* Big Data Analytics :→

Big Data Analytics is the often complex process of examining big data to uncover information -- such as hidden patterns, correlations, market trends and customer preferences -- that can help organizations make informed business decisions.

On a broad scale, data analytics technologies and techniques give organizations a way to analyze data sets and gather new information. Business intelligence (BI) queries answer basic questions about business operations and performance.

Big Data Analytics is a form of advanced analytics, which involve complex applications with elements such as predictive models, statistical algorithms and what-if analysis powered by analytics systems.

* Objective of Big Data Analytics :→

Big Data analysis helps organizations take advantage of their data and use it to identify new opportunities.

1. To provide an overview of an exciting growing field of big data analytics.
2. To introduce the tools required to manage and analyze big data like Hadoop, NoSql, MapReduce.

Name of Lecturer :

3. To teach the fundamental techniques and principles in achieving big data analytics with scalability and streaming capability.

* Scope & outcome of Big Data Analytics :→

Big Data is defined as massive amount of data which is too large and complex to be stored in traditional databases. Data has evolved over the last 5 years. Lots of data is being generated each day in every business sector.

Below are some facts about Big Data for some of the companies →

1. 40,000 search queries are performed on Google per second. i.e- 3.46 million searches a day.
2. Every minute, users send 31.25 million message and watch 2.77 million videos on facebook.
3. 55 billion messages and 4.5 billion photos are sent each day on WhatsApp.
4. By 2025, the volume of digital data will increase to 163 Zettabytes.
5. Walmart handles more then 1 million customer transactions every hour.

* **Big Data** →

According to "Gartner" Big Data is high-Volume, Velocity and Variety information assets that demand Cost-effective, innovative forms of information processing for enhanced insight and decision making.

Big Data is a term that describes the large volume of data - both structured and unstructured - that inundates a business on a day-to-day basic. Big data can be analyzed for insights that lead to better decisions and strategic business moves.

* **Challenges of Big Data** :→

Many Companies get stuck at the initial stage of their Big Data projects. This is because they are neither aware of the challenges of Big Data nor are equipped to tackle those challenges.
Let us understand them one by one -

(1.) **Lack of proper understanding of Big Data** :→

Companies fail in their Big Data initiatives due to insufficient understanding. Employees may not know what data is, its storage, processing, importance, and sources.

(2) **Data growth issues** :→

one of the most pressing challenges of Big Data is storing all these huge sets of Data properly. The amount of data being stored in data centers and

databases of Companies is increasing rapidly. As these data sets grow exponentially with time, it gets extremely difficult to handle.
Most of the data is unstructured and comes from documents, videos, audios, text files and other sources. This means that you cannot find them in databases.

(3) Confusion while Big Data tool selection :→
Companies often get confused while selecting the best tool for Big Data analysis and storage. ~~Is~~ ~~HBase or Cassandra the best technology for data storage~~

(4.) Lack of data professionals :→
To run these modern technologies and Big Data tools, Companies need skilled data professionals. These professionals will include data scientists, data analysts and data engineers who are experienced in working with the tools and making sense out of huge data sets.
Companies face a problem of lack of Big Data professionals. This is because data ~~handler~~ handling tools have ~~en~~ evolved rapidly, but in most cases, the professionals have not.

(5) Integrating data from a Variety of Sources :→
Data in an organization comes from a variety of sources, such as social media pages, ERP applications, customer logs, financial reports, e-mails, presentations and reports created by employees. Combining all this data to prepare reports is a challenging task.

(6.) Sharing (7) Searching (8) Presentation.

** Problems with Traditional Large-Scale System :→

Because data is so expensive to store in traditional systems, data is filtered and aggregated, and large volumes are thrown out because of the cost of storage. Minimizing the data to be analyzed reduces the accuracy and confidence of the results. Not only are accuracy and confidence to the resulting data affected, but it also limits an organization's ability to identify business opportunities. Atomic data can ~~yield~~ yield more insights into the data than aggregated data.

** Sources of Big Data :→

The bulk of big data generated comes from primary sources. In addition, companies need to make the distinction between data which is generated internally, that is to say it resides behind a company's firewall and externally data generated which needs to be ~~important~~ imported into a system.

The Primary Sources of Big Data :→

(1) Social Data :→

Social data comes from the Likes, Tweets & Retweets, Comments, Video uploads, and general media that are uploaded and shared via the world's favorite social media platforms.

(2) Machine Data :→

Machine data is defined as information which is generated by industrial equipment, sensors that are installed in machinery, and even web logs which track user behavior. This type of data is expected to grow exponentially as the internet of things grows ever more pervasive and expands around the world.

(3) Transactional Data :→

Transactional Data is generated from all the daily transactions that take place both online and offline. Invoices, Payment orders, Storage records, delivery receipts - all are characterized as transactional data yet data alone is almost meaningless, and most organizations struggle to make sense of the data that they are generating and how it can be put to good use.

(4) Share Market :→

Stock exchange across the world generates huge amount of data through its daily transaction.

(5) Telecom Company :→

Telecom giants like Airtel, Vodafone study the user trends and accordingly publish their plans and for this they store the data of its million users.

(6) E-Commerce Site :→

Sites like Amazon, Flipkart, Alibaba generates huge amount of logs from which users buying trends can be traced.

(7) Weather Station :→

All the weather station and Satellite gives very huge data which are stored and manipulated to forecast weather.

3 Vs of Big Data :→ (3 properties)

This Conception theory gained thrust in the early 2000s when teade and business analyst Mr. Doug Laney expressed the mainsteam explanation of the keyword big Data over the 3 pillars of 3 V's.

(1) Volume :→

organizations and firms gather as well as pull together different data from different sources, which includes business transactions and data, data from social media, login data, as well as information from the sensor as well as machine-to-machine data. Earlier, this data storage would have been an issue – but because of the advent of new technologies for handling extensive data with tools like Apache Spark, Hadoop, the burden of enormous data got decreased.

(2) Velocity :→

Data is now streaming at an exceptional speed, which has to be dealt with suitably. Sensors, Smart metering, User data as well as RFID tags are lashing the need for dealing with an inundation of data in near real-time.

(3) Variety :→

The released releases of data from various systems have diverse types and formats. They range from structured to unstructured, numeric data of traditional databases to non-numeric or text documents, emails, audios and videos, stock ticker data, login data, Blockchains encrypted data, or even financial transactions.

* Importance of Big Data :→

Big Data does not take care of how much data is there, but how it can be used. Data can be taken from various sources for analyzing it and finding answers which enable:

- Reduction in cost
- Time reductions
- New product development with optimized offers.
- well-groomed decision making.

When we merge big data with high-powered data analytics, it is possible to achieve business-related tasks like:-

- Real time determinating of core causes of failures, problem or faults.
- Produce token and coupons as per the customer's buying behavior.
- Risk-management can be done in minutes by calculating risk portfolios.
- Detection of deceptive behaviour before its influence.

* Types of Big Data :→

1. Structured :→

Structured is one of the types of big data and By Structured data, we mean data that can be processed, stored, and retrieved in a fixed format. It refers to highly organized information that can be readily and seamlessly stored and accessed from a database by simple search engine algorithms. for example→ Relational Data.
For instance, the Employee table in a Company database will be Structured as the Employee details, their Job positions, their Salaries, etc. will be present in an organized manner.

2. Unstructured :→

Unstructured data refers to the data that lacks any specific form or structure whatsoever. This makes it very difficult and time-consuming to process and analyze Unstructured data. for example :- word, pdf, Text, media Logs. Email is an example of Unstructured data.

3. Semi-Structured :→

Semi-Structured is the third type of big data. Semi-Structured data pertains to the data containing both the formats mentioned above, that is, Structured and Unstructured data.
for example - XML Data.

Working with Big Data

* **Google File System (GFS) →**

Google file System (GFS) is a Scalable Distributed file System (DFS) created by Google Inc. and developed to accommodate Google's expanding data processing requirements. GFS provides fault ~~tor~~ tolerance, reliability, Scalability, availability and performance to large networks and Connected nodes. GFS is made up of several storage Systems built from low-Cost Commodity hardware Components. It is optimized to accommodate Google's different data use and storage needs, Such as its Search Engine, which generates huge amounts of data that must be stored.
The Google file System Capitalized on the Strength of off-the-Shelf Servers while minimizing hardware weaknesses.
[GFS is also known as GoogleFS.]

- GFS has Snapshot and record append operations. Snapshot Creates a Copy of a file or a directory tree at low Cost.
- Record append allows multiple Clients to append data to the Same file Concurrently while guaranteeing the atomicity of each individual Clients append.

⇒ **Architecture of GFS :-**

A GFS Cluster Consists of a Single master and multiple Chunk Servers and is accessed by multiple Clients, as shown in the following figure :-

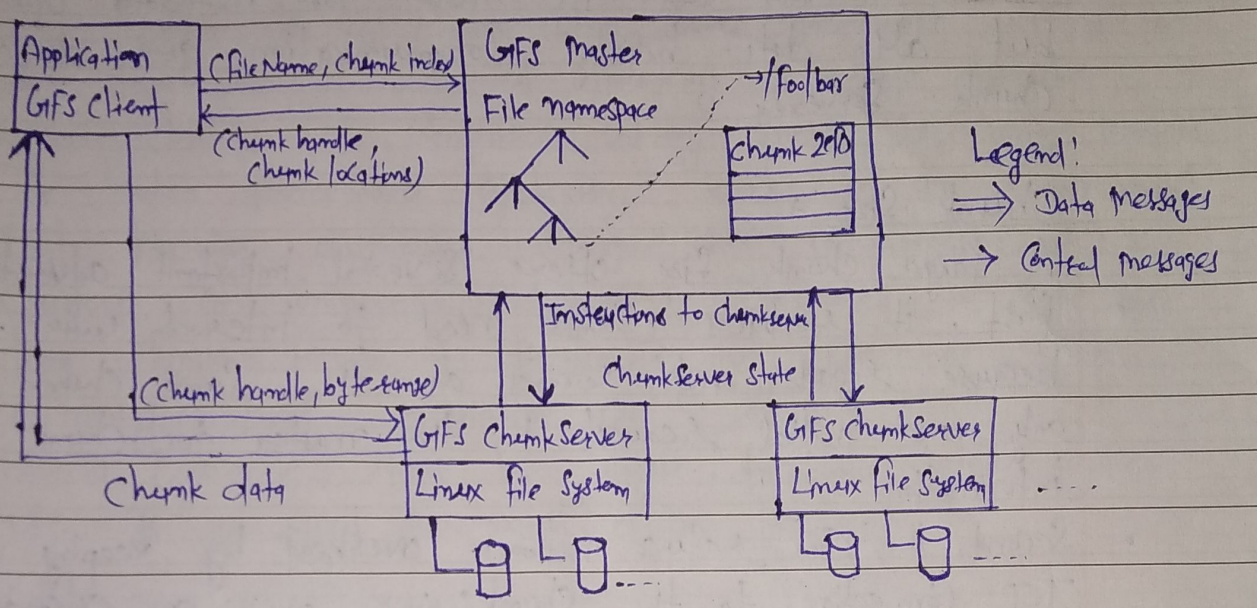


Figure -1 : GFS Architecture

- Each of these is typically a commodity Linux machine running a user-level server process.
- Files are divided into fixed-size chunks. Each chunk is identified by a fixed and globally unique 64-bit chunk handle assigned by the master at the time of chunk creation.
- Chunk servers store chunks on local disks as Linux files.
- For scalability, each chunk is replicated on multiple chunk servers.
- The master maintains all ~~five~~ file system metadata. This includes the namespace, access control information, the mapping from files to chunk and the current locations of chunks.
- It also controls system-wide activities such as chunk lease management, garbage collections of orphaned chunks, and chunk migration between chunk servers.

- The master periodically Communication with each chunk Server in Heart Beat messages to give it instructions and Collect its State.
- Clients interact with the master for metadata operations, but all data-bearing Communication goes directly to the chunk Servers.

Chunk Size :→

- A large chunk Size offers several important advantages.
- First, it reduces clients need to interact with the master because reads and writes on the Same chunk requires only one initial request to the master for chunk location information.
- Second, it can reduce network overhead by keeping a persistent TCP Connection to the chunkserver over an extended period of time.
- Third, it reduces the Size of the metadata Stored on the master.

* Hadoop :→

Hadoop is an Apache open source framework written in Java that allows distributed processing of large datasets across clusters of computers using simple programming models. The Hadoop framework application works in an environment that provides distributed storage and computation across clusters of computers. Hadoop is designed to scale up from single server to thousands of machines, each offering local computation and storage.
It is currently used by Google, facebook, Yahoo, Twitter etc. It is used for batch/offline processing.

* Advantages of Hadoop :→

1. fast :→

In HDFS the data distributed over the cluster and are mapped which helps in faster retrieval. Even the tools to process the data are often on the same servers, thus reducing the processing time. It is able to process terabytes of data in minutes and peta byte in hours.

2. Scalable :→

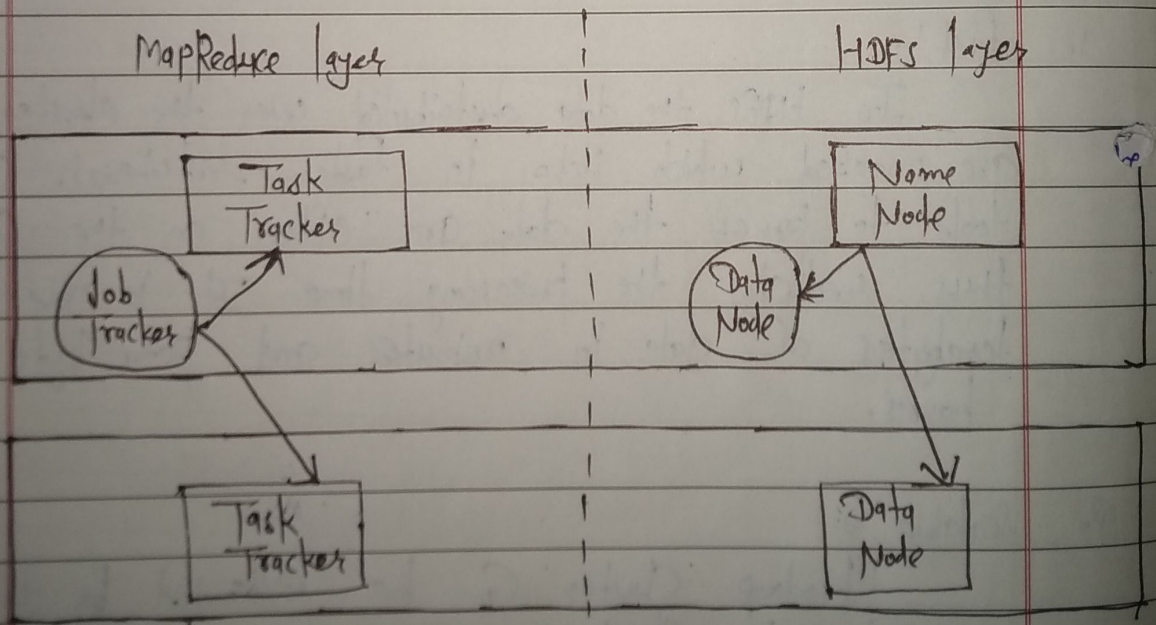
Hadoop Cluster can be extended by just adding nodes in the cluster.

3. Cost Effective :→

Hadoop is open source and uses commodity hardware to store data so it really cost effective as compared to traditional relational database management system.

* Hadoop Architecture :→

The Hadoop architecture is a package of the file system, mapReduce engine and the HDFS (Hadoop Distributed file system). The mapReduce engine can be mapReduce/MR1 or YARN/MR2. A Hadoop cluster consists of a single master and multiple slave nodes.



1. MapReduce Layer :→

The MapReduce comes into existence when the client application submits the MapReduce job to Job Tracker.

Hadoop Distributed File System (HDFS) :→

The Hadoop Distributed file System (HDFS) is the primary data storage system used by Hadoop Application. HDFS employs a NameNode and DataNode architecture to implement a distributed file system that provides high-performance access to data across highly scalable Hadoop clusters.

Hadoop itself is an open source distributed processing framework that manages data processing and storage for big data applications. HDFS is a key part of the many Hadoop ecosystem technologies. It provides a reliable means for managing pools of big data and supporting related big data analytics applications.

→ Advantages of HDFS :→

1. Hadoop framework allows the user to quickly write and test distributed systems. It is efficient and it automatic distributes the data and work across the machines and in turn, utilizes the underlying parallelism of the CPU cores.

2. Servers can be added or removed from the cluster dynamically and Hadoop continues to operate without interruption.

3. Another big advantage of Hadoop is that apart from being open source, it is compatible on all the platforms since it is Java based.

* Features of HDFS :→

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

* Goals of HDFS :→

1. Fault detection and recovery :→

Since HDFS includes a large number of commodity hardware, failure of components is frequent. Therefore HDFS should have mechanisms for quick and automatic fault detection and recovery.

2. Huge dataset :→

HDFS should have hundreds of nodes per cluster to manage the applications having huge datasets.

3 Hardware at data :→

A requested task can be done efficiently, when the computation takes place near the data. Especially where huge datasets are involved, it reduces the network traffic and increases the throughput.

HDFS Architecture / Building blocks of Hadoop / Hadoop Daemons' →

HDFS is responsible for storing data on the cluster in Hadoop. files in HDFS are split into blocks before they are stored on cluster of size 64 MB or 128 MB. On a fully configured cluster, "running Hadoop" means running a set of daemons, resident programs, on the different servers in your network. These daemons have specific roles; some exist only on one server, some exist across multiple servers. The daemons include-

1. NameNode
2. DataNode
3. Secondary NameNode
4. Job Tracker
5. Task Tracker

1. NameNode' →

Hadoop employs a master/slave architecture for both distributed storage and distributed computation. The distributed storage system is called the Hadoop file system, or HDFS.

The NameNode is the master of HDFS that directs the slave DataNode daemons to perform the low-level I/O tasks.

The NameNode is the bookkeeper of HDFS; it keeps track of how your files are broken down into file blocks, which nodes store those blocks, and the overall health of the distributed file system.

The function of the NameNode is memory and I/O intensive. As such, the server hosting the NameNode typically doesn't store

any user data or perform any computations for a MapReduce program to lower the workload on the machine.

2. **DataNode** :→

Each slave machine in your cluster will host a DataNode daemon to perform the grunt work of the distributed filesystem reading and writing HDFS blocks to actual files on the local filesystem. When we want to read or write a HDFS file, the file is broken into blocks and the NameNode will tell your client which DataNode each block resides in.

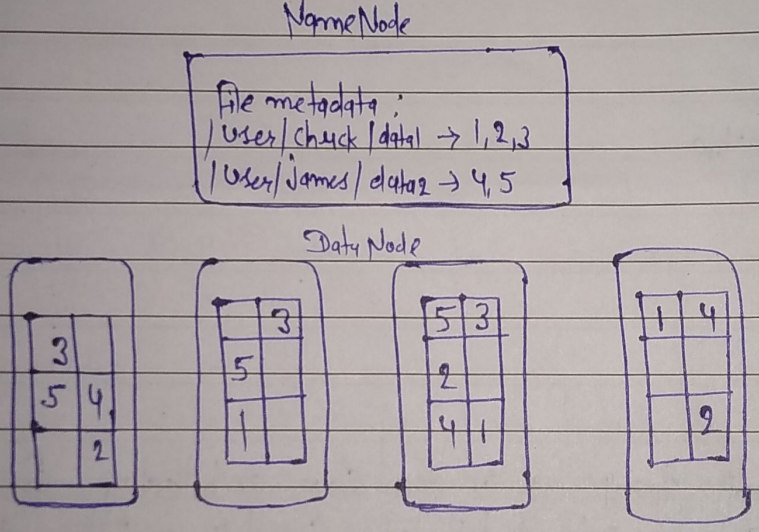


figure :- NameNode / DataNode interaction in HDFS.

The above figure illustrates the role of the NameNode and DataNodes. In the figure, we show two data files, one at /user/chuck/data1 and another at /user/james/data2. ~~The data~~ The data1 file takes up three blocks, which we denote 1, 2 and 3 and the data2 file consists of block 4 and 5. The content of the files are distributed among the DataNodes. In the illustration, each block has three replicas.

for example, block1 (used for data1) is replicated over the three rightmost DataNodes. This ensures that if any one DataNode crashes or ~~beca~~ becomes inaccessible over the network, you'll still be able to read the files.

The NameNode keeps track of the file metadata - which files are in the system and how each file is broken down into blocks. The DataNodes provides backup store of the blocks and constantly report to the NameNode to keep the metadata current.

3. Secondary NameNode :→

The Secondary NameNode (SNN) is an assistant ~~dae~~ daemon for monitoring the state of the cluster HDFS. Like the NameNode, each cluster has one SNN, and it typically resides on its own machine as well. No other DataNode or TaskTracker ~~daemons~~ daemons run on the same server. The SNN differs from the NameNode in ~~the~~ that this process doesn't receive or record any real-time changes to HDFS.

The NameNode is a single point of failure for a Hadoop cluster, and the SNN snapshots help minimize the downtime and loss of data.

4. Job Tracker :→

The JobTracker daemon is the liaison between your application and Hadoop. Once you submit your code to our cluster, the JobTracker determines the execution plan by determining which files to process, assigns nodes to different tasks, and monitors all tasks as they're running.

Should a task fail, the JobTracker will automatically relaunch the task, possibly on a different node, up to a predefined limit of retries. There is only one JobTracker daemon per Hadoop cluster. It's typically run on a server as a master node of the cluster.

5. Task Tracker :→

The JobTracker is the master for the overall execution of a MapReduce job and the TaskTrackers manage the execution of individual tasks on each slave node.

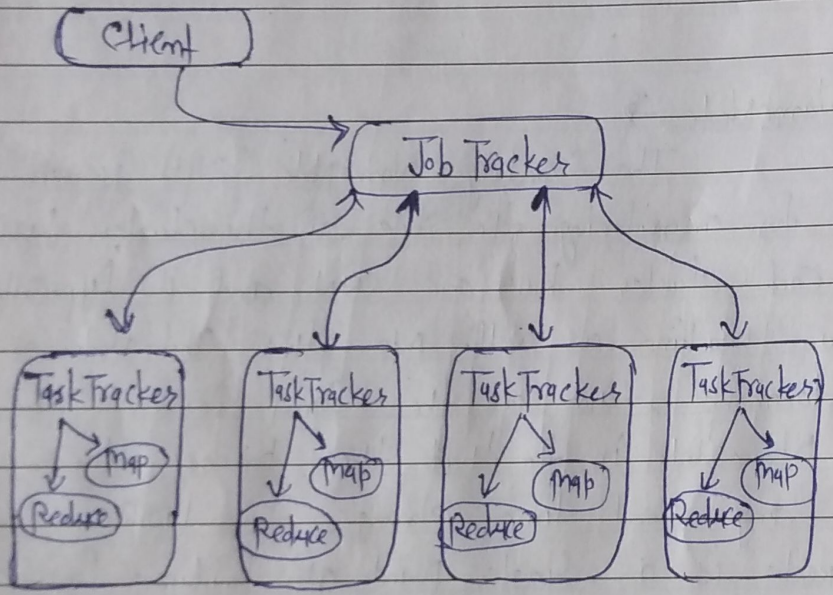


figure: JobTracker and TaskTracker interaction

The above figure shows this interaction
Each TaskTracker is responsible for executing the individual tasks that the JobTracker assigns. Although there is a single TaskTracker per slave node, each TaskTracker can spawn multiple JVMs to handle many map or reduce tasks in parallel.
one responsibility of the TaskTracker is to constantly communicate with the JobTracker.

If the JobTracker fails to receive a heartbeat from a TaskTracker within a specified amount of time, it will assume the TaskTracker has crashed and will resubmit the corresponding tasks to other nodes in the cluster.

After a client calls the JobTracker to begin a data processing job, the JobTracker partitions the work and assign different map and reduce tasks to each TaskTracker in the cluster.

The following figure shows Topology of typical Hadoop cluster. This topology features a master node running the NameNode and JobTracker daemons and a standalone node with the SNN in case the master node fails. For small cluster, the SNN can reside on one of the slave nodes.
On the other hand, for large cluster, separate the NameNode and JobTracker on two machines. The slave machines each host a DataNode and TaskTracker, for running tasks on the same node where their data is stored.

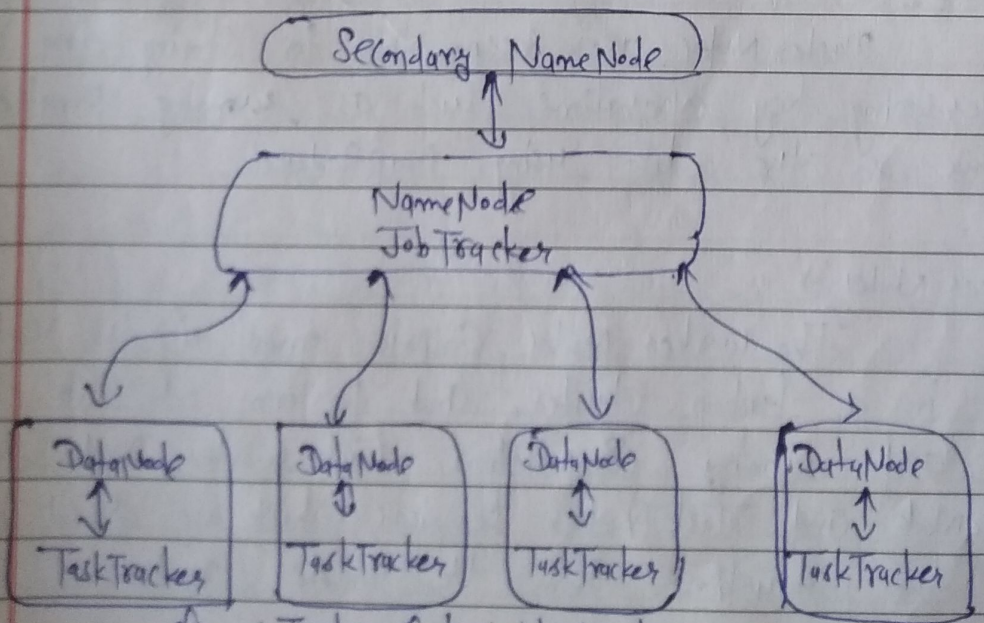


Figure: Topology of typical Hadoop cluster

* **Hadoop cluster :→**

- Apache Hadoop is an open sources, Java-based, software framework and parallel data processing Engine.
- It enables big data analytics processing tasks to be broken down into smaller tasks that can be performed in parallel by using an algorithm (like the MapReduce algorithm) and distributing them across a Hadoop cluster.
- A Hadoop cluster is a collection of computers, known as nodes, that are networked together to perform these kinds of parallel computations on big data sets. Unlike other computer cluster, Hadoop clusters are designed specifically to store and analyze mass amounts of structured and unstructured data in distributed computing environment.

* **Hadoop cluster Architecture :→**

Hadoop clusters are composed of a network of master and worker nodes that orchestrate and execute the various jobs across the Hadoop distributed file system.

(i) **Master Nodes :→**

Master Nodes are responsible for storing data in HDFS and overseeing key operations, such as running parallel computations on the data using MapReduce.

(ii) **~~The~~ Worker Nodes :→**

The worker Nodes comprise most of the virtual machines in a Hadoop cluster, and perform the job of storing the data and running computations. Each worker node runs the DataNode and TaskTracker services, which are used to receive the instructions from the master nodes.

(3) Client Nodes :→

Client Nodes are in charge of loading the data into the cluster. Client nodes first submit MapReduce jobs describing how data needs to be processed, and then fetch the results once the processing is finished.

Configuring Hadoop Cluster :→

- The Majority of Hadoop Settings are Contained in XML Configuration files.
- In order to create a Hadoop Cluster we need to Configure several XML files.
- All the Configuration files are stored in Conf directory of Hadoop_HOME.
- In hadoop-env.sh define the JAVA_HOME Environment Variable to point to the Java installation directory.
- Before Version 0.20, these XML files are hadoop-default.xml and hadoop-site.xml.
- The hadoop-default.xml Contains the default Hadoop Settings to be used unless they are explicitly overridden in Hadoop-site.xml.
- In Version 0.20 the Hadoop-site.xml file has been Seperated out into three XML files:- Core-site.xml, hdfs-site.xml and mapred-site.xml.
- A Hadoop Cluster Can be Configured in one of the following 3 modes by modifying the above XML files.

(i) Local (Standalone) Mode.
(ii) Pseudo-distributed mode.
(3) Fully-distributed mode.

} optional operational modes of Hadoop

(b) Local (standalone) mode :→

- The standalone mode is the default mode for Hadoop.
- When we first uncompressed the Hadoop source package, it does not consider our hardware setup. Hadoop chooses to be conservative and assumes a minimal configuration.
- All three XML files (or hadoop-site.xml before version 0.20) are empty under this default mode :

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!-- Put site-specific property overrides in this file. -->
<configuration>

</configuration>
```

- Its primary use is for developing and debugging the application logic of a MapReduce program without the additional complexity of interacting with the daemons.

(2) Pseudo-distributed mode :→

- The pseudo-distributed mode is running Hadoop in a "cluster of one" with all daemons running on a single machine.
- This mode allowing us to examine memory usage, HDFS I/O issues and other daemon interactions.
- Listing 2.1 provides simple XML files to configure a single server in this mode.

Listing 2.1 Example of the three configuration files for pseudo-distributed mode.

Core-site.xml

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="Configuration.xsl" ?>
<!-- put site-specific property overrides in this file. -->
<Configuration>
<property>
  <name> fs.default.name </name>
  <value> hdfs://localhost:9000 </value>
  <description> The name of the default file system. A URI whose
  scheme and authority determine the FileSystem implementation.
</description>
</property>
</Configuration>
```

mapred-site.xml →

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="Configuration.xsl"?>
<!-- put site-specific property overrides in this file. -->
<Configuration>
<property>
  <name> mapred.job.tracker </name>
  <value> localhost:9001 </value>
  <description> The host and port that the MapReduce job tracker runs
  at. </description>
</property>
</Configuration>
```

hdfs-site.xml :→

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!-- put site-specific property overrides in this file. -->
<Configuration>
<property>
  <name> dfs.replication </name>
  <value> 1 </value>
  <description> The actual number of replications can be specified
  when the file is created. </description>
</property>
</Configuration>
```

- In core-site.xml and mapred-site.xml we specify the hostname and port of the NameNode and the JobTracker, respectively.

Properties :→

(i) Configuration is required in given three files for this mode.
(ii) This is used for local code to test in HDFS.
(3) This is a cluster, where all daemons are running on one node itself.

(3) Fully-Distributed mode :→

After continually emphasizing the benefits of distributed storage and distributed computation, it's time for us to set up a full cluster. In the discussion below we'll use the following server names :-

- **Master :→**
  The master node of the cluster and host of the NameNode and Job - Tracker daemons.

- **Backup :→**
  The Server that hosts the Secondary NameNode daemon.

- **hadoop1, hadoop2, hadoop3, ... →**
  The slave boxes of the cluster running both DataNode and TaskTracker daemons.

Using the preceding naming convention, the below listing is a modified version of the pseudo-distributed configuration files that can be used as a skeleton for your cluster's setup.

* **Configuration XML files :-**

Core-Site.xml :→

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!-- put site-specific property overrides in this file. -->
<Configuration>
<property>
<name> fs. default. name </name>
<value> hdfs://master:9000 </value>
<description> The name of the default file system. A URI whose scheme
and authority determine the FileSystem implementation.
</description>
</property>
</Configuration>
```

mapred-site.xml :→

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl" ?>
<!-- Put site-specific property overrides in this file. -->
<configuration>
<property>
<name> mapred.job.tracker </name>
<value> master : 9001 </value>
<description> The host and port that the MapReduce job tracker
runs at. </description>
</property>
</configuration>
```

hdfs-site.xml :→

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl" ?>
<!-- Put site-specific property overrides in this file. -->
<configuration>
<property>
<name> dfs.replication </name>
<value> 3 </value>
<description> The actual number of replications can be specified
when the file is created. </description>
</property>
</configuration>
```

Properties :→

(1) This is a production phase.
(2) Data are used and distributed across many nodes.
(3) Different nodes will be used as master node / Data node etc.

* Data Measurement Chart :→

Data measurement	Data Size
Bit	1 or 0 (Single Binary Digit)
Byte	8 Bits
Kilobyte (KB)	1024 Bytes
Megabyte (MB)	1024 KB
Gigabyte (GB)	1024 MB
Terabyte (TB)	1024 GB
Petabyte (PB)	1024 TB
Exabyte (EB)	1024 PB
Zettabyte (ZB)	1024 EB
Yottabyte (YB)	1024 ZB

UNIT-2

* **Weather Dataset :→**

Weather Sensors Collecting data every hour at many locations across the globe gather a large volume of log data, which is a good candidate for analysis with MapReduce, since it is semi-structured and record-oriented.

Data format :→

The Data we will use is from the National Climatic Data Center. The Data is stored using a line-oriented ASCII format, in which each line is a record.

MapReduce :→

MapReduce is a Software framework and programming model used for processing huge amounts of data. MapReduce program work in two phases, - namely, Map and Reduce.

Map tasks deal with splitting and mapping of data while Reduce tasks shuffle and reduce the data.

Hadoop is capable of running MapReduce program written in various languages : Java, Ruby, Python and C++. The programs of MapReduce in cloud Computing are parallel in nature, thus are very useful for performing large-scale data analysis using multiple machines in the cluster. The input to each phase is key-value pairs. In addition, every programmer needs to specify two functions: Map function and Reduce function.

⇒ Classic MapReduce (MapReduce 1) →

A job run in classic MapReduce is illustrated in figure-1. At the highest level, there are four independent entities:

- The client, which submits the MapReduce job.
- The Jobtracker, which coordinates the job run. The Jobtracker is a Java application whose main class is JobTracker.
- The tasktrackers, which run the tasks that the job has been split into. Tasktracker are Java application whose main class is TaskTracker.
- The distributed filesystem (Normally HDFS), which is used for sharing job files between the other entities.

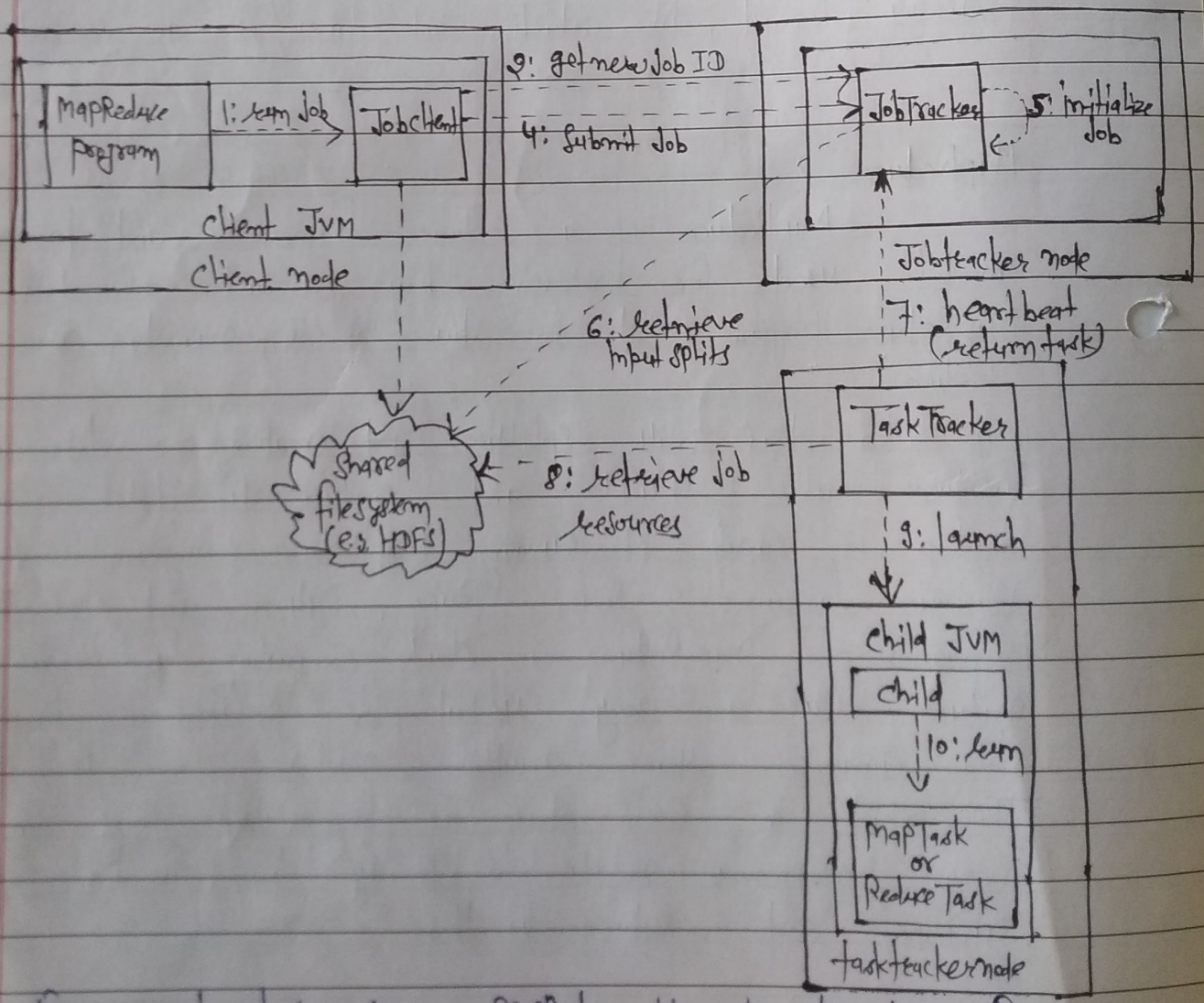


figure: 1 → How Hadoop runs a MapReduce job using the classic framework

⇒ **Job Submission :→**

The submit() method on Job creates an internal JobSubmitter instance and calls submitJobInternal() on it (Step 1) Having submitted the Job, waitForCompletion() polls the Job's progress once a second and reports the progress to the Console if it has changed since the last report.

The Job submission process implemented by JobSubmitter does the following:

- Asks the Jobtracker for a new Job ID (by calling getNewJobId() on JobTracker) (Step 2)
- Checks the output specification of the Job. for example, if the output directory has not been specified or it already exists, the Job is not submitted and an error is thrown to the MapReduce program.
- Computes the input splits for the Job. If the splits cannot be computed, because the input paths don't exist, for example, then the Job is not submitted and an error is thrown to the MapReduce program.
- Copies the resources needed to run the Job, including the Job JAR file, the configuration file, and the computed input splits, to the Jobtracker's file system in a directory named after the Job ID. The Job JAR is copied with a high replication factor (controlled by the mapred.submit. replication property, which defaults to 10) so that there are lots of copies across the cluster for the tasktrackers to access when they run tasks for the Job. (Step 3)
- Tells the Jobtracker that the Job is ready for execution (by calling submitJob() on JobTracker) (Step 4)

⇒ Job Initialization :→

When the JobTracker receives a call to its submitJob() method, it puts it into an internal queue from where the Job Scheduler will pick it up and initialize it. (Step 5)

To create the list of tasks to run, the Job Scheduler first retrieves the input splits computed by the client from the shared filesystem (Step 6). It then creates one map task for each split. The number of reduce tasks to create is determined by the mapred.reduce.tasks property in the Job, which is set by the setNumReduceTasks() method, and the Scheduler simply creates this number of reduce tasks to be run.

⇒ Task Assignment :→

Tasktrackers run a simple loop that periodically sends heartbeat method calls to the Jobtracker. Heartbeats tell the Jobtracker that a tasktracker is alive. As a part of the heartbeat, a tasktracker will indicate whether it is ready to run a new task, and if it is, the Jobtracker will allocate it a task, which it communicates to the tasktracker using the heartbeat return value (Step 7).

Tasktracker have a fixed number of slots for map tasks and for reduce tasks: for example, a tasktracker may be able to run two map tasks and two reduce tasks simultaneously. (The precise number depends on the number of cores and the amount of memory on the tasktracker.)

Task Execution now that the tasktracker has been assigned a task, the next step is for it to run the task. First, it localizes the Job JAR by copying it from the shared filesystem to the tasktracker's filesystem.

It also copies any files needed from the distributed cache by the application to the local disk. Then, it creates a local working directory for the task, and un-jars the contents of the JAR into this directory. Third it creates an instance of TaskRunner to run the task. Task Runner launches a new Java Virtual Machine (step 9) to run each task in (step 10), so that any bugs in the user-defined map and reduce function don't affect the tasktracker.

⇒ **Progress and Status Updates :→**

MapReduce jobs are long-running batch jobs, taking anything from minutes to hours to run. Because this is a significant length of time, it's important for the user to get feedback on how the job is progressing. A job and each of its tasks have a status, which includes such things as the state of the job or task (e.g. running, successfully completed, failed).

⇒ **Job Completion :→**

When the jobtracker receives a notification that the last task for a job is complete (this will be the special job cleanup task), it changes the status for the job to "successful". Then, when the job polls for status, it learns that the job has completed successfully, so it prints a message to tell the user and then returns from the waitForCompletion() method.

* Understanding Hadoop API for MapReduce framework (old and new) →

Hadoop provides two Java MapReduce APIs named as old and new respectively.
There are several notable ~~different~~ differences between the two APIs.

1. The new API favors abstract classes over interfaces, since these are easier to evolve. for example, you can add a method (with a default implementation) to an abstract class without breaking old implementations of the class. for example, the Mapper and Reducer interface in the old API are abstract classes in the new API.

2. The new API is in the org.apache.hadoop.mapreduce package (and subpackages).
The old API can still be found in org.apache.hadoop.mapred.

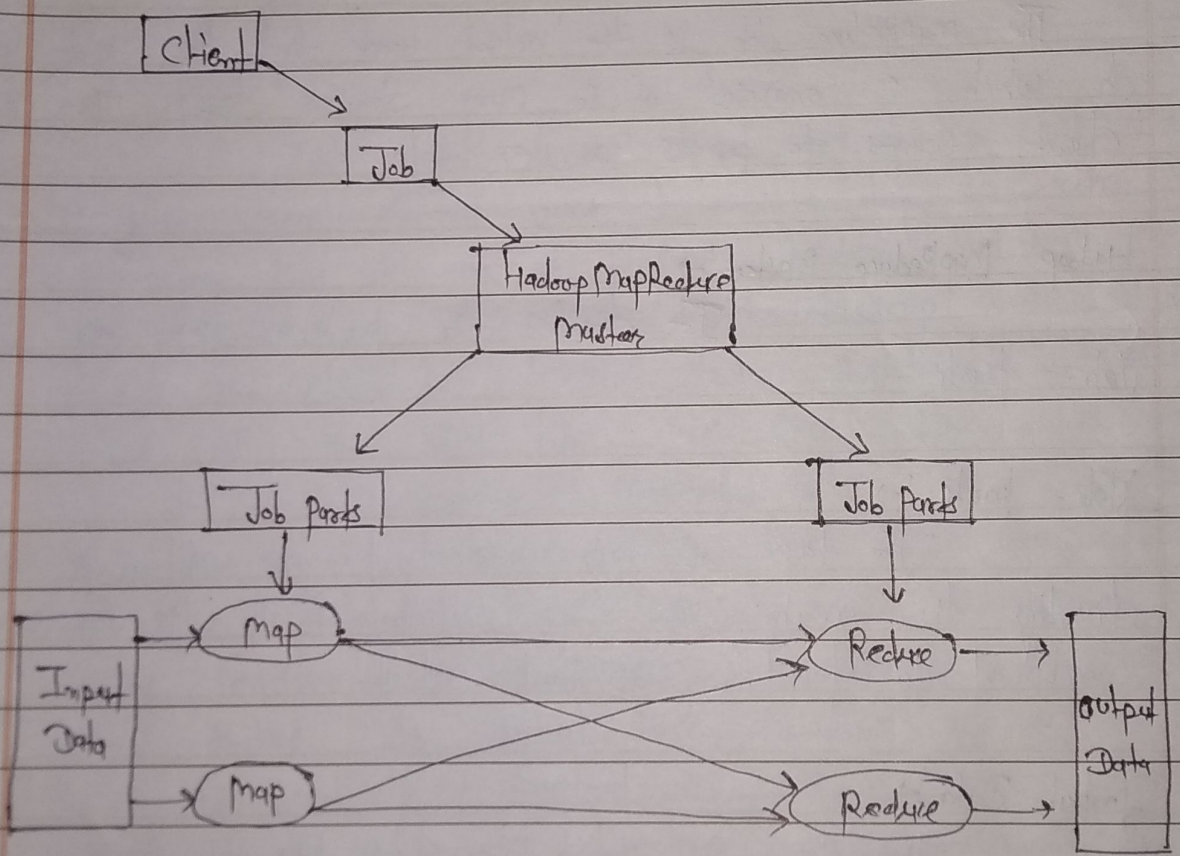
3. The New API makes extensive use of Context, for example essentially unifies the role of the JobConf, the OutputCollector and the Reporter from the old API.

4. In both APIs, key-value record pairs are pushed to the mapper and reducer, but in addition, the new API allows both mappers and reducers to control the execution flow by overriding the run() method.
In the old API this is possible for mappers by writing a MapRunnable, but no equivalent exists for reducers.

5. Configuration has been unified. The old API has a special JobConf object for Job Configuration. In the new API, this distinction is dropped, so Job Configuration is done

* MapReduce Architecture :→

MapReduce is a programming model used for efficient processing in parallel over large data-sets in a distributed manner. The data is first split and then Combined to produce the final result. The libraries for MapReduce is written in so many programming language with Various different - different optimizations.



* Components of MapReduce Architecture :→

1. Client :→
The MapReduce client is the one who brings the Job to the MapReduce for processing. There can be multiple clients available that continuously send jobs for processing to the Hadoop MapReduce Manager.

2. Job :→
The MapReduce Job is the actual work that the client wanted to do which is comprised of so many smaller tasks that the client wants to process or execute.

3. Hadoop MapReduce Master :→
It divides the particular job into subsequent Job-parts.

4. Job-parts :→
The task or sub-jobs that are obtained after dividing the main job. The result of all the Job-parts Combined to produce the final output.

5. Input Data :→
The data set that is fed to the MapReduce for processing.

6. Output Data :→
The final result is obtained after the processing.

through Configuration.

6. Job Control is performed through the Job class in the new API, rather than the old JobClient, which no longer exists in the new API.

* Basic Programs of Hadoop MapReduce :→

→ Driver Code :→

A Job object forms the specification of the job. It gives you control over how the job is run. When we run this job on a Hadoop cluster, we will package the code into a JAR file (which Hadoop will distribute around the cluster). Rather than explicitly specify the name of the JAR file, we can pass a class in the Job's setJarByClass() method, which Hadoop will use to locate the relevant JAR file by looking for the JAR file containing this class.

Having constructed a Job object, we specify the input and output paths. An input path is specified by calling the static addInputPath() method on FileInputFormat, and it can be a single file, a directory (in which case, the input forms all the files in that directory), or a file pattern.

The output path (of which there is only one) is specified by the static setOutputPath() method on FileOutputFormat. It specifies a directory where the output files from the reducer functions are written.

The driver code for weather program is specified below:

```
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
public class MaxTemperature
{
public static void main (String[] args) throws Exception
{
if (args.length != 2)
{
System.err.println("Usage: MaxTemperature <input path> <output
        path>");
System.exit(-1);
}
Job job = new Job();
job.setJarByClass(MaxTemperature.class);
job.setJobName("Max temperature");
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
job.setMapperClass(MaxTemperatureMapper.class);
job.setReducerClass(MaxTemperatureReducer.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
System.exit(job.waitForCompletion(true) ? 0 : 1);
}
}
```

* **Mapper Code :→**

The Mapper class is a generic type, with four formal type parameters that specify the input key, input value, output key, and output value types of the map function. For the present example, the input key is a long integer offset, the input value is a line of text, the output key is a year, and the output value is an air temperature (an integer). The following example shows the implementation of our map method.

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
public class MaxTemperatureMapper
extends Mapper<LongWritable, Text, Text, IntWritable>
{
private static final int MISSING = 9999;
@Override
public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException
{
String line = value.toString();
String year = line.substring(15, 19);
int airTemperature;
if (line.charAt(87) == '+')
{
airTemperature = Integer.parseInt(line.substring(88, 92));
}
```

~~ELSE~~

Else

{

airTemperature = Integer.parseInt(line.substring(87, 92));

}

String quality = line.substring(92, 93);

if(airTemperature != MISSING && quality.matches("[01459]"))

{

context.write(new Text(year), new IntWritable(airTemperature));

}

}

The map() method is passed a key and a value. We convert the Text value containing the line of input into a Java string, then use its substring() method to extract the columns we are interested in.

The map() method also provides an instance of Context to write the output to. In this case, we write the year as a Text object (since we are just using it as a key), and the temperature is wrapped in an IntWritable. We write an output record only if the temperature is present and the quality code ~~indicates~~ indicates the temperature reading is ok.

* Reducer Code :→

Again, four formal type parameters are used to specify the input and output types, this time for the reduce function. The input types of the reduce function must match the output types of the map function: Text and IntWritable. And in this case, the output types of the reduce function are Text and IntWritable, for a year and its maximum temperature, which we find by iterating through the temperatures and comparing each with a record of the highest found so far.

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
public class MaxTemperatureReducer
extends Reducer<Text, IntWritable, Text, IntWritable>
{
@Override
public void reduce(Text key, Iterable<IntWritable> values, Context context)
throws IOException, InterruptedException
{
int maxValue = Integer.MIN_VALUE;
for(IntWritable value : values)
{
maxValue = Math.max(maxValue, value.get());
}
context.write(key, new IntWritable(maxValue));
}
}
```

* RecordReader :→

RecordReader is responsible for creating key/value pair which has been fed to map task to process. Each InputFormat has to provide its own RecordReader implementation to generate key/value pairs. For example, the default TextInputFormat provides LineRecordReader which generates byte offset of the file as key and \n seperated line in the input file as value.

* Combiner Code :→

Many MapReduce jobs are limited by the bandwidth available on the cluster, so it pays to minimize the data transferred between map and reduce tasks. Hadoop allows the user to specify a Combiner function to be run on the map output - the Combiner function's output forms the input to the reduce function. Since the Combiner function is an optimization Hadoop does not provide a guarantee of how many times it will call it for a particular map output record, if at all. In other words, calling the Combiner function zero, one, or many times should produce the same output from the reducer. The Combiner function doesn't replace the reduce function. But it can help cut down the amount of data shuffled between the maps and reduces.

```
public class MaxTemperatureWithCombiner
{
  public static void main (String[] args) throws Exception
  {
    if (args.length != 2)
    {
```

```
System.err.println("Usage: MaxTemperatureWithCombiner <input path>"
        +"<output path>");
System.exit(-1);
}

Job job = new Job();
job.setJarByClass(MaxTemperatureWithCombiner.class);
job.setJobName("max temperature");
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
job.setMapperClass(MaxTemperatureMapper.class);
job.setCombinerClass(MaxTemperatureReducer.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
System.exit(job.waitForCompletion(true) ? 0:1);
}
}
```

* **Partitioner Code :→**

The Partitioning phase takes place after the map phase and before the reduce phase. The number of partitions is equal to the number of reducers. The data gets partitioned across the reducers according to the partitioning function. The difference between a partitioner and a Combiner is that the partitioner divides the data according to the number of reducers so that all the data in a single partition gets executed by a single reducer. However, the Combiner functions similar to the reducer and processes

the data in each partition. The combiner is an optimization to the reducer. The default partitioning function is the hash partitioning function where the hashing is done on the key. However it might be useful to partition the data according to some other function of the key or the value.

UNIT-3

* Writable Interface :→

Writable is interface mechanism to Serialise and de-Serialise your data. It is an interface which has write method and read fields method.

- **Write field :→**

  Write field is to writing your data into the output Steeam or network.

- **Read field :→**

  Read field is to read data from the input Steeam.

The Hadoop is used for MapReduce Computations. It uses the Writable interface based classes as the data types. These data types from writable are used throughout the MapReduce Data flow Structure. It Starts from reading input data, transferring intermediate data between Map & Reduce and then writing output data.

* Writable Interface functions :→

- A data type must implements the org.apache.hadoop.io.Writable interface in order to be used as a Value data type of a MapReduce Computation.
- It is only one interface will define how a Value should be Serialized and de-Serialized in Hadoop for transmitting and Storing the data.

Coding :-

```
package org.apache.hadoop.io;
import java.io.DataInput;
import java.io.DataOutput;
public interface Writable
{
Void write(DataOutput out) throws IOException;
Void readFields(DataInput in) throws IOException;
}
```

* Writable Comparable and Comparators :→

IntWritable implements the ~~int~~ WritableComparable interface, which is just a subinterface of the Writable and java.lang.Comparable interface.

~~java.lang.Comparable<int>~~

```
Package org.apache.hadoop.io;
public interface WritableComparable <T> extends Writable, Comparable <T>
{
}
```

Comparison of types is crucial for MapReduce, where there is a sorting phase during which keys are compared with one another. One optimization that Hadoop provides is the RawComparator extension of Java's Comparator.

```
package org.apache.hadoop.io;
import java.util.Comparator;
public interface RawComparator <T> extends Comparator <T>
{
public int compare(byte [] b1, int s1, int l1, byte [] b2, int s2, int l2)
}
```

In the above example, the Comparator for IntWritables implements the raw compare() method by reading an integer from each of the byte arrays b1 and b2 and comparing them directly, from the given start positions (s1 and s2) and length (l1 and l2).

Difference between Writable Comparable and Writable Comparator :→

	Writable Comparable	Writable Comparator
1.	Writable Comparables can be compared to each other, typically via Comparators. Any type which is to be used as a key in the Hadoop Map-Reduce framework should implement this interface.	A Comparator for Writable Comparables. This base implementation uses the natural ordering. To define alternate orderings, override compare (WritableComparable, WritableComparable).
2.	org.apache.hadoop.io.WritableComparable	org.apache.hadoop.io.WritableComparator

For implementing a WritableComparable we must have CompareTo method apart from readFields and write methods,	A Comparator that operates directly on byte representations of object. Compare(byte[] b1, int s1, int l1, byte[] b2, int l2) Compare two objects in binary. b1[s1:l1] is the first object and b2[s2:l2] is the second object.

* **Writable Classes :→**

Hadoop comes with a large selection of writable classes in the org.apache.hadoop.io package.

Hadoop provides classes that wrap the Java primitive types and implement the WritableComparable and Writable Interfaces. They are provided in the org.apache.hadoop.io package.

They form the class hierarchy shown in figure:-

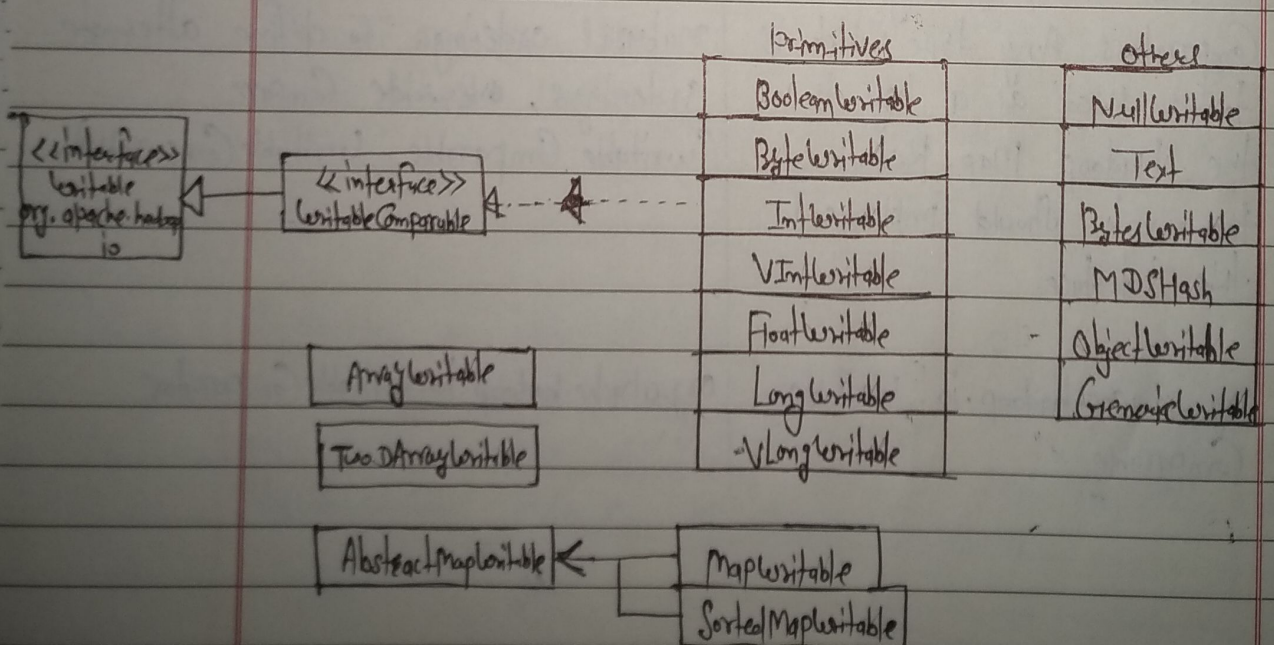


figure :- Writable class hierarchy.

Name of Lecturer : .....

⇒ **Writable Wrappers for Java Primitives :→**

There are Writable wrappers for all the Java primitive types except char (which can be stored in an IntWritable) as shown in table! All have a get() and a set() method for retrieving and storing the wrapped value.

Java Primitive	Writable implementation	Serialized size (bytes)
boolean	BooleanWritable	1
byte	ByteWritable	1
short	ShortWritable	2
int	IntWritable	4
	VIntWritable	1-5
float	FloatWritable	4
long	LongWritable	8
	VLongWritable	1-9
double	DoubleWritable	8

Table 1 :- Writable wrapper classes for Java primitives

When encoding integers, there is a choice between the fixed-length formats (IntWritable and LongWritable) and the variable-length formats (VIntWritable and VLongWritable). The variable-length formats use only a single byte to encode the value if it is small enough (between -112 and 127, inclusive); otherwise, they use the first byte to indicate whether the value is positive or negative. ~~and~~

And 163 ~~require~~ requires two bytes.

⇒ Text :→

Text is a writable for UTF-8 sequences. It can be thought of as the writable equivalent of Java.lang.string. The ~~text~~ Text class uses an int (with a variable-length encoding) to store the number of bytes in the string encoding, so the maximum value is 2 GB.

- Indexing :-

Because of its emphasis on using standard UTF-8, ~~there are some~~

- Unicode :→

when we start using characters that are encoded with more than a single byte. Consider the unicode character shown in table 2.

Unicode code point	U+0041	U+00DF	U+6771	U+10400
Name	LATIN CAPITAL LETTER A	LATIN SMALL LETTER SHARP S	N/A (a unified Han ideograph)	DESERET CAPITAL LETTER LONG I
UTF-8 code units	41	C3 9F	E6 9D B1	F0 90 90 80
Java representation	\u0041	\u00DF	\u6771	\uD801\uDC00

Table-2 :- Unicode characters.

- Iteration :→

Iteration over the Unicode characters in Text is complicated by the use of byte offsets for indexing, since you can't just increment the index.

⇒ **BytesWritable :→**

BytesWritable is a wrapper for an array of binary data. Its Serialized format is an interface field (4 bytes) that specifies the number of bytes to follow, followed by the bytes themselves. for example :- the byte array of length two with value 3 and 5 is serialized as - a 4-bite integer (00000002) followed by the two bytes from the array (03 and 05).

```
BytesWritable b = new BytesWritable(new byte[] {3, 5});
byte[] bytes = Serialize(b);
assertThat(StringUtils.byteToHexString(bytes), is("000000020305"));
```

BytesWritable is mutable, and its value may be changed by calling its set() method.

⇒ **NullWritable :→**

NullWritable is a special type of Writable, as it has a zero-length serialization. No bytes are written to, or read from, the stream. It is used as a placeholder; for example → in MapReduce, a key or a value can be declared as a NullWritable when you don't need to use that position - it effectively stores a constant empty value. NullWritable can also be useful as a key in SequenceFile when we want to store a list of values, as opposed to key-value pairs.

⇒ **ObjectWritable :→**

ObjectWritable is a general-purpose wrapper for the following: Java primitives, String, Enum, Writable, null, or arrays of any of these types.

⇒ **GenericWritable :→**

GenericWritable is useful when a field can be of more than one type:
for example → if the values in a SequenceFile have multiple types, then we can declare the value type as an GenericWritable and wrap each type in an GenericWritable.

* **Writable Collections :→**

There are six Writable Collection types in the org.apache.hadoop.io package.
ArrayWritable, ArrayPrimitiveWritable, TwoDArrayWritable, MapWritable, SortedMapWritable and EnumSetWritable.

- ArrayWritable and TwoDArrayWritable are Writable implementations for arrays and Two-dimensional Arrays (array of arrays) of Writable instances. All the elements of an ArrayWritable or a TwoDArrayWritable must be instances of the same class, which is specified at construction, as follows:-

```
ArrayWritable writable = new ArrayWritable (Text.class);
```

ArrayWritable and TwoDArrayWritable both have get() and set() methods as well as a toArray() method, which creates a shallow copy of the array.

- ArrayPrimitiveWritable is a wrapper for arrays of Java primitives. The component type is detected when you call set(), so there is no need to subclass to set the type.

- MapWritable and SortedMapWritable are implementations of java.util.Map<Writable, Writable> and java.util.SortedMap<WritableComparable, Writable>, respectively. Here's demonstration of using a MapWritable with different types for keys and values.

* Implementing a Custom Writable :→

Hadoop comes with a useful set of Writable implementations that serve most purposes; you may need to write your own custom implementation. And with a custom Writable we have full control over the binary representation and the sort order.

Because Writable are at the heart of the MapReduce data path, tuning the binary representation can have a significant effect on performance. To demonstrate how to create a custom Writable, we shall write an implementation that represents a pair of strings, called TextPair.

* Implementing a RawComparator for Speed :→

Implementing the org.apache.hadoop.io.RawComparator interface will definitely help speed up your Map/Reduce Jobs. As you may recall, a MR (Map/Reduce) Job is composed of receiving and sending key-value pairs. The process look like the following:

(k1, v1) → Map → (k2, v2)
(k2, List[v2]) → ~~Recc~~ Reduce → (k3, v3)

The key-value pairs (k2, v2) are called the intermediary key-value pairs. They are ~~for~~ passed from the mapper to the reducer. Before these ~~intercepen intermi~~ intermediary key-value pairs reach the reducer, a Shuffle and Sort step is performed. The Shuffle is the assignment of the intermediary keys (k2) to reducers and the Sort is the sorting of these keys.

Two ways you may Compare your keys is by implementing the org.apache.hadoop.io.WritableComparable interface or By implementing the RawComparator interface.
In the former approach, you will Compare objects, but in the latter approach, you will Compare the keys using their Corresponding raw bytes.

Name of Lecturer : .....

* Custom Comparators :→

As we can see with TextPair, writing raw comparators takes some care, since we have to deal with details at the byte level.

Custom Comparators should also be written to be RawComparators, if possible. These are comparators that implement a different sort order to the natural sort order defined by the default Comparator.

Note we override the compare() method that takes objects so both compare() methods have the same semantics.

UNIT - 4

* Apache Pig :→

Apache pig is an abstraction over MapReduce. It is a tool/platform which is used to analyze larger sets of data representing them as data flows. Pig is generally used with Hadoop. We can perform all the data manipulation operations in Hadoop using Apache pig.
To write data analysis programs, pig provides a high-level language known as Pig Latin. This language provides various operators using which programmers can develop their own functions for reading, writing and processing data.

* Features of Pig :→

1. Rich set of Operators :→

It provides many operators to perform operations like Join, Sort, filer etc.

2. Easy of Programming :→

Pig Latin is similar to SQL and it is easy to write a pig script if we are good at SQL.

3. Optimization opportunities :→

The tasks in Apache pig optimize their execution automatically, so the programmers need to focus only on semantics of the language.

4. Extensibility :→

Using the existing operators, users can develop their own functions to read, process and write data.

5. Handles all kinds of data :→

Apache pig analyzes all kinds of data, both structured as well as unstructured. It stores the results in HDFS.

* Admiring the Pig Architecture :→

Pig is made up of two (count 'em, two) components :

1. The language itself :→

As proof that programmers have a sense of humor, the programming language for Pig is known as Pig Latin, a high-level language that allows you to write data processing and analysis programs.

2. The Pig Latin Compiler :→

The Pig Latin Compiler converts the Pig Latin code into executable code. The executable code is either in the form of MapReduce jobs or it can spawn a process where a virtual Hadoop instance is created to run the Pig code on a single node.

The sequence of MapReduce program enable Pig programs to do data processing and analysis in parallel, leveraging Hadoop MapReduce and HDFS. Running the Pig job in virtual Hadoop instance is a useful strategy for testing ~~out~~ our Pig scripts.

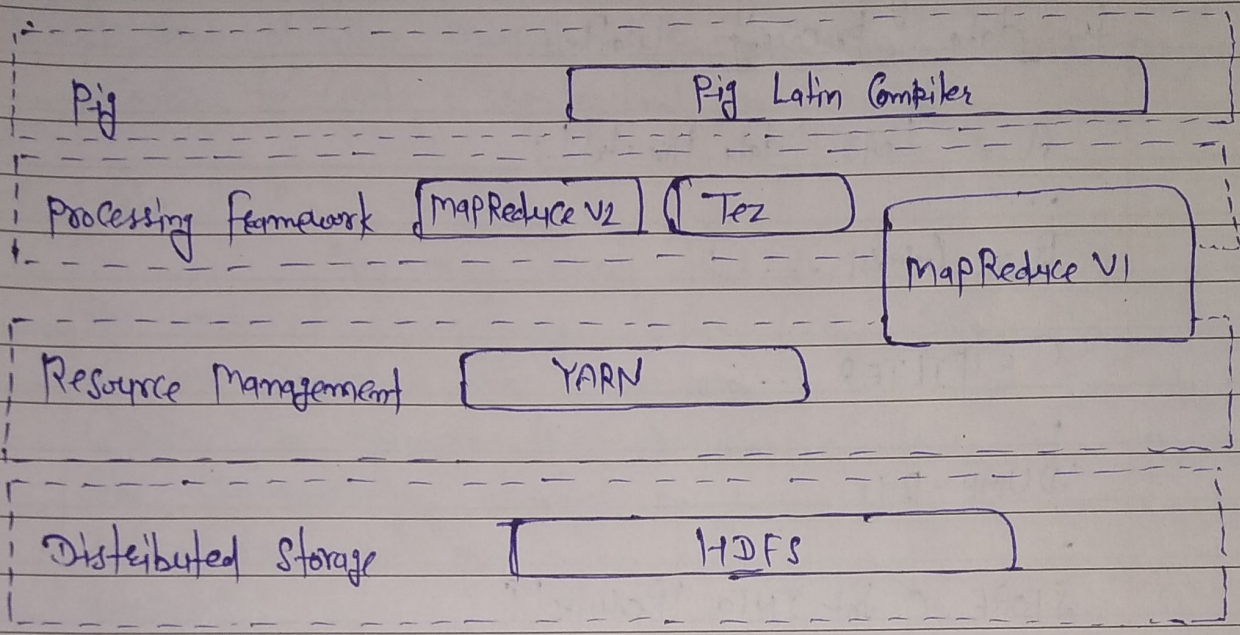


Figure 1: ~~Pig~~ Pig Architecture

Pig Program can run on MapReduce 1 or MapReduce 2 without any code changes, regardless of what mode your cluster is running. However, Pig scripts can also run using the Tez API instead.

* Going with the Pig Latin Application flow :→

At its core, Pig Latin is a dataflow language, where ~~you~~ we define a data stream and a series of transformations that are applied to the data as it flows through our application. This is in contrast to a control flow language (like C or Java), where we write a series of instructions. In control flow language, we use constructs like loop and conditional logic (like an if statement). We won't find loops and if statements in Pig Latin.

If we need some convincing that working with Pig is a significantly easier than having to write Map and Reduce programs, start by taking a look at some real Pig syntax.
The following Listing specifies sample Pig code to illustrate the data processing dataflow.

```
A = LOAD 'data_file.txt';
...
B = GROUP ... ;
C = FILTER ... ;
...
DUMP B;
..
STORE C INTO 'Results';
```

* Working through the ABCs of Pig Latin :→

Pig Latin is the language for Pig programs. Pig translates the Pig Latin script into MapReduce jobs that can be executed within Hadoop cluster. When coming up with Pig Latin, the development team followed three key design principles :-

(1) Keep it simple :→

Pig latin provides a streamlined method for interacting with Java MapReduce. It's an abstraction, in other words, that simplifies the creation of parallel programs on the Hadoop cluster for data flows and analysis.
Complex tasks may require a series of interrelated data transformations - such series are encoded as data flow sequences.

writing data transformation and flows as pig Latin scripts instead of Java MapReduce programs make these programs easier to write, understand and maintain.

(2) Make it Smart :→

You may recall that the pig Latin Compiler does the work of transforming a pig Latin program into a series of Java MapReduce jobs. The trick is to make sure that the Compiler can optimize the execution of these Java MapReduce jobs automatically, allowing the user to focus on semantics rather than on how to optimize and access the dataset.

(3) Don't limit development :→

make ~~up~~ pig extensible so that developers can add functions to address their particular business problems.

* Evaluating Local and Distributed modes of Running Pig Scripts :→

Before you can run your first pig script, you need to have a handle on how pig programs can be packaged with the pig Server.

Pig has two modes for running script, as shown in figure :-

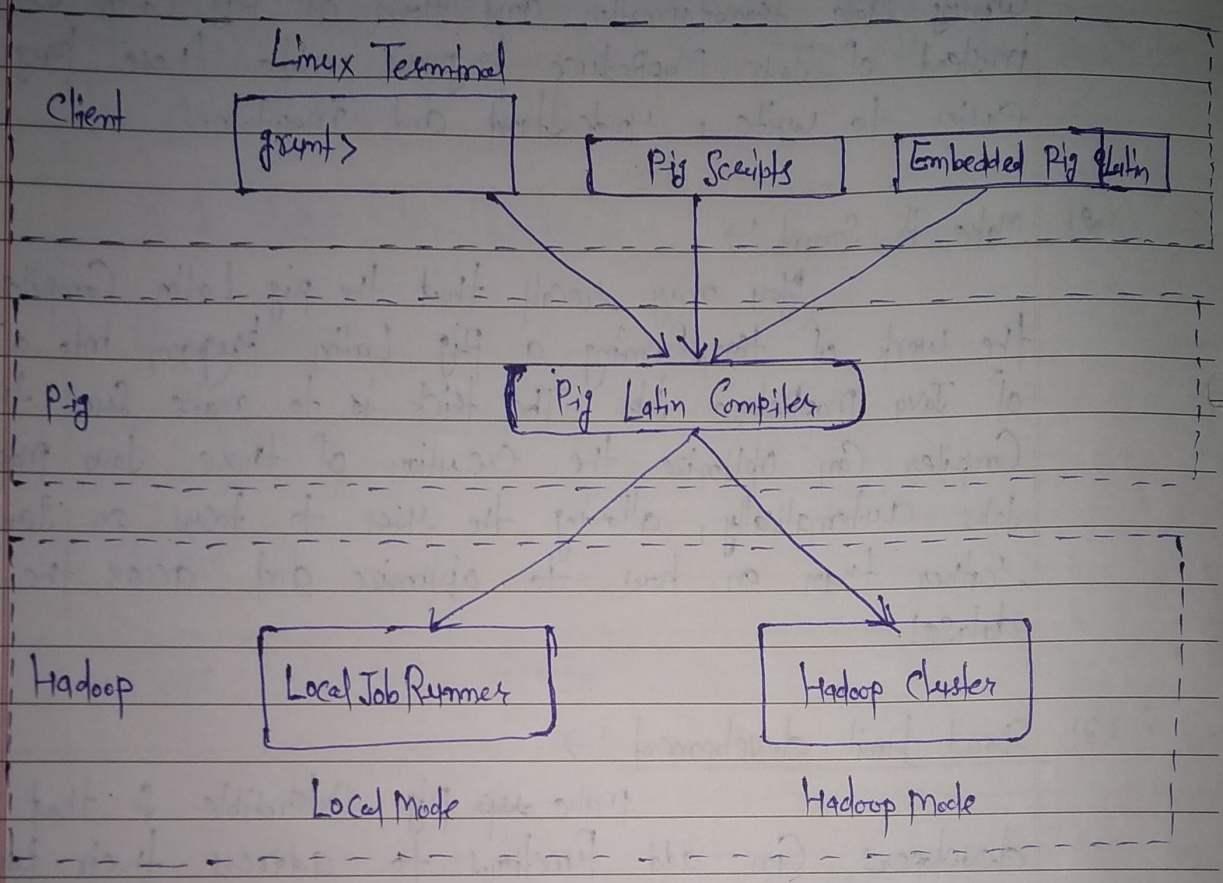


figure : Pig Modes

Local mode :→

All Scripts are run on a Single machine without requiring Hadoop MapReduce and HDFS. This can be useful for developing and testing Pig logic. If you're using a Small Set of data to develope or test your Code, then local mode could be faster then going through the MapReduce infrastructure.

Local mode doesn't require Hadoop. When you run in Local mode, the pig program runs in the Context of a local Java Virtual machine, and data access is via the local file System of a Single machine. Local mode is actually

a local simulation of MapReduce in Hadoop's LocalJobRunner class.

MapReduce Mode / Hadoop mode :→

MapReduce mode is also known as Hadoop Mode. Pig is executed on the Hadoop cluster. In this case, the pig script gets converted into a series of MapReduce jobs that are then run on the Hadoop cluster. If you have a terabyte of data that you want to perform operations on and you want to interactively develop a program, you may soon find things slowing down considerably and you may start growing your storage.

* Checking out the pig script Interfaces :→

The pig programming language is designed to handle any kind of data tossed its ways - structured, semi-structured, unstructured data, pig programs can be packaged in three different ways :-

(1) Script :→

This method is nothing more than a file containing pig Latin commands, identified by the pig suffix (FlightData.pig). Ending your pig program with the .pig extension is a convention but not required. The commands are interpreted by the pig Latin compiler and executed in the order determined by the pig optimizer.

(2) Grunt :→

Grunt acts as a command interpreter where we can interactively enter pig Latin at the Grunt command line and immediately see the response. This method is helpful for prototyping during initial development and with what-if scenarios.

(3) Embedded :→

Pig Latin statements can be executed within Java, Python or JavaScript programs.

Pig scripts, Grunt shell pig commands, and embedded pig programs can run in either Local mode or MapReduce mode.
The Grunt shell provides an interactive shell to submit pig commands or run pig scripts.
To start the Grunt shell in Interactive mode, just submit the command pig at your shell. To specify whether a script or Grunt shell is executed locally or in Hadoop mode just specify it in the -x flag to the pig command.
The following is an example of how you'd specify running your pig script in local mode.

- Pig -x local milesPerCarrier.pig

Here's how you'd run the pig script in Hadoop mode, which is the default if you don't specify the flag.

- Pig -x mapreduce milesPerCarrier.Pig

By default, when we specify the pig command ~~without~~ (without any parameters, it starts the Grunt shell in Hadoop mode. If we want to start the Grunt shell in local mode just add the -x local flag to the command.

Here is an example :-

pig -x local

* Scripting with Pig Latin :→

Hadoop is a rich and quickly evolving ecosystem with a growing set of new applications. Rather than try to keep up with all the requirements for new capabilities, Pig is designed to be extensible via user-defined functions, also known as ~~UDF~~ UDFs. UDFs can be written in a number of programming languages, including Java, Python and Javascript. Developers are also posting and sharing a growing collection of UDFs online.

(Look for ~~Piggy~~ Piggy Bank and DataFu, to name just two examples of such online collections)

Some of the Pig UDFs that are part of these repositories are LOAD/STORE function (XML, for example), date time, functions, text, math, and stats functions.

Pig can also be embedded in host languages such as Java, Python and Javascript, which allows you to integrate Pig with your existing applications.

* HDFS Architecture :→

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

HDFS Architecture.

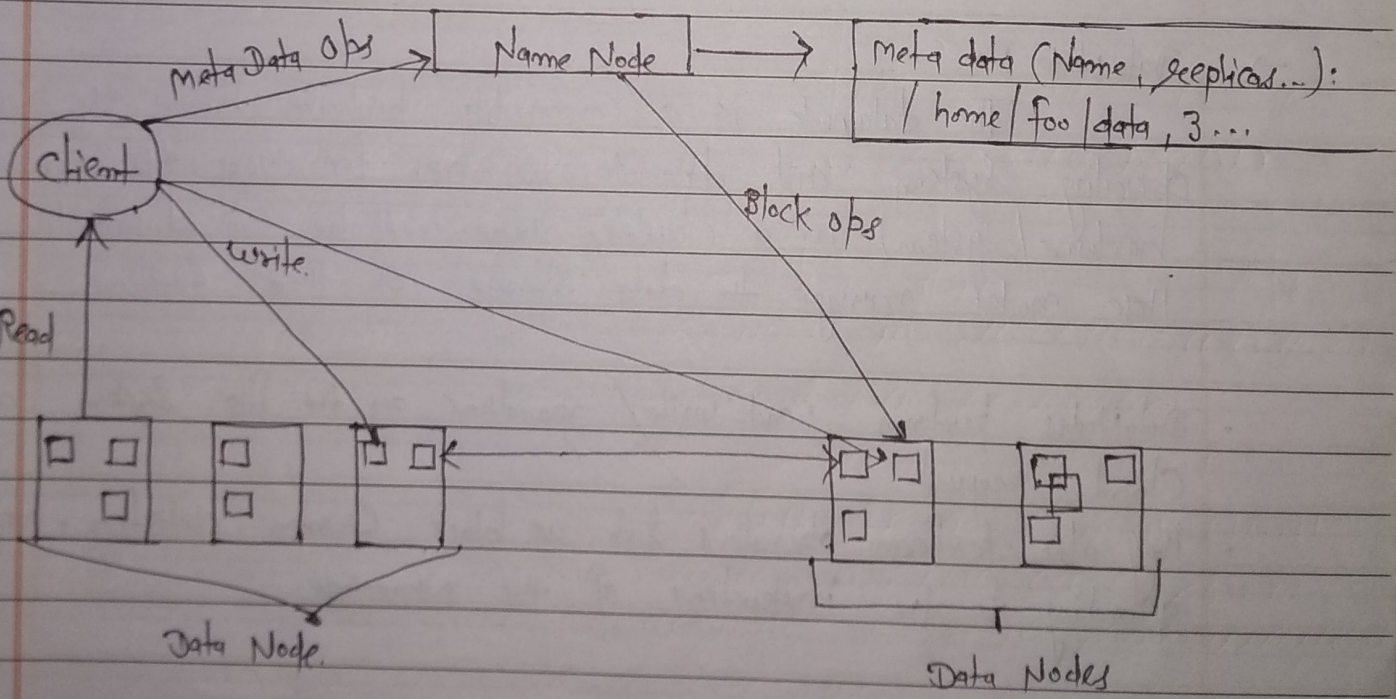


figure : HDFS Architecture.

→ NameNode :→

The NameNode is the Commodity hardware that Contains the GNU/LINUX Operating System and the NameNode Software. It is a Software that can be run on Commodity hardware. The System having the NameNode acts as the master Server and it does the following tasks –

- Manages the files
- Store Meta Data
- It also executes file System operations such as renaming, closing, and opening files and directories.

→ DataNode :→

The datanode is a Commodity hardware having the GNU/Linux Operating System and dataNode Software. for every Node (Commodity hardware/System) in a Cluster, there will be a dataNode. These nodes manage the data storage of their System.

- DataNodes perform read-write operations on the file System, as per Client request.
- They also perform operations such as block Creation, deletion, and replication according to the instructions of the NameNode.

→ Block :→

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

UNIT-5

* Hive :→

Hive is a data warehouse system which is used to analyze structured data. It is built on the top of Hadoop. It was developed by facebook.
Hive provides the functionality of reading, writing, and managing large datasets residing in distributed storage. It runs SQL like queries called HQL (Hive Query Language). which gets internally converted to MapReduce Jobs.
Using Hive, we can skip the requirement of the traditional approach of writing complex MapReduce programs.
Hive ~~supp~~ supports Data Definition Language (DDL), Data Manipulation Language (DML) and User Defined functions (UDF).

* features of Hive :→

There are the following features of Hive :-

1. Hive is fast and scalable.
2. It provides SQL-like queries (i.e. HQL) that are implicitly transformed to MapReduce or Spark jobs.
3. It is capable of analyzing large datasets stored in HDFS.
4. It allows different storage types such as plain text, RCFile and HBase.
   ↳ (Record Columnar file)
5. It uses indexing to accelerate queries.
6. It can operate on compressed data stored in the Hadoop ecosystem.

7. It supports user-defined functions (UDFs) where user can provide its functionality.

* Limitations of Hive :→

1. Hive is not capable of handling real-time data.
2. It is not designed for online transaction processing.
3. Hive queries contain high latency.

* Difference between Hive and Pig :→

	Hive	Pig
1.	Hive is commonly used by Data Analysts.	Pig is commonly used by programmers.
2.	It follows SQL-like queries	It follows the data-flow language.
3.	It can handle structured data.	It can handle semi-structured data.
4.	It works on server-side of HDFS cluster.	It works on client-side of HDFS cluster.
5.	Hive is slower than Pig.	Pig is comparatively faster than Hive.

Name of Lecturer : .....

* Modes of Hive :→

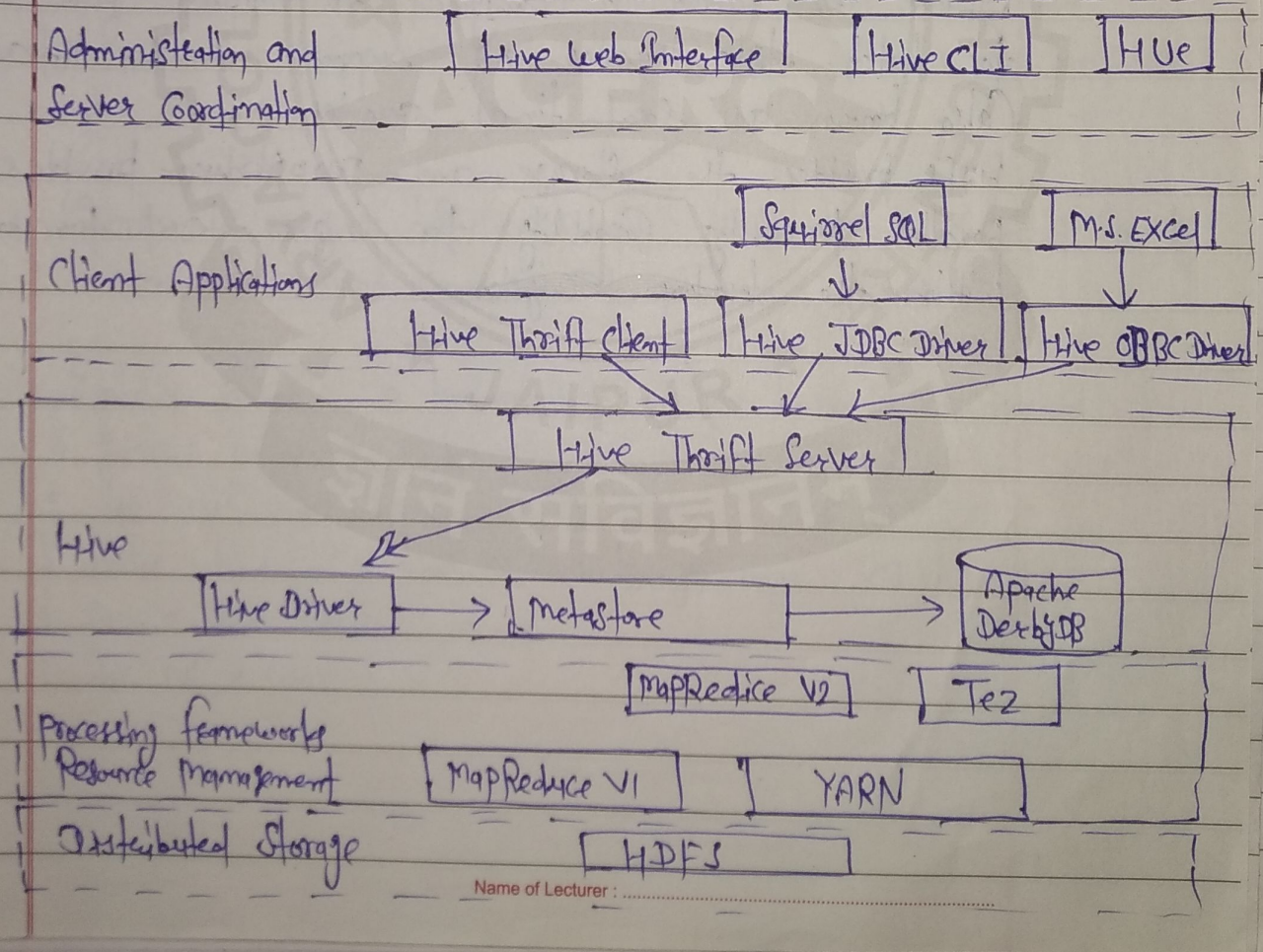
Hive can operate in two modes depending on the size of data Nodes in Hadoop.

These modes are :-

1. Local mode
2. Map reduce mode

* ~~How~~ Hive is put Together :→

In this we illustrate the architecture of Apache Hive and explain its various components, as shown in the illustration in ~~figure~~ following figure :-



In the above figure we can see at the bottom the Hive sits on top of the Hadoop Distributed file system (HDFS) and MapReduce systems. In the case of MapReduce, figure-1 shows both the Hadoop1 and Hadoop2 components.
With Hadoop 1, Hive queries are converted to MapReduce code and executed using the MapReduce V1 (MRV1) infrastructure, like the Job Tracker and Task Tracker.
With Hadoop 2, YARN has decoupled resource management and scheduling from the MapReduce framework.
Hive queries can still be converted to MapReduce code and executed, now with MapReduce ~~framework~~. V2 (MRV2) and the YARN infrastructure.
There is a new framework under development called Apache Tez, which is designed to improve Hive performance for batch-style queries and support smaller interactive (also known as real-time) queries.
HDFS provides the storage, and MapReduce provides the parallel processing capability for higher-level functions within the Hadoop ecosystem.

* Getting Started with Apache Hive :→

Installation :→

The setup steps seem something like this :

1. Download the latest Hive →
   We downloaded Hive version 11.0. You also need the Hadoop and MapReduce subsystems, so be sure to complete Step 2

2. Download Hadoop version 1.2.1

3. Using the commands in the following listing, place the releases in separate directories, and then uncompress and untar them.

(untar is one of those geeky unix terms which simply means to expand an achived software package.)

```
$ mkdir hadoop ; cp hadoop-1.2.1.tar.gz hadoop ; cd hadoop
$ gunzip hadoop-1.2.1.tar.gz
$ tar xvf *.tar
$ mkdir hive ; cp hive-0.11.0.tar.gz hive ; cd hive
$ gunzip hive-0.11.0.tar.gz
$ tar xvf *.tar
```

4. Using the commands in the following listing, set up your Apache Hive environment variable, including ~~Hadoo~~ HADOOP_HOME, JAVA_HOME, HIVE_HOME and PATH, in your shell profile script:

```
export HADOOP_HOME=/Home/user/Hive/hadoop/hadoop-1.2.1
export JAVA_HOME=/opt/jdk
export HIVE_HOME=/home/user/hive/hive-0.11.0
export PATH=$HADOOP_HOME/bin:$HIVE_HOME/bin:$JAVA_HOME
    /bin:$PATH
```

5. Create the Hive configuration file that you'll use to define specific Hive configuration settings.

```
$ cd $HIVE_HOME/conf
$ cp hive-default.xml.template hive-site.xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<Configuration>
<!-- Hive Execution Parameters ->
<property>
  <name> hive.metastore.warehouse.dir </name>
  <value> /home/biadmin/Hive/warehouse </value>
  <description> location of default database for the warehouse
</description>
</property>
</configuration>
```

Name of Lecturer : .....

Because you're running Hive in stand-alone mode on a virtual machine rather than in a real-life Apache Hadoop cluster, Configure the system to use local storage rather than the HDFS: Simply set the hive.metastore.warehouse.dir parameter.
When you start a Hive client, the $HIVE-HOME environment variable tells the client that it should look for your configuration file (hive-site.xml) in the Conf directory.

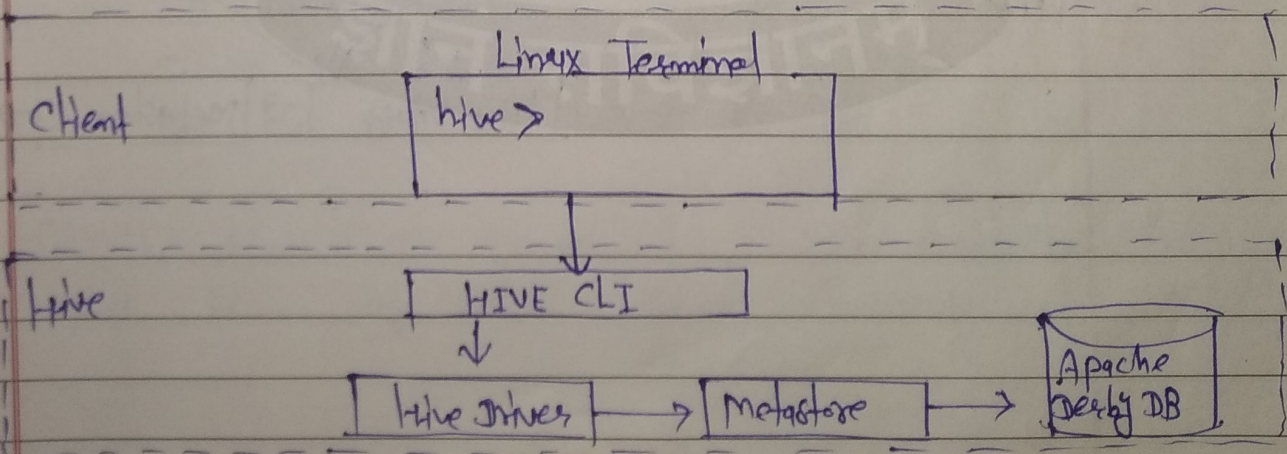
Examining the Hive Clients :→

In general, we have the following hive clients :

1. Hive Command-line interface (CLI)
2. Hive web Interface (HWI) Server
3. Squirrel

1. The Hive CLI Client :→

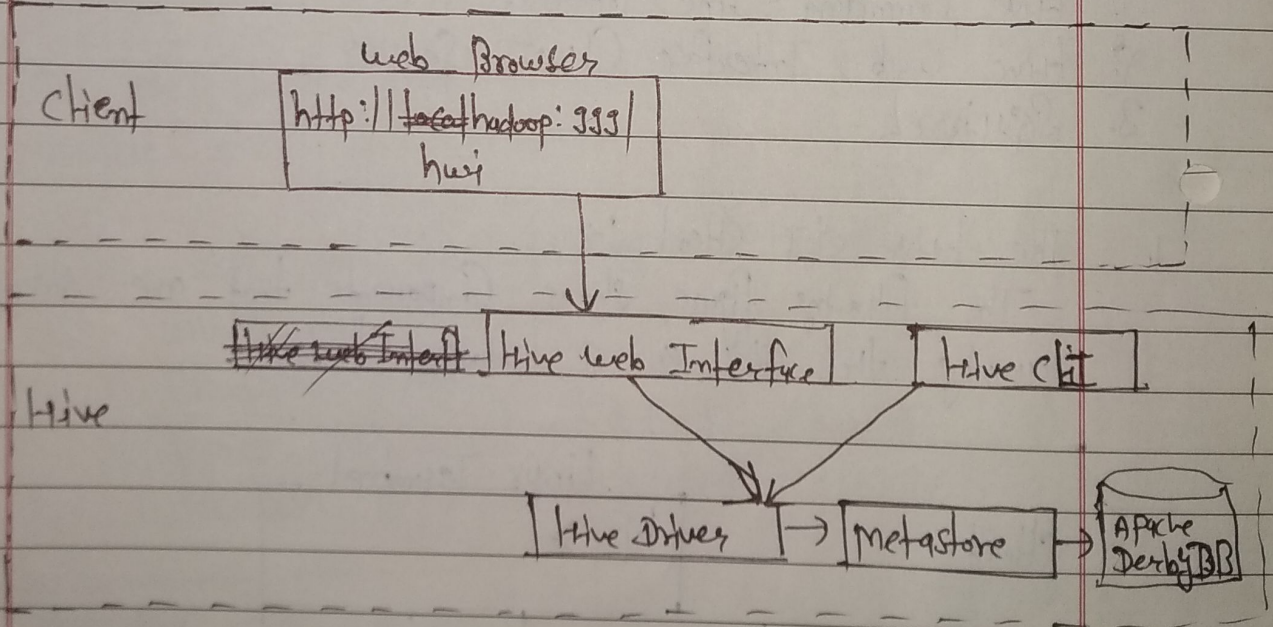
The following figure shows Components that are required when running the CLI on a Hadoop Cluster.



The examples in this chapter, we run Hive in local mode which uses local storage, rather then the HDFS, for your data. To run the HIVE CLI, you execute the hive command and specify the CLI as the service you want to run.

2. Hive web Interface (HWI) Server :→

When we want to access Hive using a web browser, you first need to start Hive web Interface (HWI) Server and then point your browser to the port on which the server is listening. following figure shows the HWI Client Configuration.



The following steps shows you what you need to do before you can start the HWI Server :

1. Configure the $HIVE-HOME/conf/hive-site.xml file as below to ensure that Hive can find and load the HWI's Java Server pages.

```
<property>
<name> hive.hwi.war.file </name>
<value> ${HIVE-HOME 3}/lib/hive-hwi.war </value>
<description>
This is the WAR file with the jsp content for Hive web
Interface
</description>
</property>
```

2. The HWI Server requires Apache Ant libraries to run, so download Ant.

3. Install Ant using the following Commands :

```
mkdir ant
cp apache-ant-1.9.2-bin.tar.gz ant ; cd ant
gunzip apache-ant-1.9.2-bin.tar.gz
tar xvf apache-ant-1.9.2-bin.tar
```

4. Set the $ANT-LIB Environment Variable and start the HWI Server by using following Commands :

$ export ANT_LIB = /home/user/ant/apache-ant-1.9.2/lib
$ bin/hive --service hwi

3. Squirrel as Hive client with the JDBC Driver :→

The last HIVE client is the open source tool SQuirrel SQL. It provides a user interface to Hive and simplifies the tasks of querying large tables and analyzing data with Apache Hive.

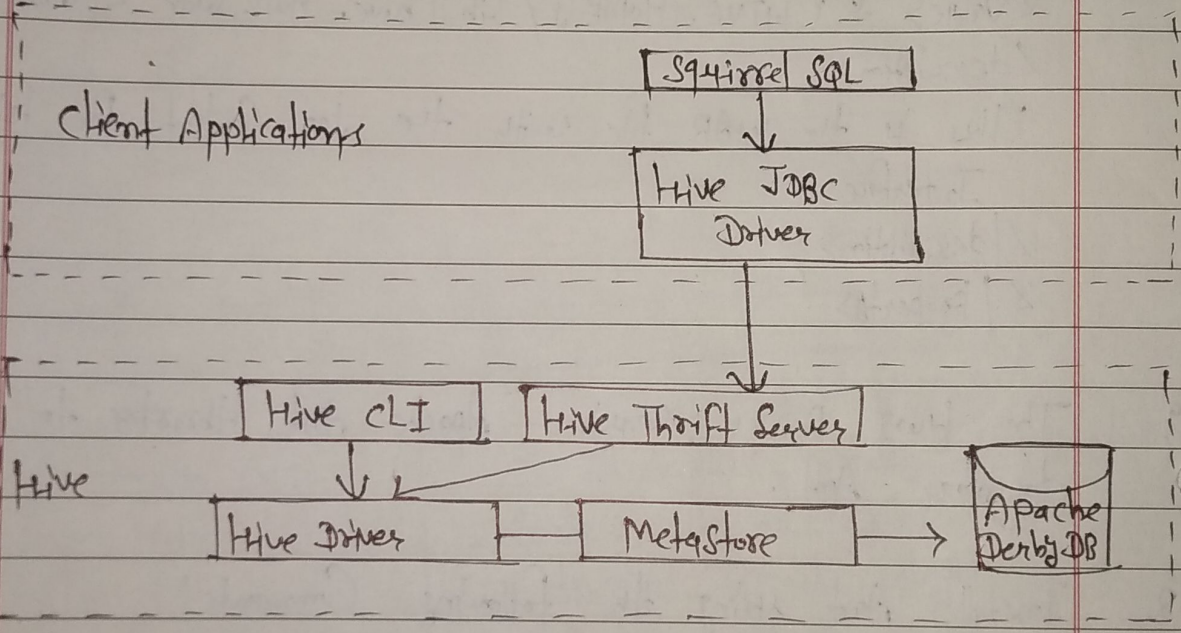


figure :- Using the Squirrel Client with Apache Hive.

In the above figure, we can see that the Squirrel Client uses the JDBC APIs to pass commands to the Hive Driver by way of the Hive Thrift Server.

* Hive Data types :→

Data types are very important elements in Hive query language and data modeling. for defining the table Column types, we must have to know about the data types and its usage.

The following gives brief overview of some data types in Hive ~~ther~~ these are :-

1. Numeric Types
2. String Types
3. Date/Time Types.
4. Complex Types

1. Numeric Types :→

	Type	Memory Allocation
(1.)	~~In~~ TINY INT	1- byte Signed Integer (-128 to 127)
(2)	SMALL INT	2-byte byte Signed integer (-32768 to 32767)
(3)	INT	4- byte Signed Integer
(4)	BIG INT	8- byte Signed Integer
(5)	FLOAT	4- byte Signed peeCision floating point number
(6)	DOUBLE	8- byte double precision floating point number
(7)	DECIMAL	we Can define precision and Scale in this type.

2. String Types :→

	Type	Length
(1)	CHAR	255
(2)	VARCHAR	1 to 65355
(3)	STRING	We can define length here (no limit)

3. Date / Time types :→

	Type	Usage
(1)	Timestamp	Supports traditional Unix timestamp with optional nanosecond
(2)	Date	• It's in YYYY-MM-DD format • The range of values supported for the Date type is be 0000-01-01 to 9999-12-31 dependent on support by the primitive .eg Date type.

4. Complex Types :→

	Type	Usage
(1)	Arrays	Array<data_type> Negative values and non-Constant expressions not allowed
(2)	Maps	MAP<primitive_type, data_type> Negative values and non-Constant expressions not allowed.
(3)	Structs	STRUCT <Col_name : datat-type, ...>
(4)	Union	UNIONTYPE <data_type, datat-type, ...>

Name of Lecturer : .....

* Creating and Managing Databases and Tables :→

⇒ Create Database :→

Create Database is a statement used to create a database in Hive. A database in Hive is a namespace or a collection of tables.

Syntax of the Create Database :→

```
CREATE DATABASE | SCHEMA [IF NOT EXISTS] <database name>
```

⇒ DROP Database Statement :→

Drop Database is a statement that drops all the tables and deletes the database. Its Syntax is as follows:

```
DROP DATABASE Statement DROP (DATABASE | SCHEMA) [IF EXISTS]
  database_name [RESTRICT | CASCADE];
```

The following queries are used to drop a database. Let us assume that the database name is Userdb.

```
hive> DROP DATABASE IF EXISTS Userdb;
```

The following query drops the database using CASCADE. It means dropping respective tables before dropping the database.

[hive> DROP DATABASE IF EXISTS userdb CASCADE;]

The following query drops the database using SCHEMA.

[hive> DROP SCHEMA userdb;]

⇒ Create Table Statement :→

Create Table is a statement used to Create a table in Hive. The Syntax and Example are as follows:-

Syntax:→

CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS]
[db-name.] table-name
[(col-name data-type [COMMENT col-comment],...)]
[COMMENT table-Comment]
[ROW FORMAT row-format]
[STORED AS file-format]

~~If Let us assume~~

Example:-

Let us assume here give the data

Example

Let us assume you need to create a table named Employee using CREATE TABLE statement. The following table lists the fields and their data types in Employee table:

Sr. No.	Field Name	Data Type
1	Eid	int
2	Name	String
3	Salary	float
4	~~Dept~~ Designation	String.

The following data is a Comment, Row formatted fields such as field terminator, Lines terminator and Stored file type.

```
COMMENT 'Employee details'
FIELDS TERMINATED BY '\t'
LINES TERMINATED BY '\n'
STORED IN TEXT FILE
```

The following query creates a table named employee using the above data.

```
CREATE TABLE IF NOT EXISTS employee (eid int, name String
  Salary String, destination String)
COMMENT 'Employee details'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '(t'
```

LINES TERMINATED BY '\n'
STORED AS TEXTFILE;

⇒ Load Data Statement →

Generally, after creating a table in SQL, we can insert data using the Insert statement. But in Hive, we can insert data using the LOAD DATA statement.

While inserting data into Hive, it is better to use LOAD DATA to store bulk records.

~~There are two ways to load data:~~

~~1. from local file System~~

~~2. from Hadoop file System.~~

Syntax:-

The syntax for load data is as follows:

LOAD DATA [LOCAL] INPATH 'filepath' [OVERWRITE] INTO TABLE tablename [PARTITION (partcol1 = val1, partcol2 = val2 ...)]

- LOCAL is identifier to specify the local path. It is optional.
- OVERWRITE is optional to overwrite the data in the table.
- PARTITION is optional.

Example :→

We will insert the following data into the table. It is a text file named Sample.txt in the ~~directory~~ /home/user directory.

1201	Gopal	50000	OP Admin
1202	Ram	45000	Hr admin
1203	Shyam	55000	Proof reader
1204	Mohan	35000	Technical Manager
1205	Mamtha	60000	Branch Manager

The following query loads the given text into the table.

```
hive> LOAD DATA LOCAL INPATH '/home/user/sample.txt'
  OVERWRITE INTO TABLE employee;
```

On successful download, you get to see the following response.

```
Ok
Time taken: 15.905 seconds
hive>
```

* Alter Table Statement :→

It is used to alter a table in Hive.

Syntax :-

The statement takes any of the following syntaxes based on what attributes we wish to modify in a table.

```
ALTER TABLE name RENAME TO new_name
ALTER TABLE name ADD COLUMNS (col_spec [, col_spec ...])
ALTER TABLE name DROP [COLUMN] column_name
ALTER TABLE name CHANGE column_name new_name new_type
ALTER TABLE name REPLACE COLUMNS (col_spec [, col_spec ...])
```

Rename To... Statement :→

The following query renames the table from employee to emp.

```
ALTER TABLE employee RENAME TO emp;
```

* Change Statement :→

The following table contains the fields of employee table and it shows the fields to be changed (in Black)

Field Name	Convert from Data type	change field Name	Convert to Data type
eid	int	eid	int
name	String	ename	String
Salary	float	Salary	Double
designation	String	designation	String

The following queries rename the Column name and Column data type using the above data.

```
ALTER TABLE Employee CHANGE name ename String;
ALTER TABLE Employee CHANGE Salary Salary Double;
```

* Add Columns Statement :→

The following query adds a column named dept to the Employee table.

```
hive> ALTER TABLE Employee ADD COLUMNS (
      dept STRING COMMENT 'Department name');
```

* Replace Statement :→

The following query deletes all the Columns from the employee table and replaces it with emp and name Columns:

```
hive> ALTER TABLE employee REPLACE COLUMNS (
      eid INT empid Int,
      ename STRING name string);
```

* Drop TABLE Statement :→

The syntax is as follows :-

```
DROP TABLE [IF EXISTS] table_name;
```

The following query drops a table named employee:

```
hive> DROP TABLE IF EXISTS employee;
```

* Hive Data Manipulation Language (DML) Commands :→

Hive DML (Data Manipulation Language) Commands are used to Insert, update, retrieve, and delete data from the Hive table once the table and database Schema has been defined using Hive DDL Commands.

The Various Hive DML Commands are :

1. ~~Load~~ LOAD
2. SELECT
3. INSERT
4. DELETE
5. UPDATE
6. EXPORT
7. IMPORT

1. LOAD Command :→

The LOAD Statement in Hive is used to move data files into the locations corresponding to Hive tables.

→ If a LOCAL keyword is specified, then the LOAD Command will look for the file path in the local file system.
→ If the LOCAL keyword is not specified, then the Hive will need the absolute URI of the files.
→ In Case the keyword OVERWRITE is specified, then the Contents of the target table / partition will be deleted and replaced by the files referred by file path.
→ If the OVERWRITE keyword is not specified, then the files referred by file path will be appended to the table.

Syntax :->

```
LOAD DATA [LOCAL] INPATH 'filepath' [OVERWRITE] INTO TABLE
tablename [PARTITION (partCol1 = val1 , partCol2 = val2 ...) ];
```

Example ->

Here we are trying to load data from the 'dab' file in the local filesystem to the 'emp_data' table.

> LOAD DATA LOCAL INPATH '/home/dataflair/dab' INTO TABLE emp_data;

9. SELECT COMMAND :->

The SELECT Statement in Hive is similar to the SELECT Statement in SQL used for retrieving data from the database.

Syntax :->

SELECT col1 , col2 FROM tablename ;

Example :->

SELECT * from emp_data ;

3. INSERT Command :→

The INSERT Command in Hive loads the data into a Hive table. We can do insert to both the Hive table or partition.

(a) INSERT INTO :→

The INSERT INTO Statement appends the data into existing data in the table or partition. INSERT INTO Statement works from Hive Version 0.8.

Syntax →

```
[ INSERT INTO TABLE tablename1 [PARTION (partcol1 = val1,
  partcol2 = val2 ...)] Select_statement1 FROM from_statement; ]
```

Example :→

Here in this example, we are trying to Insert the data of 'emp_data' table created above into the table 'example'.

firstly we create the table :→

```
> CREATE TABLE IF NOT EXISTS example ( eid STRING, name
  STRING, dep STRING, state STRING, salary STRING, year
  STRING );
```

Now Insert Statement to load data into table "example".

```
[ >INSERT INTO TABLE example SELECT emp.emp_id, emp.emp_name
  , emp.emp_dep, emp.state, emp.salary, emp.year_of_joining FROM
  emp_data emp ; ]
```

(b) INSERT OVERWRITE :→

The INSERT OVERWRITE table overwrites the existing data in the table or partition.

Syntax :-

INSERT OVERWRITE TABLE tablename1 [PARTITION (partcol1 = val1, ...) [IF NOT EXISTS]] Select - statement FROM from - statement ;]

Example :→

Here we are overwriting the existing data of the table 'example' with the data of table/Database 'dummy' using INSERT OVERWRITE Statement.

> INSERT OVERWRITE TABLE example SELECT dmy. emroll, dmy. name, dmy. department, dmy. state, dmy. salary, dmy. year FROM dummy dmy;

By using the SELECT Statement we can verify whether the existing data of the table 'example' is overwritten by the data of table 'dummy' or not.

> SELECT * from example ;

(C). INSERT.. VALUES :→

INSERT.. VALUES statement in Hive inserts data into the table directly from SQL. It is available from Hive 0.14.

Syntax →

```
INSERT INTO TABLE tablename [PARTITION (partCol1 [= Val1],
partCol2 [= Val2]...)] VALUES values_row [, values_row...];
```

Example:→

Inserting data into the 'student' table using INSERT.. VALUES statement.

```
> INSERT INTO TABLE student VALUES (101, 'Callen', 'IT', '7.8'),
(103, 'Joseph', 'CS', '8.2'), (105, 'Alex', 'IT', '7.3');

>SELECT * FROM student;
```

4. DELETE Command :→

The DELETE Statement in hive deletes the table data. If the WHERE clause is specified, then it deletes the rows that satisfy the condition in where clause.

The DELETE statement can only be used on the hive tables that support ACID (ACID property → Atomicity, Consistency, Isolation, Durability).

Syntax →

DELETE FROM tablename [WHERE expression];

Example :→

In the below example, we are deleting the data of the Student from table 'Student' whose roll_no is 105.

> DELETE FROM student WHERE roll_no = 105;

By using the SELECT statement we can verify whether the data of the student from table 'student' whose roll_no is 105 is deleted or not.

> SELECT * FROM student;

5. UPDATE Command :→

The Update can be performed on the hive tables that support ACID.
The Update statement in Hive deletes the table data. If the WHERE clause is specified, then it updates the column of the rows that satisfy the condition in WHERE clause.

Syntax :-

UPDATE tablename SET Column = Value [, Column = Value ...]
[WHERE expression];

Example :→

In this example, we are updating the branch of the student whose roll_no is 103 in the 'student' table using an ~~update~~ UPDATE statement.

> UPDATE student SET branch = 'IT' WHERE roll_no = 103;

6. ~~Export~~ EXPORT Command :→

The Hive EXPORT statement exports the table or partition data along with the metadata to the specified output location in the HDFS.

metadata is exported in a metadata file, and data is exported in a subdirectory 'data'.

Hive Partitions :→

Apache Hive organizes table into partitions. Partitioning is a way of dividing a table into related parts based on the values of particular columns like date, City and department. Each table in the hive can have one or more partition keys to identify a particular partition. Using partition it is easy to do queries on slices of the data.

Syntax :→

~~Export~~

```
EXPORT TABLE tablename [PARTITION (part_column = "value"
  [, ...])] TO 'export_target_path' [FOR replication
  ('eventid')];
```

Example :→

Here in this example, we are exporting the Student table to the HDFS directory "export_from_hive".

```
>EXPORT TABLE student TO 'export_from_hive';
```

The table successfully exported. You can check for the _metadata file and data sub-directory using ls command

```
$ hadoop fs -ls -R /user/dataflair/export_from_hive/;
```

7. ~~Impo~~

7. IMPORT Command :→

The Hive IMPORT Command imports the data from a specified location to a new table or already existing table.

Syntax :→

```
IMPORT [[EXTERNAL] TABLE new_or_original_tablename
[PARTITION (part_column = "value" [, ...])]]
FROM 'source_path' [LOCATION 'import_target_path'];
```

Example :→

Here in this example, we are importing the data exported in the above example into a new Hive table 'imported_table'.

```
> import IMPORT TABLE imported_table from '/user/dataflair/export_from_hive';
```

* Querying and Analyzing Data :→

- Hive data types, Hive's DDL and Hive's DML, helps Creating and managing tables but now we help you explore some HiveQL features for querying and analysing data. We begin by exploring table joins in Hive.

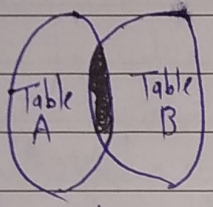
Joining tables with Hive :→

- In relational database modelling, we split the tables for normalization purpose. and use Join operation get the data when it is required.
- Database normalization is a technique that guards against data loss, redundancy, and other anomalies as data is update and retrieved.
- MapReduce is the Engine for Joining tables, and the Hadoop file System (HDFS) is the Underlying Storage.
- Hive table reads and writes via HDFS usually involve very large blocks of data, more data you can manage altogether in one table, the better the overall performance.
- With this background information in mind, we can tackle making Joins with Hive. Fortunately, the Hive development Community was realistic and understood that users would want and need to Join tables with HiveQL.
- Hive supports Equijoins, a specific type of Join that only uses equality Comparisons in the Join predicate.
- This restriction is only because of limitations on the Underlying MapReduce Engine.

Hive Join Example :-

1. Inner Join :→

~~Full outer Join :→~~



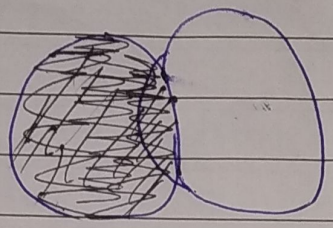
Basically, to Combine and retrieve the records from multiple tables we use Hive Join clause. Moreover, in SQL JOIN is as same as OUTER JOIN. Moreover, by using the primary keys and foreign keys of the tables JOIN Condition is to be raised.
Furthermore, the below query executes JOIN the CUSTOMER and ORDER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

```
> SELECT C.ID, C.Name, C.AGE, O.AMOUNT FROM CUSTOMERS
  C JOIN ORDERS O ON (C.ID = O.CUSTOMER_ID);
```

2. 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.
In additional, 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.

> SELECT C.ID, C.Name, O.Amount, O.DATE FROM CUSTOMERS C LEFT OUTER JOIN ORDERS O ON (C.ID = O.CUSTOMER_ID);

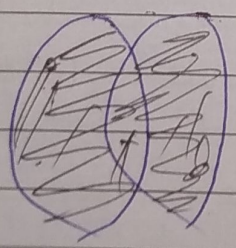


3. Full Outer Join :->

The major purpose of this HiveQL full outer Join is it combines the records of both the left and the right outer tables which fulfills the Hive JOIN condition. Moreover, this joined table contains either all the records from both the tables or fills in NULL values for missing matches on either side.

However, the below query shows FULL OUTER JOIN between CUSTOMER as well as ORDER tables:

> SELECT C.ID, C.NAME, O.AMOUNT, O.DATE FROM CUSTOMERS C FULL OUTER JOIN ORDERS O ON (C.ID = O.CUSTOMER_ID);



⇒ Improving your Hive queries with Indexes :→

Creating an Index is common practice with relational databases when we want to speed access to a Column or Set of Columns in your database. Without an index, the database System has to read all rows in the table to find the data we have selected. Indexes become even more essential when the tables grow extremely large. Hive supports index creation on tables.

⇒ Hive Indexes :→

Hive indexes are implemented as tables. ~~This is why we need to first~~ Create the index table and then build it to populate the table.

Therefore, we can use indexes in at least two ways:

1. Count on the System to automatically use indexes that you create.
2. Rewrite some queries to leverage the new index table.

⇒ Windowing in HiveQL :→

The Concept of windowing, introduced in the SQL:2003 standard, allows the SQL programmer to create a frame from the data against which aggregate and other window functions can operate.

HiveQL now supports windowing per the SQL standard. Examples are quite helpful when explaining windowing and aggregate functions.

- We first discovered this data set was, 'what exactly is the average flight delay per day?' so we created a query in Listing 13-13 that produces the average departure delay per day in 2008.

* Other key HiveQL features :→

1. Security :→

Apache Hive provides a security subsystem that can be quite helpful in preventing accidental data corruption or compromise among trusted members of workgroup.

2. Multi-User Locking :→

Hive supports multi-user warehouse access when configured with Apache Zookeeper. Without this support, one user may read a table at the same time another user is deleting that table - which is, obviously, unacceptable.

3. Functions :→

HiveQL provides a rich set of built-in operators, built-in functions, built-in aggregate functions, and built-in table-generating functions. Several examples in this chapter use built-in operators as well as built-in aggregate function (AVG, MIN, and Count, for example). To list all built-in functions for any particular Hive release, use the SHOW FUNCTIONS HIVEQL Command. You can also retrieve information about a built-in function by using the HiveQL Commands DESCRIBE FUNCTION function-name and DESCRIBE FUNCTION EXTENDED function-name.

4. Data Compression :→

Data Compression can not only save space on the HDFS but also improve performance by reducing the overall size of input/output operations. Additionally, Compression between the Hadoop mappers and reducers can improve performance, because less data is passed between nodes in the cluster. Hive supports intermediate Compression between the mappers and reducers as well as table output Compression. Hive also understands how to ingest Compressed data into the warehouse. files Compressed with GZip or BZip2 can be read by Hive's LOAD DATA Command.