Unit 3
HDFS, HIVE AND HIVEQL, HBASE

Dr. Nilesh M. Patil

Associate Professor,

Dept. of Computer Engineering, DJSCE

Syllabus

- HDFS-Overview, Installation and Shell, Java API; Hive Architecture and Installation, Comparison with Traditional Database, HiveQL Querying Data, Sorting, and Aggregating,
- Map Reduce Scripts, Joins & Subqueries
- HBase concepts, Advanced Usage, Schema Design, Advance Indexing, PIGGrunt – pig data model – Pig Latin – developing and testing Pig Latin scripts
- Zookeeper , how it helps in monitoring a cluster
- Build Applications with Zookeeper and HBase
- **12 Hours**
- **Marks: 30 (approx.)**

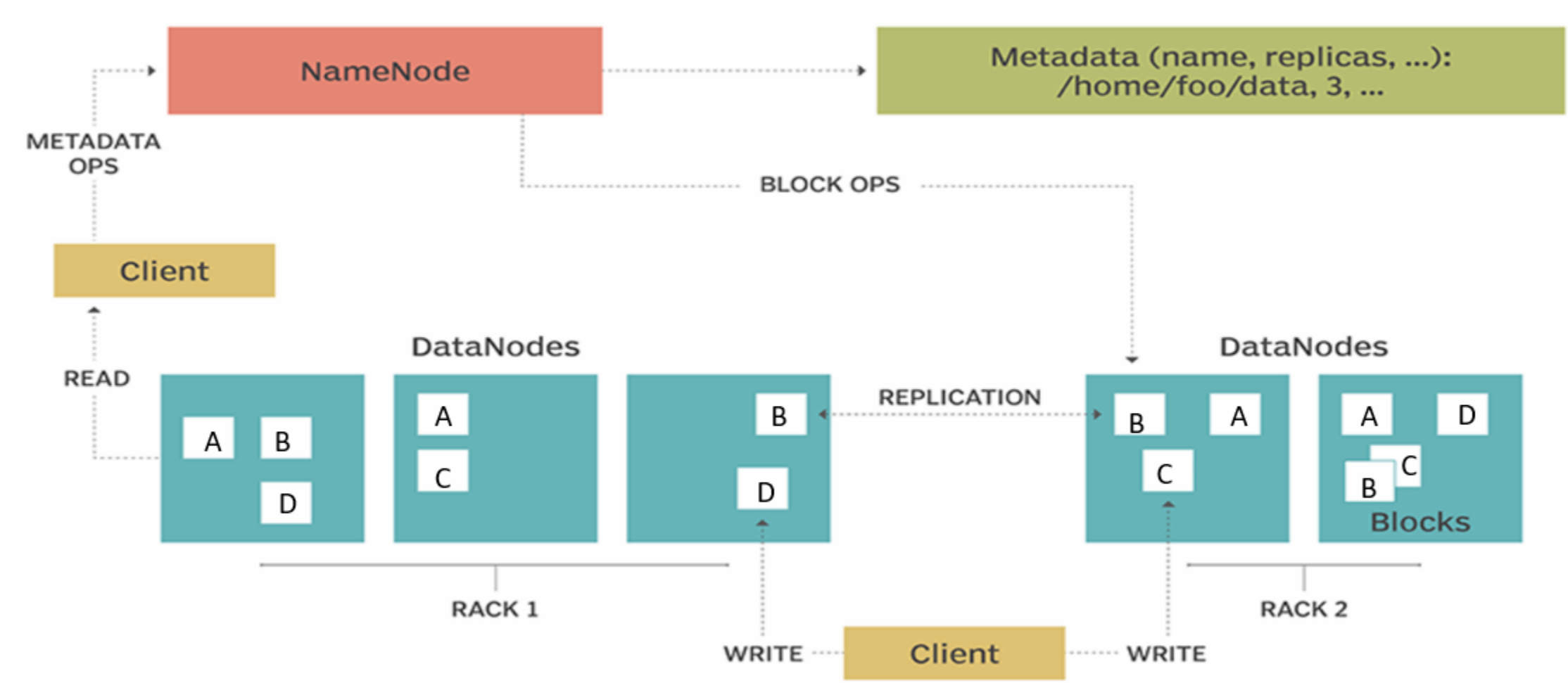
Hadoop Distributed File System (HDFS)

- Hadoop File System was developed using distributed file system design.
- It is run on commodity hardware.
- Unlike other distributed systems, HDFS is highly fault-tolerant and designed using low-cost hardware.
- HDFS holds a 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 a redundant fashion to rescue the system from possible data losses in case of failure.
- HDFS also makes applications available for parallel processing.

HDFS Features

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

HDFS Architecture

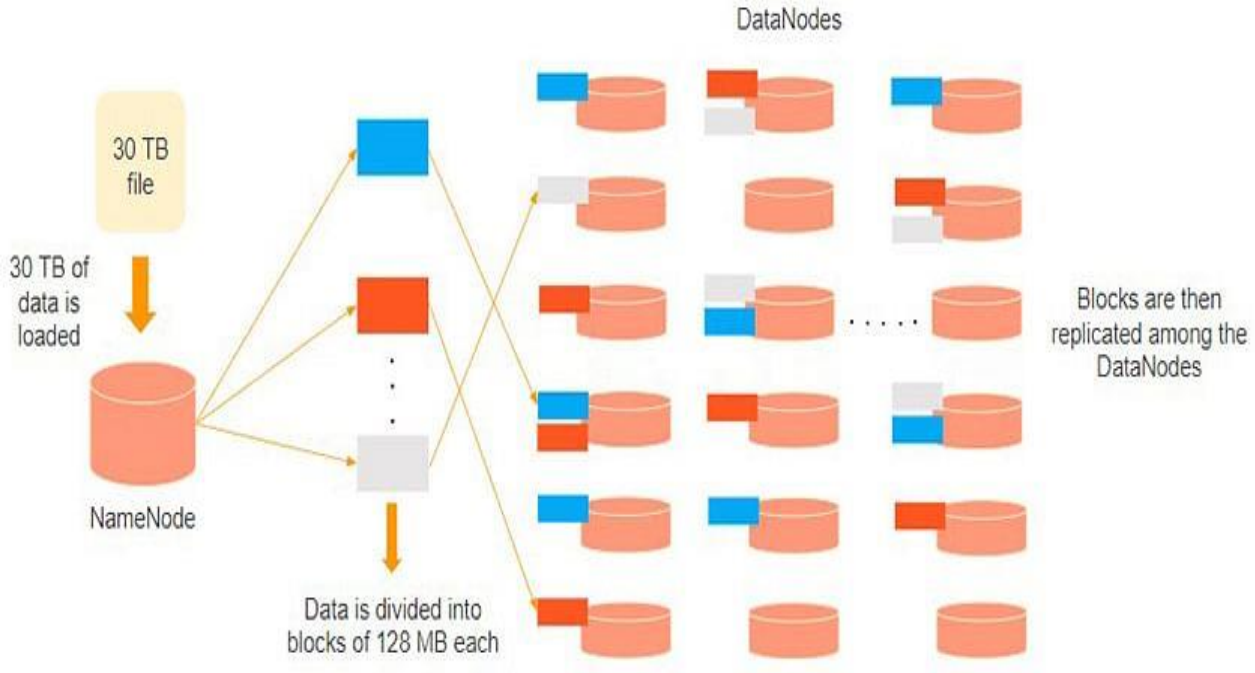


NameNode

- NameNode is the master node that contains the metadata.
- The NameNode is responsible for the workings of the data nodes.
- NameNode is the primary server that manages the file system namespace and controls client access to files.
- The NameNode performs file system namespace operations, including opening, closing and renaming files and directories.
- The NameNode also governs the mapping of blocks to the DataNodes.

DataNode

- The DataNodes are called the slaves.
- The DataNodes read, write, process, and replicate the data.
- They also send signals, known as heartbeats, to the NameNode. These heartbeats show the status of the DataNode.
- While there is only one NameNode, there can be multiple DataNodes.



- Consider that 30TB of data is loaded into the NameNode.
- The NameNode distributes it across the DataNodes, and this data is replicated among the DataNodes.
- You can see in the image above that the blue, grey, and red data are replicated among the three DataNodes.
- Replication of the data is performed three times by default. It is done this way, so if a commodity machine fails, you can replace it with a new machine that has the same data.

The Communication Protocol in HDFS

- All HDFS communication protocols are layered on top of the TCP/IP protocol
- A client establishes a connection to a configurable TCP port on the Namenode machine. It talks ClientProtocol with the Namenode.
- The Datanodes talk to the Namenode using Datanode protocol.
- RPC abstraction wraps both ClientProtocol and Datanode protocol.
- Namenode is simply a server and never initiates a request; it only responds to RPC requests issued by DataNodes or clients.

Application Programming Interface

- HDFS provides Java API for application to use.
- Python access is also used in many applications.
- A C language wrapper for Java API is also available.
- A HTTP browser can be used to browse the files of a HDFS instance.

FS Shell, Admin and Browser Interface

- HDFS organizes its data in files and directories.
- It provides a command line interface called the FS shell that lets the user interact with data in the HDFS.
- The syntax of the commands is similar to bash and csh.
- Example: to create a directory /foodir

/bin/hadoop dfs –mkdir /foodir

- There is also DFSAdmin interface available
- Browser interface is also available to view the namespace.

Space Reclamation

- When a file is deleted by a client, HDFS renames file to a file in the /trash directory for a configurable amount of time.
- A client can request an undelete in this allowed time.
- After the specified time, the file is deleted and the space is reclaimed.
- When the replication factor is reduced, the Namenode selects excess replicas that can be deleted.
- Next heartbeat(?) transfers this information to the Datanode that clears the blocks for use.

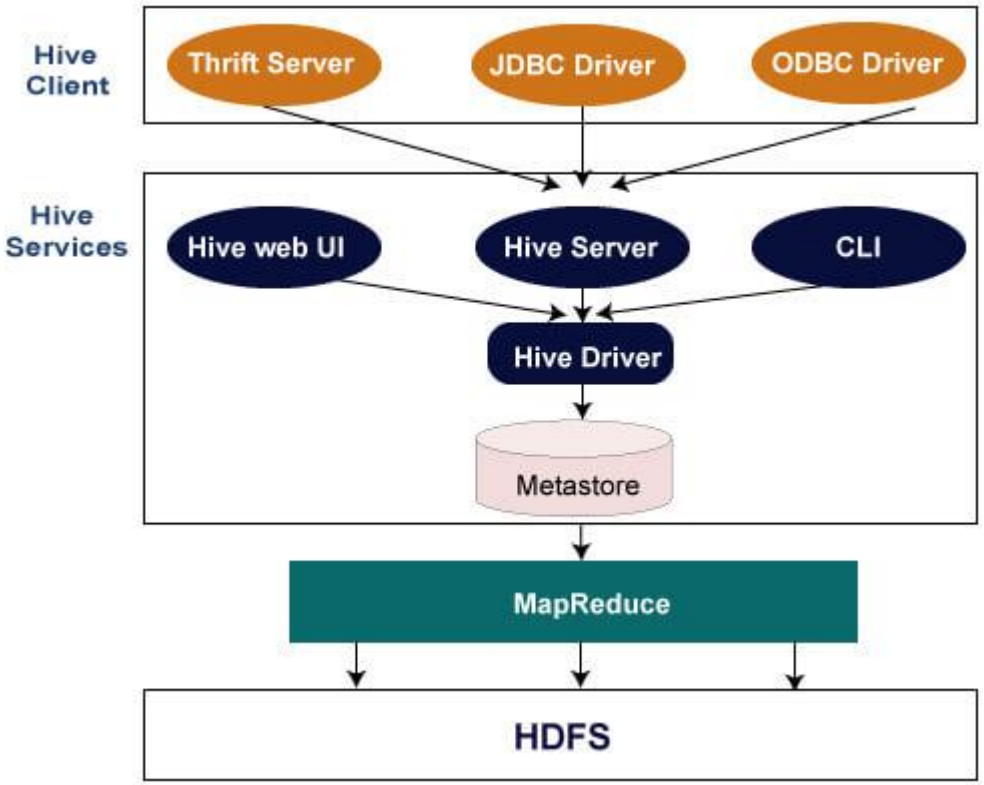
Apache HIVE Features

- Apache Hive is an open source data warehouse software for reading, writing and managing large data set files that are stored directly in either the Apache Hadoop Distributed File System (HDFS) or other data storage systems such as Apache HBase.
- Hive enables SQL developers to write Hive Query Language (HQL) statements that are similar to standard SQL statements for data query and analysis.
- It is designed to make MapReduce programming easier because you don’t have to know and write lengthy Java code.
- Instead, you can write queries more simply in HQL, and Hive can then create the map and reduce the functions.

Apache HIVE Features

- Included with the installation of Hive is the Hive metastore, which enables you to apply a table structure onto large amounts of unstructured data.
- Once you create a Hive table, defining the columns, rows, data types, etc., all of this information is stored in the metastore and becomes part of the Hive architecture.
- Other tools such as Apache Spark and Apache Pig can then access the data in the metastore.
- As with any database management system (DBMS), you can run your Hive queries from a command-line interface (known as the Hive shell), from a Java™ Database Connectivity (JDBC) or from an Open Database Connectivity (ODBC) application, using the Hive JDBC/ODBC drivers.

Hive Architecture



Hive Client

Hive allows writing applications in various languages, including Java, Python, and C++. It supports different types of clients such as:-

- Thrift Server - It is a cross-language service provider platform that serves the request from all those programming languages that supports Thrift.
- JDBC Driver - It is used to establish a connection between hive and Java applications. The JDBC Driver is present in the class org.apache.hadoop.hive.jdbc.HiveDriver.
- ODBC Driver - It allows the applications that support the ODBC protocol to connect to Hive.

Hive Services

The following are the services provided by Hive:-

- **Hive CLI** - The Hive CLI (Command Line Interface) is a shell where we can execute Hive queries and commands.
- **Hive Web User Interface** - The Hive Web UI is just an alternative of Hive CLI. It provides a web-based GUI for executing Hive queries and commands.
- **Hive MetaStore** - It is a central repository that stores all the structure information of various tables and partitions in the warehouse. It also includes metadata of column and its type information, the serializers and deserializers which is used to read and write data and the corresponding HDFS files where the data is stored.
- **Hive Server** - It is referred to as Apache Thrift Server. It accepts the request from different clients and provides it to Hive Driver.
- **Hive Driver** - It receives queries from different sources like web UI, CLI, Thrift, and JDBC/ODBC driver. It transfers the queries to the compiler.
- **Hive Compiler** - The purpose of the compiler is to parse the query and perform semantic analysis on the different query blocks and expressions. It converts HiveQL statements into MapReduce jobs.
- **Hive Execution Engine** - Optimizer generates the logical plan in the form of DAG of map-reduce tasks and HDFS tasks. In the end, the execution engine executes the incoming tasks in the order of their dependencies.

Hive Vs Traditional Database

Hive	Traditional database
Schema on READ – it does not verify the schema while it load the data	Schema on WRITE – table schema is enforced at data load time i.e. if the data being loaded doesn't conformed on schema in that case it will rejected
It's very easily scalable at low cost	Not much Scalable, costly scale up.
It's based on Hadoop notation that is Write once and read many times	In traditional database we can read and write many time
Record level updates is not possible in Hive	Record level updates, insertions and deletes, transactions and indexes are possible
OLTP (On-line Transaction Processing) is not yet supported in Hive but it's supported OLAP (On-line Analytical Processing)	Both OLTP (On-line Transaction Processing) and OLAP (On-line Analytical Processing) are supported in RDBMS

HIVE Data Types

- Hive data types are categorized in numeric types, string types, misc types, and complex types. A list of Hive data types is given below.

Integer Types

Type	Size	Range
TINYINT	1-byte signed integer	-128 to 127
SMALLINT	2-byte signed integer	32,768 to 32,767
INT	4-byte signed integer	2,147,483,648 to 2,147,483,647
BIGINT	8-byte signed integer	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Decimal Type

Type	Size	Range
FLOAT	4-byte	Single precision floating point number
DOUBLE	8-byte	Double precision floating point number

Date/Time Types

TIMESTAMP

- It supports traditional UNIX timestamp with optional nanosecond precision.
- As Integer numeric type, it is interpreted as UNIX timestamp in seconds.
- As Floating point numeric type, it is interpreted as UNIX timestamp in seconds with decimal precision.
- As string, it follows java.sql.Timestamp format "YYYY-MM-DD HH:MM:SS.fffffffff" (9 decimal place precision)

DATES

The Date value is used to specify a particular year, month and day, in the form YYYY--MM--DD. However, it didn't provide the time of the day. The range of Date type lies between 0000--01--01 to 9999--12--31.

String Types

STRING

The string is a sequence of characters. It values can be enclosed within single quotes (') or double quotes (").

Varchar

The varchar is a variable length type whose range lies between 1 and 65535, which specifies that the maximum number of characters allowed in the character string.

CHAR

The char is a fixed-length type whose maximum length is fixed at 255.

Complex Type

Type	Size	Range
Struct	It is similar to C struct or an object where fields are accessed using the "dot" notation.	struct('James','Roy')
Map	It contains the key-value tuples where the fields are accessed using array notation.	map('first','James','last','Roy')
Array	It is a collection of similar type of values that indexable using zero-based integers.	array('James','Roy')

MapReduce Scripts in Hive

- Similar to any other scripting language, Hive scripts are used to execute a set of Hive commands collectively.
- Hive scripting helps us to reduce the time and effort invested in writing and executing the individual commands manually.
- Hive scripting is supported in Hive 0.10.0 or higher versions of Hive.

Joins and SubQueries

- **JOINS :**
- Join queries can be performed on two tables present in Hive.
- Joins are of 4 types, these are :
- Inner join: The Records common to both tables will be retrieved by this Inner Join.
- Left outer Join: Returns all the rows from the left table even though there are no matches in the right table.
- Right Outer Join: Returns all the rows from the Right table even though there are no matches in the left table.
- Full Outer Join: It combines records of both the tables based on the JOIN Condition given in the query. It returns all the records from both tables and fills in NULL Values for the columns missing values matched on either side.

SUBQUERIES :

- A Query present within a Query is known as a subquery.
- The main query will depend on the values returned by the subqueries.
- Subqueries can be classified into two types :

i. Subqueries in FROM clause
ii. Subqueries in WHERE clause

- When to use :

i. To get a particular value combined from two column values from different tables.
ii. Dependency of one table values on other tables.
iii. Comparative checking of one column values from other tables.

- Syntax :

Subquery in FROM clause

SELECT <column names 1, 2...n>From (SubQuery) <TableName_Main >

Subquery in WHERE clause

SELECT <column names 1, 2...n> From<TableName_Main>WHERE col1 IN (SubQuery);

HiveQL

HBase

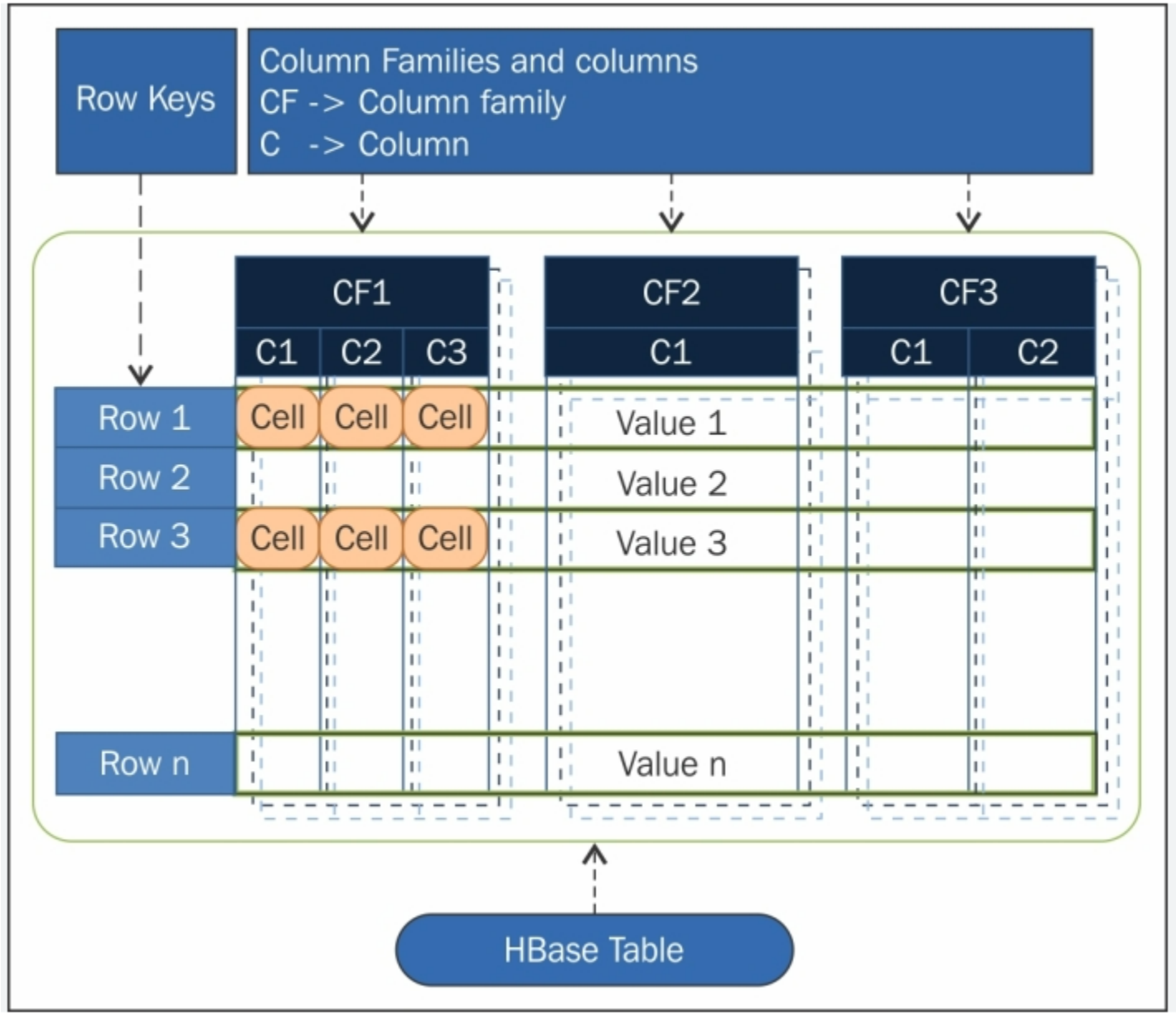
- HBase is a distributed column-oriented database built on top of the Hadoop file system.
- It is an open-source project and is horizontally scalable.
- HBase is a data model that is similar to Google's big table designed to provide quick random access to huge amounts of structured data.
- It leverages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

HBase Vs RDBMS

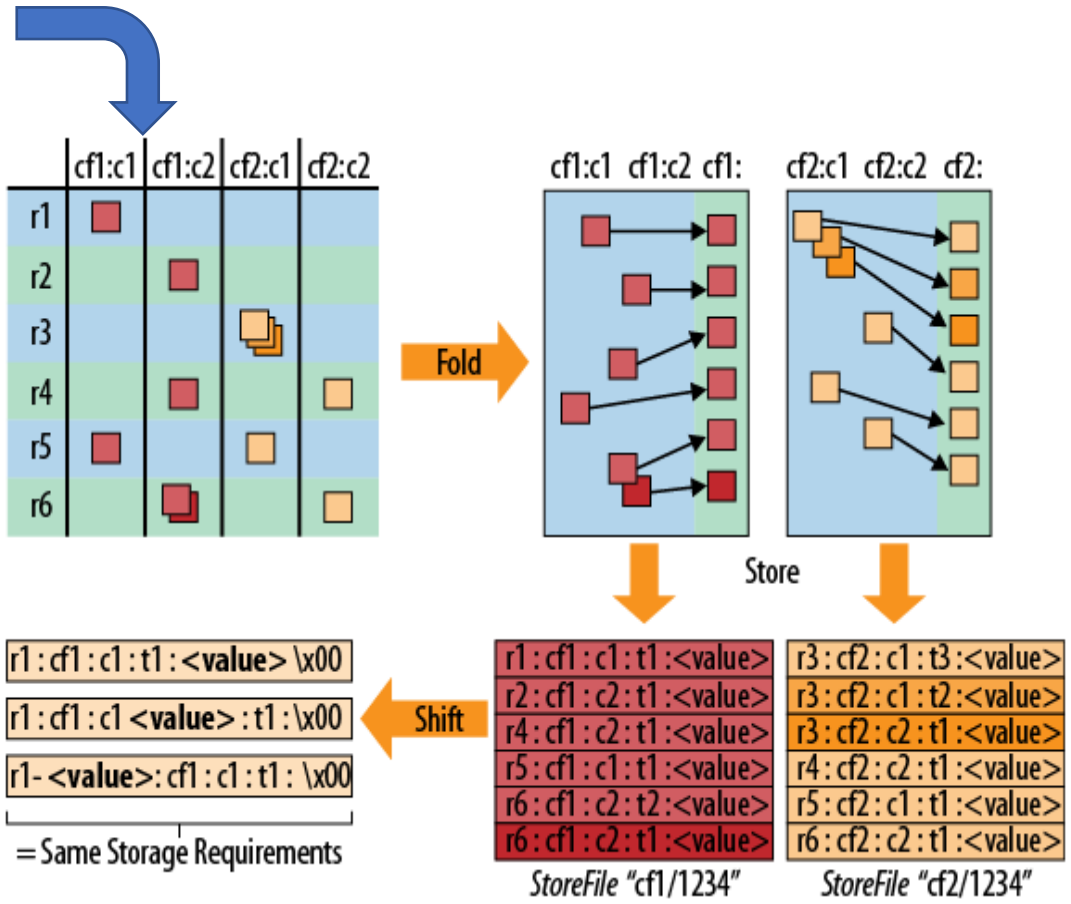
RDBMS	**HBase**
It requires SQL (structured query language)	NO SQL
It has a fixed schema	No fixed schema
It is row-oriented	It is column-oriented
It is not scalable	It is scalable
It is static in nature	Dynamic in nature
Slower retrieval of data	Faster retrieval of data
It follows the ACID (Atomicity, Consistency, Isolation and Durability) property.	It follows CAP (Consistency, Availability, Partition-tolerance) theorem.
It can handle structured data	It can handle structured, unstructured as well as semi-structured data
It cannot handle sparse data	It can handle sparse data

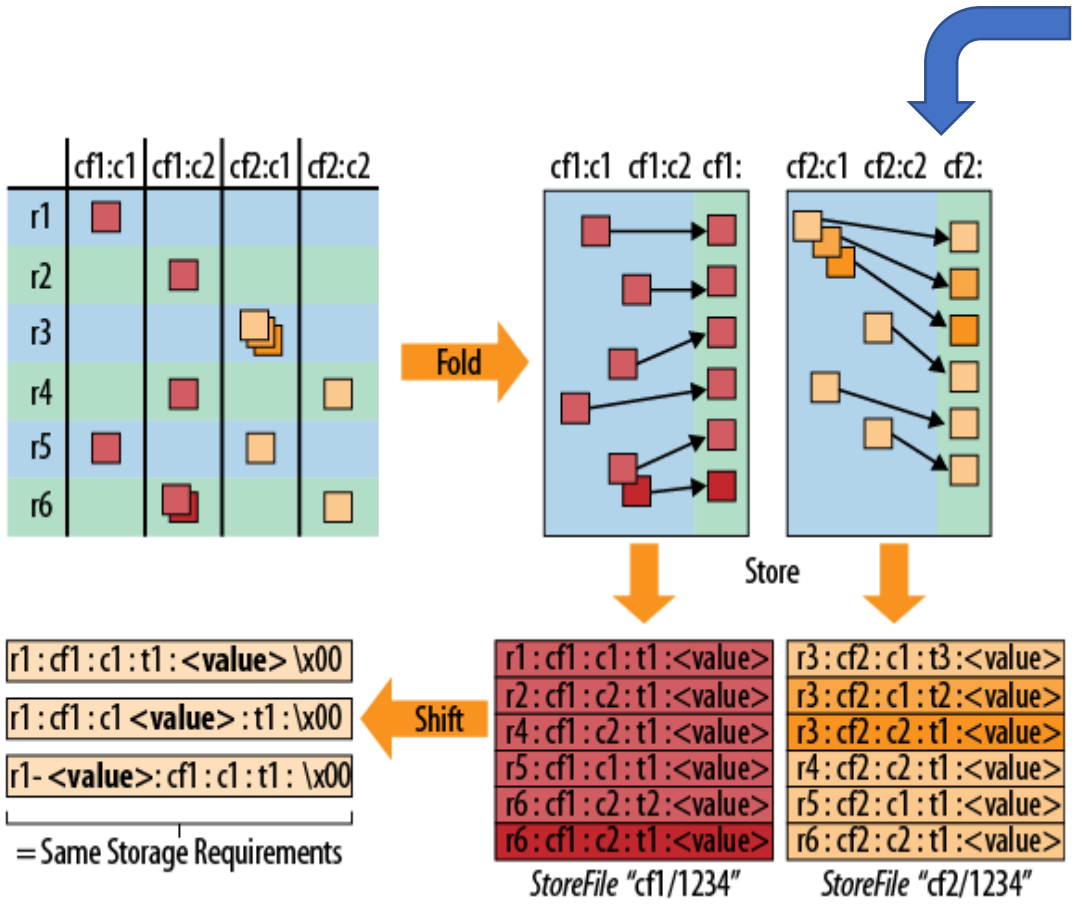
Hbase Advanced Usage

- HBase has two fundamental key structures: the row key and the column key.
- Both can be used to convey meaning, by either the data they store, or by exploiting their sorting order.
- HBase's main unit of separation within a table is the column family.

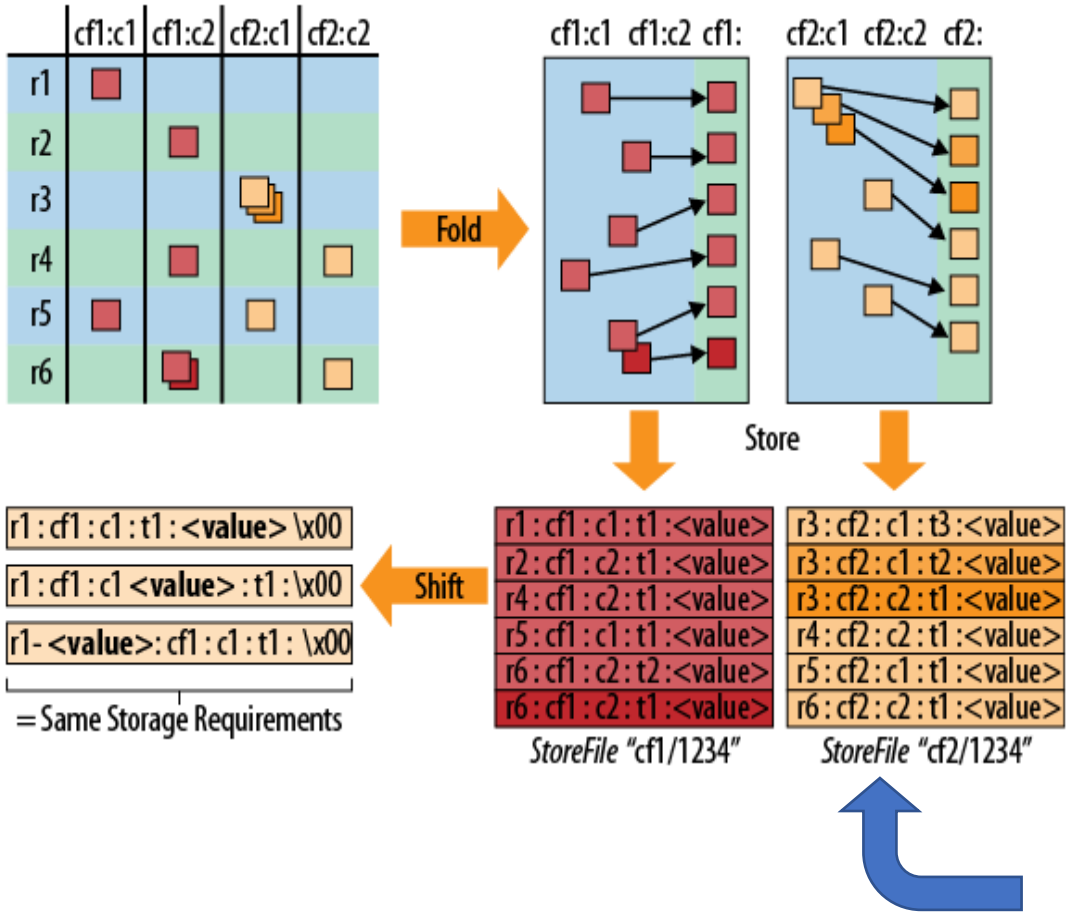


The top-left part of the figure shows the logical layout of your data: you have rows and columns. The columns are the typical HBase combination of a column family name and a column qualifier, forming the column key. The rows also have a row key so that you can address all columns in one logical row.





The top-right hand side shows how the logical layout is folded into the actual physical storage layout. The cells of each row are stored one after the other, in separate storage files per column family. In other words, on disk you will have all cells of one family in (one or more) StoreFiles, and all cells of another in a different file, located in a different directory.



The lower-right part of the figure shows the resultant layout of the logical table inside the physical storage files. The HBase API has various means of querying the stored data, with decreasing granularity from left to right: you can select rows by row keys and effectively reduce the amount of data that needs to be scanned when looking for a specific row, or a range of rows. Specifying the column family as part of the query can eliminate the need to search the separate storage files. If you only need the data of one family, it is highly recommended that you specify the family for your read operation.

Schema Design

- HBase table can scale to billions of rows and any number of columns based on your requirements.
- This table allows you to store terabytes of data in it.
- The HBase table supports the high read and writes throughput at low latency.
- A single value in each row is indexed; this value is known as the row key.
- The HBase schema design is very different compared to the relational database schema design.
- Some of the general concepts that should be followed while designing schema in HBase:

1. **Row key:** Each table in the HBase is indexed on the row key. There are no secondary indices available on the HBase table.
2. **Atomicity:** Avoid designing a table that requires atomicity across all rows. All operations on HBase rows are atomic at row level.
3. **Even distribution:** Read and write should be uniformly distributed across all nodes available in the cluster. Design row key in such a way that, related entities should be stored in adjacent rows to increase read efficacy.

HBase Architecture

HBase follows the master-slave pattern. There will be only one master named HMaster (namenode) and multiple slave nodes named region servers (datanodes).

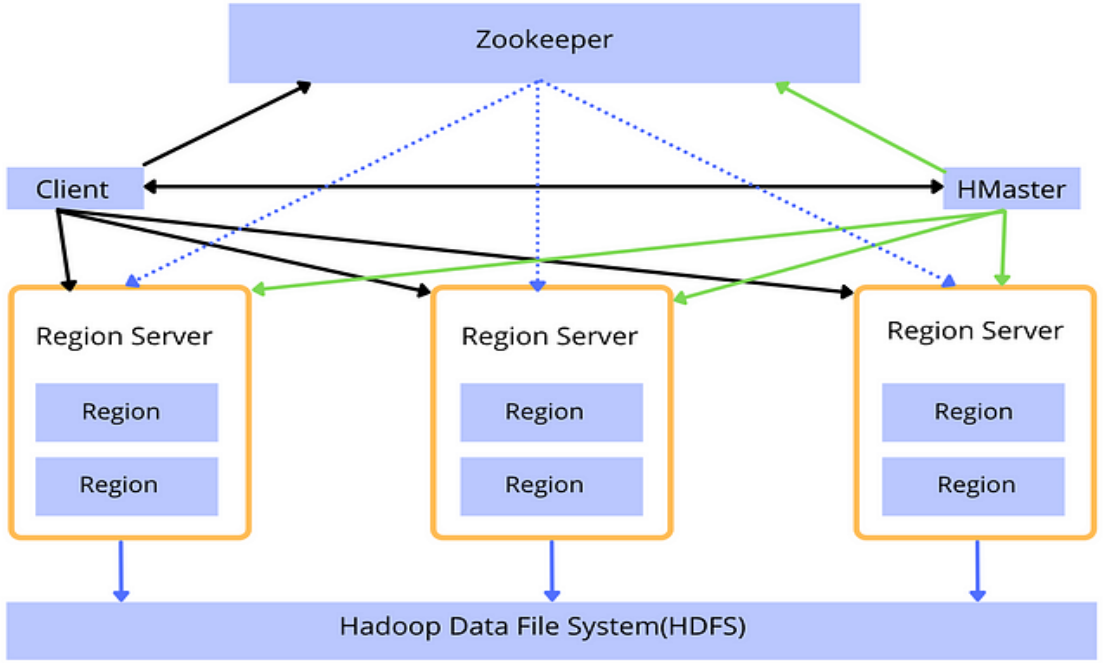
HBase has three major components:

HMaster or Namenode

- It is the master node that handles all region servers within HBase.
- The HMaster is responsible for assigning regions to region servers and performs create, delete, and update operations.
- It also tries to distribute the load across all region servers with the help of Zookeeper.
- If any of the region servers are down, it is responsible for load balancing and recovery actions.

Regions

- As HBase is a distributed database, region servers are responsible for the distributing table keys and managing read/write operation in particular regions.
- Within one region server, there can be multiple regions.
- Table data is split into multiple regions horizontally.
- Each region contains certain ranges of keys, and keys in each region are sorted.
- The default size of each region is 256MB, which can be further configured according to requirements.

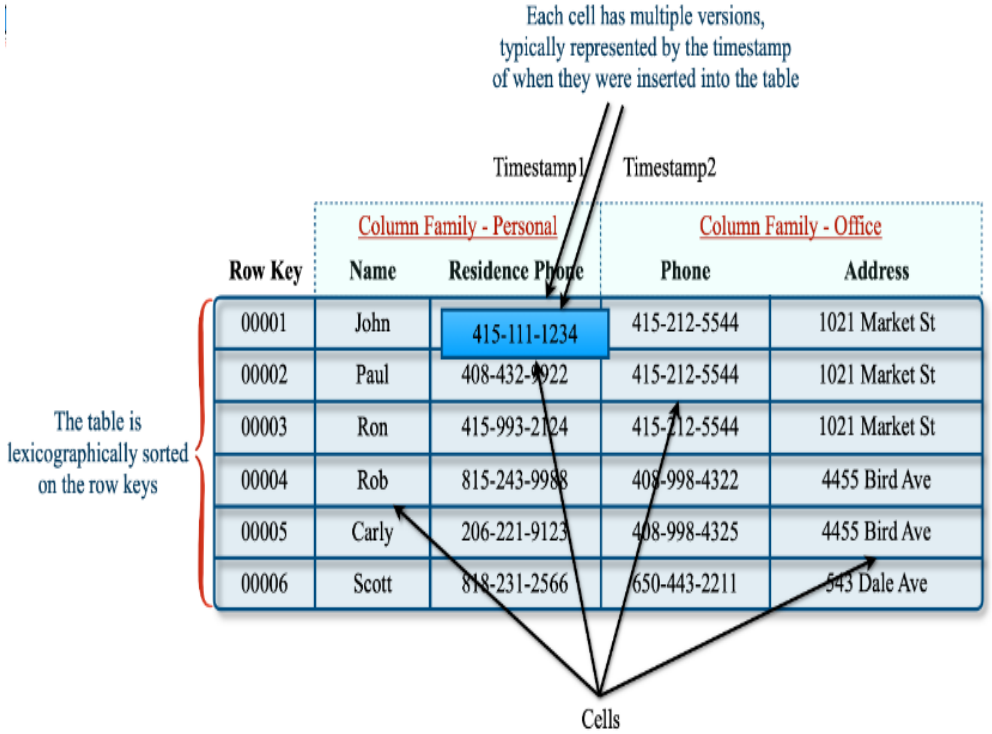


Zookeeper

- It manages the health of all the region servers and the HMaster by maintaining sessions.
- Zookeeper listens to the heartbeat of the HMaster and region servers continuously. By doing so, it makes regular checks for the health of all nodes. In turn, the master node can distribute multiple requests in different region servers.
- HBase cluster maintains two types of master nodes: (i) Active HMaster and (ii) Inactive HMaster.
- When the active master node fails to send the heartbeat, Zookeeper deletes the session of the active master node and creates a new session for the inactive master node, making it active.
- A similar thing happens with region servers. If it fails to send a heartbeat, Zookeeper will expire the session of that region server and inform the master node to perform the recovery action.

Concepts in HBase Schema

- **Table**: HBase organizes data into tables. Table names are Strings and composed of characters that are safe for use in a file system path.
- **Row**: Within a table, data is stored according to its row. Rows are identified uniquely by their row key. Row keys do not have a data type and are always treated as a byte[ ] (byte array).
- **Column Family**: Data within a row is grouped by column family. Column families also impact the physical arrangement of data stored in HBase. For this reason, they must be defined up front and are not easily modified. Every row in a table has the same column families, although a row need not store data in all its families. Column families are Strings and composed of characters that are safe for use in a file system path.
- **Column Qualifier**: Data within a column family is addressed via its column qualifier, or simply, column. Column qualifiers need not be specified in advance. Column qualifiers need not be consistent between rows. Like row keys, column qualifiers do not have a data type and are always treated as a byte[ ].
- **Cell**: A combination of row key, column family, and column qualifier uniquely identifies a cell. The data stored in a cell is referred to as that cell's value. Values also do not have a data type and are always treated as a byte[ ].
- **Timestamp**: Values within a cell are versioned. Versions are identified by their version number, which by default is the timestamp of when the cell was written. If a timestamp is not specified during a write, the current timestamp is used. If the timestamp is not specified for a read, the latest one is returned. The number of cell value versions retained by HBase is configured for each column family. The default number of cell versions is three.



Advance Indexing In HBase

- In HBase, the row key provides the same data retrieval benefits as a primary index. So, when you create a secondary index, use elements that are different from the row key.
- Secondary indexes allow you to have a secondary way to read an HBase table. They provide a way to efficiently access records by means of some piece of information other than the primary key.
- Secondary indexes require additional cluster space and processing because the act of creating a secondary index requires both space and processing cycles to update.
- A method of index maintenance, called Diff-Index, can help IBM® Big SQL to create secondary indexes for HBase, maintain those indexes, and use indexes to speed up queries.

Why Indexing is Important?

- With secondary indexing, I can either find a single row to find all of the rows that contain an attribute with a specific value.
- Like everything in HBase, its stored in sorted order so it becomes rather trivial for the client to fetch several rows and join them in sort order, or to take the intersection if we are trying to find the records that meet a specific qualification. (e.g. find all of Bob's employees who live in Cleveland, OH. Or find the average cost of repairing a Volvo S80 that was involved in a front end collision....)
- As more people want to use HBase like a database and apply SQL, using secondary indexing makes filtering and doing data joins much more efficient. One just takes the intersection of the indexed qualifiers specified, and then apply the unindexed qualifiers as filters further reducing the resulset.

Issues in Indexing

- This design appears to mimic the concept of the column families, where like data is stored in a column family file. So that like data can be accessed quickly. But like the column families, we run in to the same problem... too many column families can be a bad thing when it comes to compaction.
- Depending on the number of columns to be indexed, the size of the index table could easily be larger than the base table. (Especially when you add in geo-spatial indexing.)

Basic HBase Shell Queries

Pig

- Pig is a high-level platform or tool which is used to process large datasets.
- It provides a high level of abstraction for processing over MapReduce.
- It provides a high-level scripting language, known as *Pig Latin* which is used to develop the data analysis codes.
- Pig Latin and Pig Engine are the two main components of the Apache Pig tool.
- The result of Pig is always stored in the HDFS.
- One limitation of MapReduce is that the development cycle is very long. Writing the reducer and mapper, compiling and packaging the code, submitting the job and retrieving the output is a time-consuming task.
- Apache Pig reduces the time of development using the multi-query approach.
- Pig is beneficial for programmers who are not from Java backgrounds.
- 200 lines of Java code can be written in only 10 lines using the Pig Latin language.
- Programmers who have SQL knowledge need less effort to learn Pig Latin.

Applications of Pig

- For exploring large datasets Pig Scripting is used.
- Provides supports across large data sets for Ad-hoc queries.
- In the prototyping of large data-sets processing algorithms.
- Required to process the time-sensitive data loads.
- For collecting large amounts of datasets in form of search logs and web crawls.
- Used where the analytical insights are needed using the sampling.

Execution Modes of Pig

Apache Pig scripts can be executed in three ways :

- **Interactive Mode** (Grunt shell) :

i. You can run Apache Pig in interactive mode using the Grunt shell.

ii. In this shell, you can enter the Pig Latin statements and get the output (using the Dump operator).

- **Batch Mode** (Script) :

i. You can run Apache Pig in Batch mode by writing the Pig Latin script in a single file with the **.pig** extension.

- **Embedded Mode** (UDF) :

i. Apache Pig provides the provision of defining our own functions (**U**ser **D**efined **F**unctions) in programming languages such as Java and using them in our script.

Apache Pig Vs MapReduce

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

Pig Vs SQL

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

Pig Grunt

- Grunt shell is a shell command.
- The Grunt shell of the Apace pig is mainly used to write pig Latin scripts.
- Pig script can be executed with grunt shell which is a native shell provided by Apache pig to execute pig queries.
- We can invoke shell commands using sh and fs.
- Syntax of sh command : grunt> sh ls
- Syntax of fs command : grunt> fs -ls

Pig Latin

- The Pig Latin is a data flow language used by Apache Pig to analyze the data in Hadoop.
- It is a textual language that abstracts the programming from the Java MapReduce idiom into a notation.
- The Pig Latin statements are used to process the data.
- It is an operator that accepts a relation as an input and generates another relation as an output.
- It can span multiple lines.
- Each statement must end with a semi-colon.
- It may include expression and schemas.
- By default, these statements are processed using multi-query execution.

Pig Architecture

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

1. Parser

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

2. Optimizer

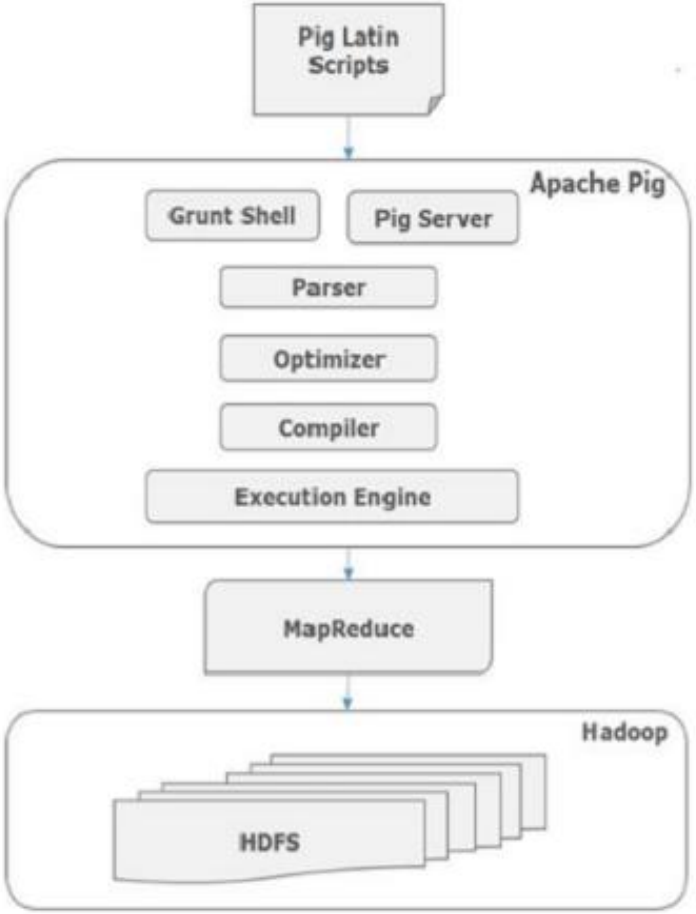
The logical plan (DAG) is passed to the logical optimizer, which carries out the logical optimizations such as projection and pushdown.

3. Compiler

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

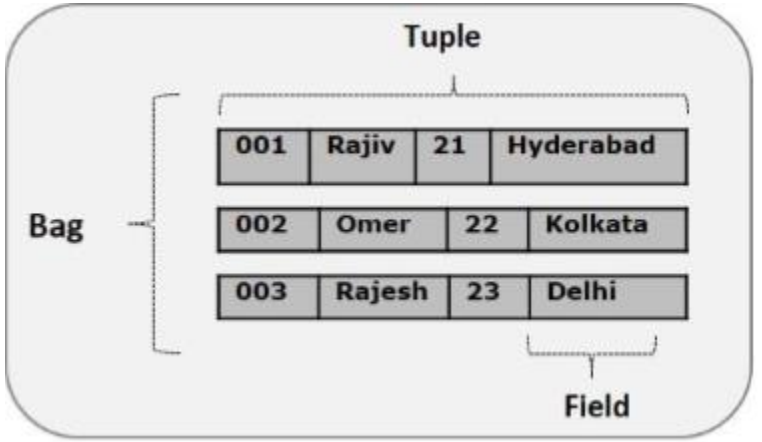
4. Execution Engine

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



Pig Data Model

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



1. Atom

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

Example – 'raja' or '30'

2. Tuple

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

Example – (Raja, 30)

3. Bag

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

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

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

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

4. Map

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

Example – [name#Raja, age#30]

5. Relation

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

ZooKeeper

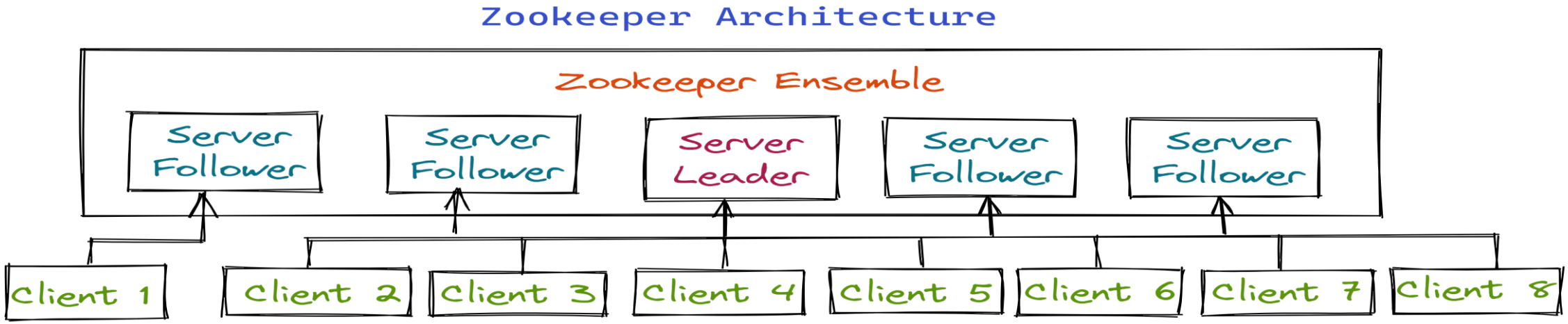
- Apache ZooKeeper is a software project of Apache Software Foundation.
- It is an open-source technology that maintains configuration information and provides synchronized as well as group services which are deployed on Hadoop cluster to administer the infrastructure.
- The ZooKeeper framework was originally built at Yahoo! for easier accessing of applications but, later on, ZooKeeper was used for organizing services used by distributed frameworks like Hadoop, HBase, etc., and Apache ZooKeeper became a standard.
- It was designed to be a vigorous service that enabled application developers to focus mainly on their application logic rather than coordination.
- ZooKeeper is a distributed coordination service that also helps to manage a large set of hosts.
- Managing and coordinating a service especially in a distributed environment is a complicated process, so ZooKeeper solves this problem due to its simple architecture as well as API, that allows developers to implement common coordination tasks like electing a master server, managing group membership, and managing metadata.
- Apache ZooKeeper is used for maintaining centralized configuration information, naming, providing distributed synchronization, and providing group services in a simple interface so that we don’t have to write it from scratch.
- Apache Kafka also uses ZooKeeper to manage configuration.
- ZooKeeper allows developers to focus on the core application logic, and it implements various protocols on the cluster so that the applications need not implement them on their own.

Features of Apache ZooKeeper

Apache ZooKeeper provides a wide range of good features to the user such as:

- **Updating the Node's Status**: Apache ZooKeeper is capable of updating every node that allows it to store updated information about each node across the cluster.
- **Managing the Cluster**: This technology can manage the cluster in such a way that the status of each node is maintained in real time, leaving lesser chances for errors and ambiguity.
- **Naming Service**: ZooKeeper attaches a unique identification to every node which is quite similar to the DNA that helps identify it.
- **Automatic Failure Recovery**: Apache ZooKeeper locks the data while modifying which helps the cluster recover it automatically if a failure occurs in the database.

Zookeeper Architecture



Part	Description
Client	Client node in our distributed applications cluster is used to access information from the server. It sends a message to the server to let the server know that the client is alive, and if there is no response from the connected server the client automatically resends the message to another server.
Server	The server gives an acknowledgement to the client to inform that the server is alive, and it provides all services to clients.
Leader	If any of the server nodes is failed, this server node performs automatic recovery.
Follower	It is a server node which follows the instructions given by the leader.
Ensemble	A cluster or ensemble is a group of Zookeeper servers. When running Apache, you can use ZooKeeper infrastructure in cluster mode to keep the system functioning at its best.

Working of Apache Zookeeper

- The first thing that happens as soon as the ensemble (a group of ZooKeeper servers) starts is, it waits for the clients to connect to the servers.
- After that, the clients in the ZooKeeper ensemble will connect to one of the nodes. That node can be any of a leader node or a follower node.
- Once the client is connected to a particular node, the node assigns a session ID to the client and sends an acknowledgement to that particular client.
- If the client does not get any acknowledgement from the node, then it resends the message to another node in the ZooKeeper ensemble and tries to connect with it.
- On receiving the acknowledgement, the client makes sure that the connection is not lost by sending the heartbeats to the node at regular intervals.
- Finally, the client can perform functions like read, write, or store the data as per the need.

Benefits of Apache ZooKeeper

- **Simplicity:** Coordination is done with the help of a shared hierarchical namespace.
- **Reliability:** The system keeps performing even if more than one node fails.
- **Order:** It keeps track by stamping each update with a number denoting its order.
- **Speed:** It runs with a ratio of 10:1 in the cases where ‘reads’ are more common.
- **Scalability:** The performance can be enhanced by deploying more machines.