphs, and maps.

Characteristics of Effective Graphical Visual:

- It shows or visualizes data very clearly in an understandable manner.
- It encourages viewers to compare different pieces of data.
- It closely integrates statistical and verbal descriptions of data set.
- It grabs our interest, focuses our mind, and keeps our eyes on message as human brain tends to focus on visual data more than written data.
- It also helps in identifying area that needs more attention and improvement.
- Using graphical representation, a story can be told more efficiently. Also, it requires less time to understand picture than it takes to understand textual data.

Categories of Data Visualization:

Data visualization is very critical to market research where both numerical and categorical data can be visualized that helps in an increase in impacts of insights and also helps in reducing risk of analysis paralysis. So, data visualization is categorized into following categories :

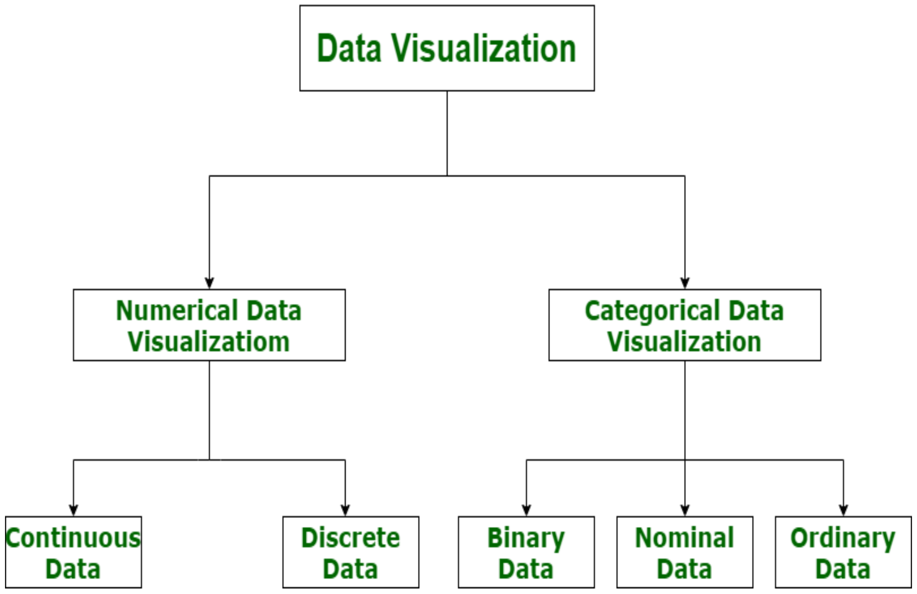


Figure – Categories of Data Visualization

1. Numerical Data:

Numerical data is also known as Quantitative data. Numerical data is any data where data generally represents amount such as height, weight, age of a person, etc. Numerical data visualization is easiest way to visualize data. It is generally used for helping others to digest large data sets and raw numbers in a way that makes it easier to interpret into action. Numerical data is categorized into two categories :

Continuous Data:

It can be narrowed or categorized (Example: Height measurements).

Discrete Data:

This type of data is not "continuous" (Example: Number of cars or children's a household has).

The type of visualization techniques that are used to represent numerical data visualization is Charts and Numerical Values. Examples are Pie Charts, Bar Charts, Averages, Scorecards, etc.

2. Categorical Data:

Categorical data is also known as Qualitative data. Categorical data is any data where data generally represents groups. It simply consists of categorical variables that are used to represent characteristics such as a person’s ranking, a person’s gender, etc. Categorical data visualization is all about depicting key themes, establishing connections, and lending context. Categorical data is classified into three categories :

Binary Data:

In this, classification is based on positioning (Example: Agrees or Disagrees).

Nominal Data:

In this, classification is based on attributes (Example: Male or Female).

Ordinal Data:

In this, classification is based on ordering of information (Example: Timeline or processes).

The type of visualization techniques that are used to represent categorical data is Graphics, Diagrams, and Flowcharts. Examples are Word clouds, Sentiment Mapping, Venn Diagram, etc.

11.2 EXPLANATION OF DATA VISUALIZATION

What is Data Visualization?

Data visualization is a graphical representation of quantitative information and data by using visual elements like graphs, charts, and maps. Data visualization convert large and small data sets into visuals, which is easy to understand and process for humans. Data visualization tools provide accessible ways to understand outliers, patterns, and trends in the data. In the world of Big Data, the data visualization tools and technologies are required to analyse vast amounts of information.

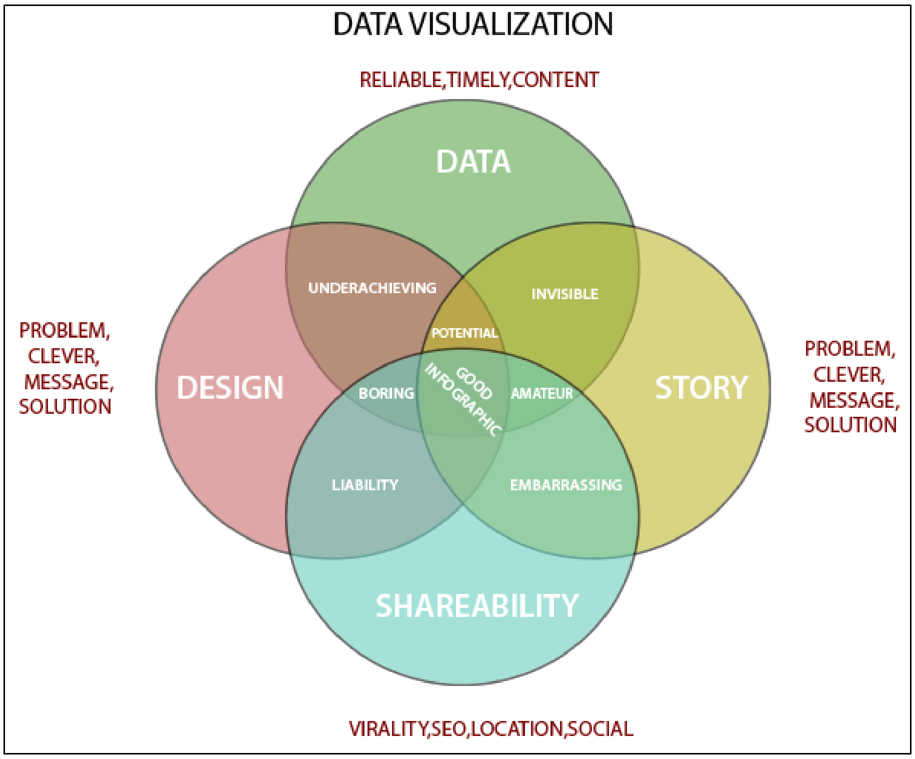
Data visualizations are common in your everyday life, but they always appear in the form of graphs and charts. The combination of multiple visualizations and bits of information are still referred to as Infographics. Data visualizations are used to discover unknown facts and trends. You can see visualizations in the form of line charts to display change over time. Bar and column charts are useful for observing relationships and making comparisons. A pie chart is a great way to show parts-of-a-whole. And maps are the best way to share geographical data visually.

Today's data visualization tools go beyond the charts and graphs used in the Microsoft Excel spreadsheet, which displays the data in more sophisticated ways such as dials and gauges, geographic maps, heat maps, pie chart, and fever chart.

What makes Data Visualization Effective?

Effective data visualization are created by communication, data science, and design collide. Data visualizations did right key insights into complicated data sets into meaningful and natural.

American statistician and Yale professor **Edward Tufte** believe useful data visualizations consist of ?complex ideas communicated with clarity, precision, and efficiency.



To craft an effective data visualization, you need to start with clean data that is well-sourced and complete. After the data is ready to visualize, you need to pick the right chart.

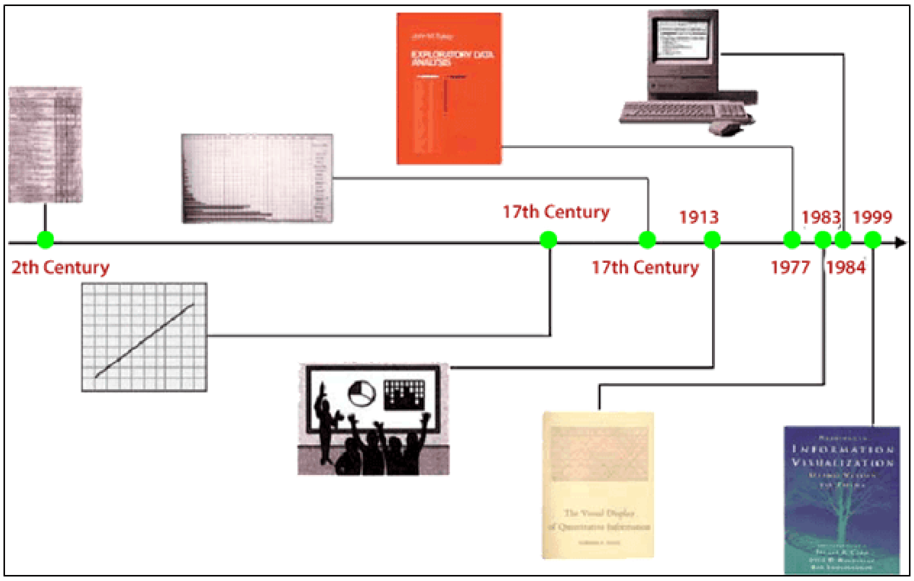
After you have decided the chart type, you need to design and customize your visualization to your liking. Simplicity is essential - you don't want to add any elements that distract from the data.

History of Data Visualization:

The concept of using picture was launched in the 17th century to understand the data from the maps and graphs, and then in the early 1800s, it was reinvented to the pie chart.

Several decades later, one of the most advanced examples of statistical graphics occurred when **Charles Minard** mapped Napoleon's invasion of Russia. The map represents the size of the army and the path of Napoleon's retreat from Moscow - and that information tied to temperature and time scales for a more in-depth understanding of the event.

Computers made it possible to process a large amount of data at lightning-fast speeds. Nowadays, data visualization becomes a fast-evolving blend of art and science that certain to change the corporate landscape over the next few years.


17th Century
1913
1983
1999
2th Century
17th Century
1977
1984
EXPLORATORY DATA ANALYSIS
INFORMATION VISUALIZATION
The Visual Display of Quantitative Information

Importance of Data Visualization:

Data visualization is important because of the processing of information in human brains. Using graphs and charts to visualize a large amount of the complex data sets is more comfortable in comparison to studying the spreadsheet and reports.

Data visualization is an easy and quick way to convey concepts universally. You can experiment with a different outline by making a slight adjustment.

Data visualization has some more specialties such as:

- Data visualization can identify areas that need improvement or modifications.
- Data visualization can clarify which factor influence customer behaviour.
- Data visualization helps you to understand which products to place where.
- Data visualization can predict sales volumes.

Data visualization tools have been necessary for democratizing data, analytics, and making data-driven perception available to workers throughout an organization. They are easy to operate in comparison to earlier versions of BI software or traditional statistical analysis software. This guide to a rise in lines of business implementing data visualization tools on their own, without support from IT.

Why Use Data Visualization?

1. To make easier in understand and remember.

2. To discover unknown facts, outliers, and trends.

3. To visualize relationships and patterns quickly.
4. To ask a better question and make better decisions.
5. To competitive analyse.
6. To improve insights.

11.3 CHALLENGES OF BIG DATA VISUALIZATION

The challenges in Big Data are the real implementation hurdles. These require immediate attention and need to be handled because if not handled then the failure of the technology may take place which can also lead to some unpleasant result. Big data challenges include the storing, analysing the extremely large and fast-growing data.

Some of the Big Data challenges are:

1. Sharing and Accessing Data:

- Perhaps the most frequent challenge in big data efforts is the inaccessibility of data sets from external sources.
- Sharing data can cause substantial challenges.
- It includes the need for inter and intra- institutional legal documents.
- Accessing data from public repositories leads to multiple difficulties.
- It is necessary for the data to be available in an accurate, complete and timely manner because if data in the company's information system is to be used to make accurate decisions in time then it becomes necessary for data to be available in this manner.

2. Privacy and Security:

- It is another most important challenge with Big Data. This challenge includes sensitive, conceptual, technical as well as legal significance.
- Most of the organizations are unable to maintain regular checks due to large amounts of data generation. However, it should be necessary to perform security checks and observation in real time because it is most beneficial.
- There is some information of a person which when combined with external large data may lead to some facts of a person which may be secretive and he might not want the owner to know this information about that person.
- Some of the organization collects information of the people in order to add value to their business. This is done by making insights into their lives that they're unaware of.

3. Analytical Challenges:

- There are some huge analytical challenges in big data which arise some main challenges questions like how to deal with a problem if data volume gets too large?
- Or how to find out the important data points?
- Or how to use data to the best advantage?
- These large amount of data on which these type of analysis is to be done can be structured (organized data), semi-structured (Semi-organized data) or unstructured (unorganized data). There are two techniques through which decision making can be done:
- Either incorporate massive data volumes in the analysis.
- Or determine upfront which Big data is relevant.

4. Technical challenges:

Quality of data:

- When there is a collection of a large amount of data and storage of this data, it comes at a cost. Big companies, business leaders and IT leaders always want large data storage.
- For better results and conclusions, Big data rather than having irrelevant data, focuses on quality data storage.
- This further arise a question that how it can be ensured that data is relevant, how much data would be enough for decision making and whether the stored data is accurate or not.

Fault tolerance:

- Fault tolerance is another technical challenge and fault tolerance computing is extremely hard, involving intricate algorithms.
- Nowadays some of the new technologies like cloud computing and big data always intended that whenever the failure occurs the damage done should be within the acceptable threshold that is the whole task should not begin from the scratch.

Scalability:

- Big data projects can grow and evolve rapidly. The scalability issue of Big Data has lead towards cloud computing.
- It leads to various challenges like how to run and execute various jobs so that goal of each workload can be achieved cost-effectively.
- It also requires dealing with the system failures in an efficient manner. This leads to a big question again that what kinds of storage devices are to be used.

11.4 APPROACHES TO BIG DATA VISUALIZATION

Data visualization is used in many areas to model complex events and visualize phenomena that cannot be observed directly, such as weather patterns, medical conditions or mathematical relationships. Here we review basic data visualization tools and techniques.

By SciForce:

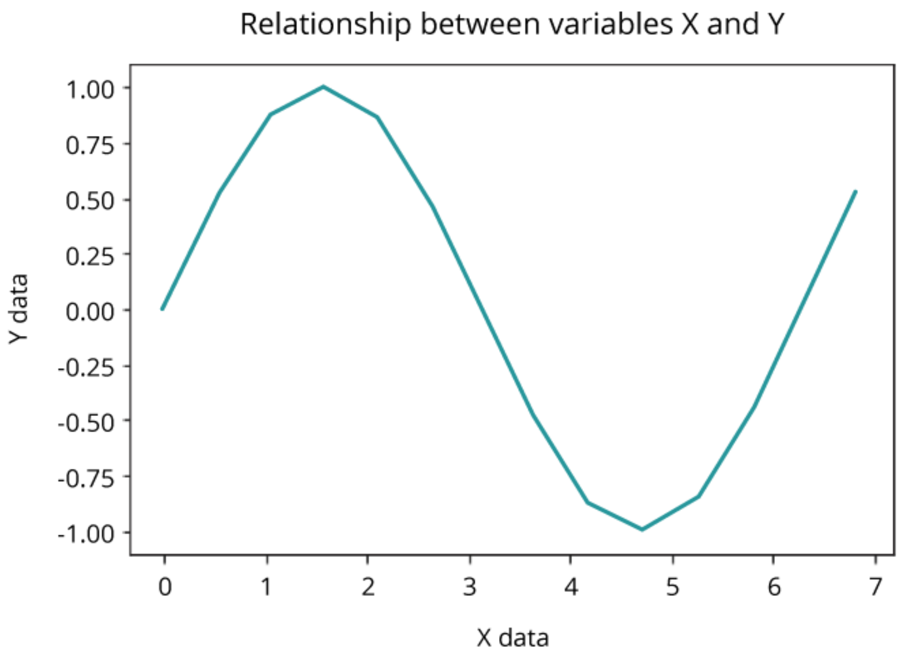
Researchers agree that vision is our dominant sense: 80–85% of information we perceive, learn or process is mediated through vision. It is even more so when we are trying to understand and interpret data or when we are looking for relationships among hundreds or thousands of variables to determine their relative importance. One of the most effective ways to discern important relationships is through advanced analysis and easy-to-understand visualizations.

Data visualization is applied in practically every field of knowledge. Scientists in various disciplines use computer techniques to model complex events and visualize phenomena that cannot be observed directly, such as weather patterns, medical conditions or mathematical relationships.

Data visualization provides an important suite of tools and techniques for gaining a qualitative understanding. The basic techniques are the following plots:

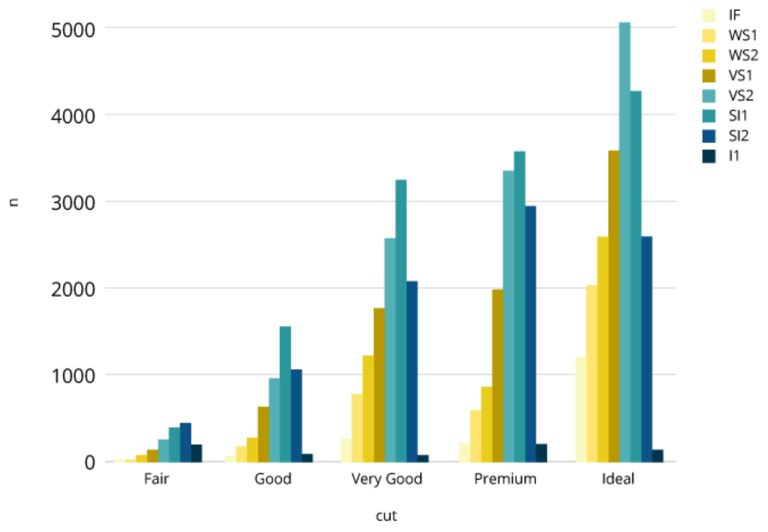
Line Plot:

The simplest technique, a line plot is used to plot the relationship or dependence of one variable on another. To plot the relationship between the two variables, we can simply call the plot function.



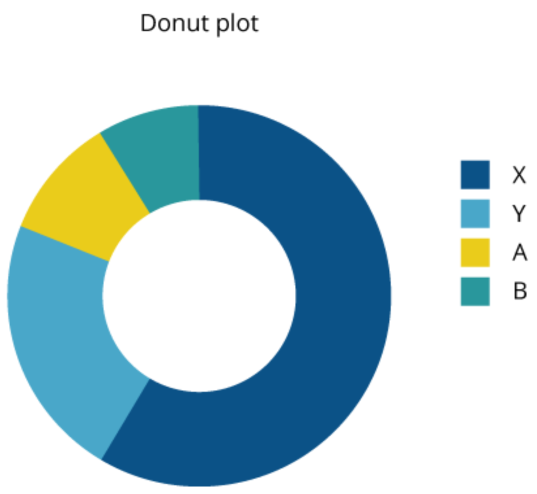
Bar Chart:

Bar charts are used for comparing the quantities of different categories or groups. Values of a category are represented with the help of bars and they can be configured with vertical or horizontal bars, with the length or height of each bar representing the value.



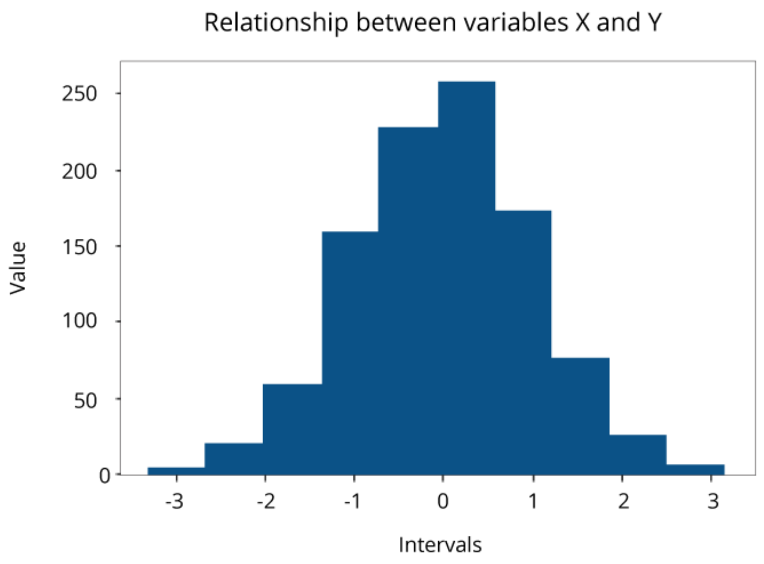
Pie and Donut Charts:

There is much debate around the value of pie and donut charts. As a rule, they are used to compare the parts of a whole and are most effective when there are limited components and when text and percentages are included to describe the content. However, they can be difficult to interpret because the human eye has a hard time estimating areas and comparing visual angles.



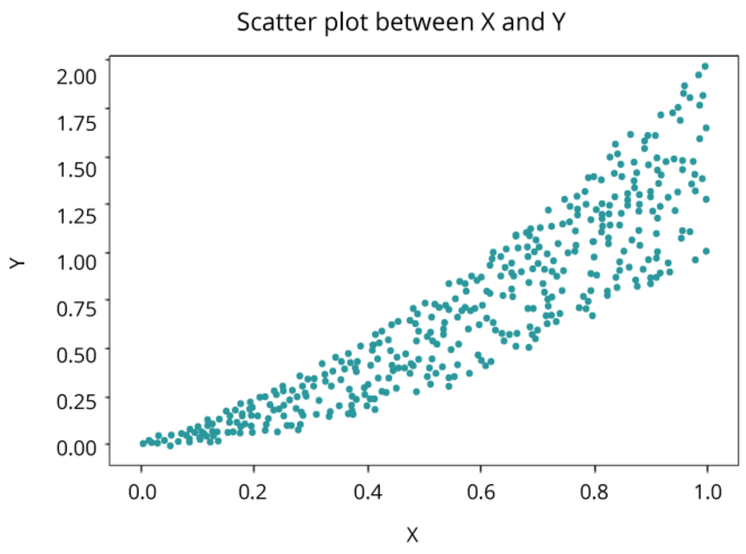
Histogram Plot:

A histogram, representing the distribution of a continuous variable over a given interval or period of time, is one of the most frequently used data visualization techniques in machine learning. It plots the data by chunking it into intervals called ‘bins’. It is used to inspect the underlying frequency distribution, outliers, skewness, and so on.



Scatter Plot:

Another common visualization techniques is a scatter plot that is a two-dimensional plot representing the joint variation of two data items. Each marker (symbols such as dots, squares and plus signs) represents an observation. The marker position indicates the value for each observation. When you assign more than two measures, a scatter plot matrix is produced that is a series of scatter plots displaying every possible pairing of the measures that are assigned to the visualization. Scatter plots are used for examining the relationship, or correlations, between X and Y variables.

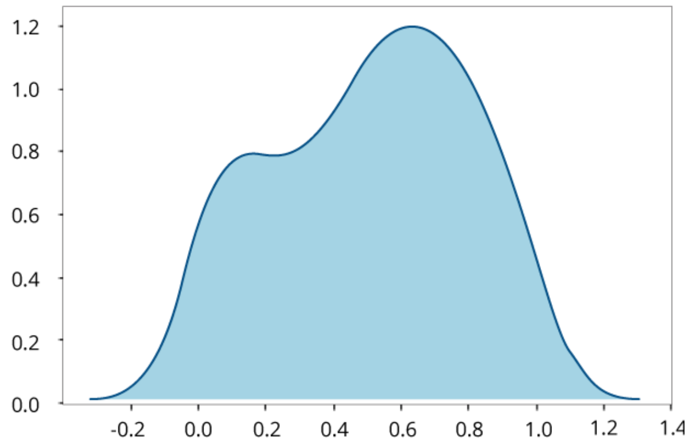


Visualizing Big Data:

Today, organizations generate and collect data each minute. The huge amount of generated data, known as Big Data, brings new challenges to visualization because of the speed, size and diversity of information that must be considered. The volume, variety and velocity of such data requires from an organization to leave its comfort zone technologically to derive intelligence for effective decisions. New and more sophisticated visualization techniques based on core fundamentals of data analysis take into account not only the cardinality, but also the structure and the origin of such data.

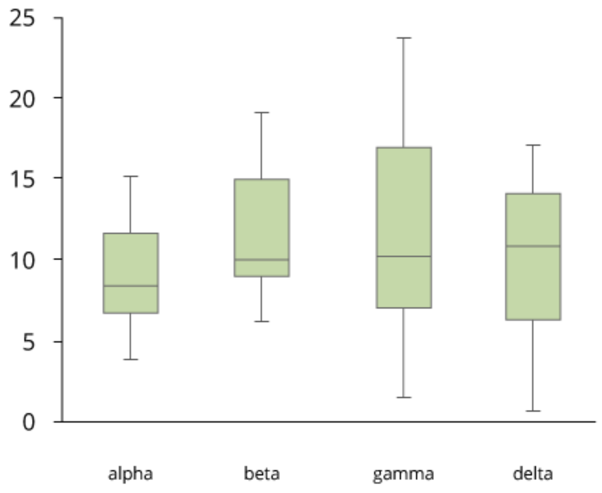
Kernel Density Estimation for Non-Parametric Data:

If we have no knowledge about the population and the underlying distribution of data, such data is called non-parametric and is best visualized with the help of Kernel Density Function that represents the probability distribution function of a random variable. It is used when the parametric distribution of the data doesn't make much sense, and you want to avoid making assumptions about the data.



Box and Whisker Plot for Large Data:

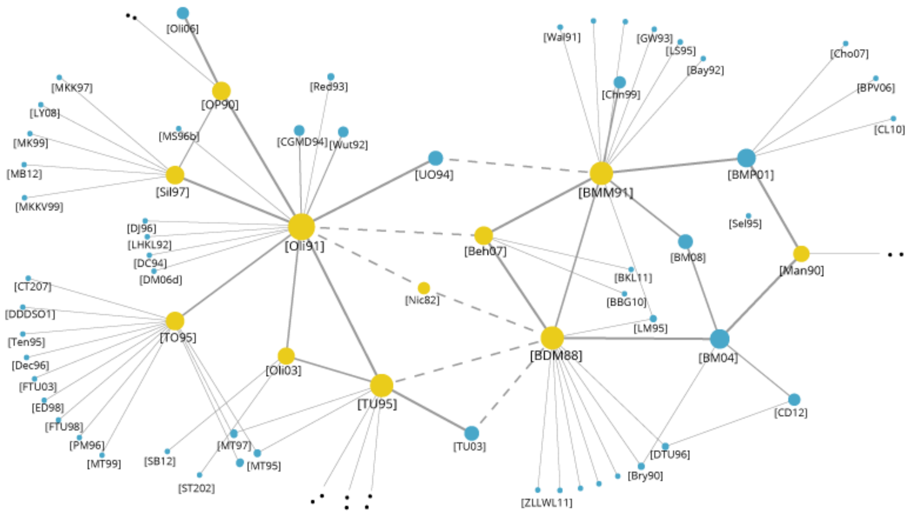
A binned box plot with whiskers shows the distribution of large data and easily see outliers. In its essence, it is a graphical display of five statistics (the minimum, lower quartile, median, upper quartile and maximum) that summarizes the distribution of a set of data. The lower quartile (25th percentile) is represented by the lower edge of the box, and the upper quartile (75th percentile) is represented by the upper edge of the box. The median (50th percentile) is represented by a central line that divides the box into sections. Extreme values are represented by whiskers that extend out from the edges of the box. Box plots are often used to understand the outliers in the data.



Word Clouds and Network Diagrams for Unstructured Data:

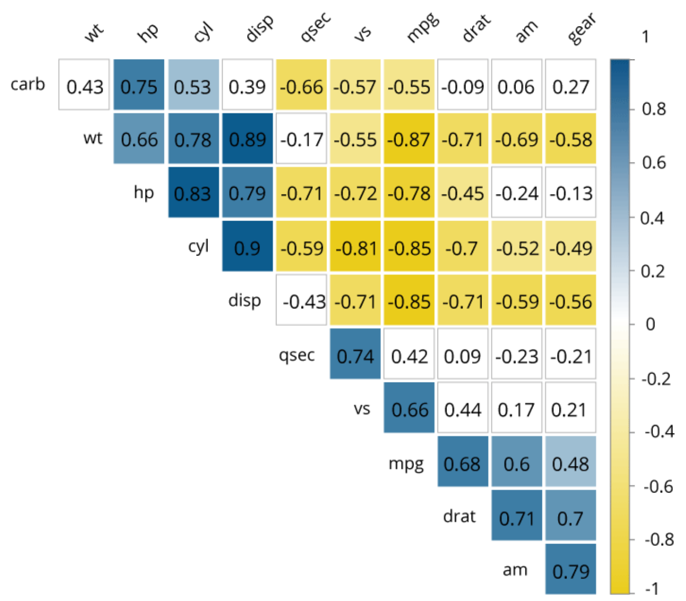
The variety of big data brings challenges because semistructured and unstructured data require new visualization techniques. A word cloud visual represents the frequency of a word within a body of text with its relative size in the cloud. This technique is used on unstructured data as a way to display high- or low-frequency words.

Another visualization technique that can be used for semistructured or unstructured data is the network diagram. Network diagrams represent relationships as nodes (individual actors within the network) and ties (relationships between the individuals). They are used in many applications, for example for analysis of social networks or mapping product sales across geographic areas.



Correlation Matrices:

A correlation matrix allows quick identification of relationships between variables by combining big data and fast response times. Basically, a correlation matrix is a table showing correlation coefficients between variables: Each cell in the table represents the relationship between two variables. Correlation matrices are used as a way to summarize data, as an input into a more advanced analysis, and as a diagnostic for advanced analyses.



Data visualization may become a valuable addition to any presentation and the quickest path to understanding your data. Besides, the process of visualizing data can be both enjoyable and challenging. However, with the many techniques available, it is easy to end up presenting the information using a wrong tool. To choose the most appropriate visualization technique you need to understand the data, its type and composition, what information you are trying to convey to your audience, and how viewers process visual information. Sometimes, a simple line plot can do the task saving time and effort spent on trying to plot the data using advanced Big Data techniques. Understand your data—and it will open its hidden values to you.

*****