Introduction to Apache HBase

What is HBase?

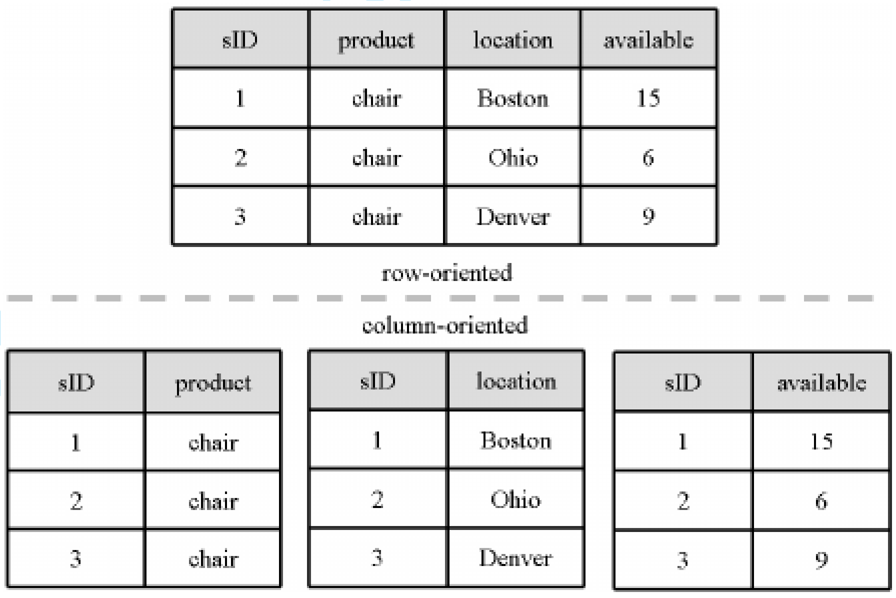
- Hbase is a column-oriented database management system that runs on top of HDFS (Hadoop Distributed File System).
- Initially, it was Google Big Table, afterward; it was renamed as HBase and is primarily written in Java.
- Apache HBase is needed for real-time Big Data applications.
- HBase can store massive amounts of data from terabytes to petabytes.
- The tables present in HBase consist of billions of rows having millions of columns.
- HBase is built for low latency operations, which is having some specific features compared to traditional relational models.

DataBase Type Based on Feature	Example of Database	Use case (When to Use)
Key/ Value	Redis, MemcacheDB	Caching, Queue-ing, Distributing information
Column-Oriented	Cassandra, HBase	Scaling, Keeping Unstructured, non-volatile
Document-Oriented	MongoDB, Couchbase	Nested Information, JavaScript friendly
Graph-Based	OrientDB, Neo4J	Handling Complex relational information. Modeling and Handling classification.

Hive vs Hbase

Features	**Hbase**	**Hive**
Database Model	Wide column store	Relational DBMS
Data Schema	Schema free	With schema
SQL Support	No	Yes, it uses HQL
Partition Methods	Shrading	Shrading
Consistency Level	Immediate consistency	Eventual consistency
Replication Methods	Selectable replication factor	Selectable replication factor

- HBase provides unique features and will solve typical industrial use cases. As column-oriented storage, it provides fast querying, fetching of results, and a high amount of data storage.



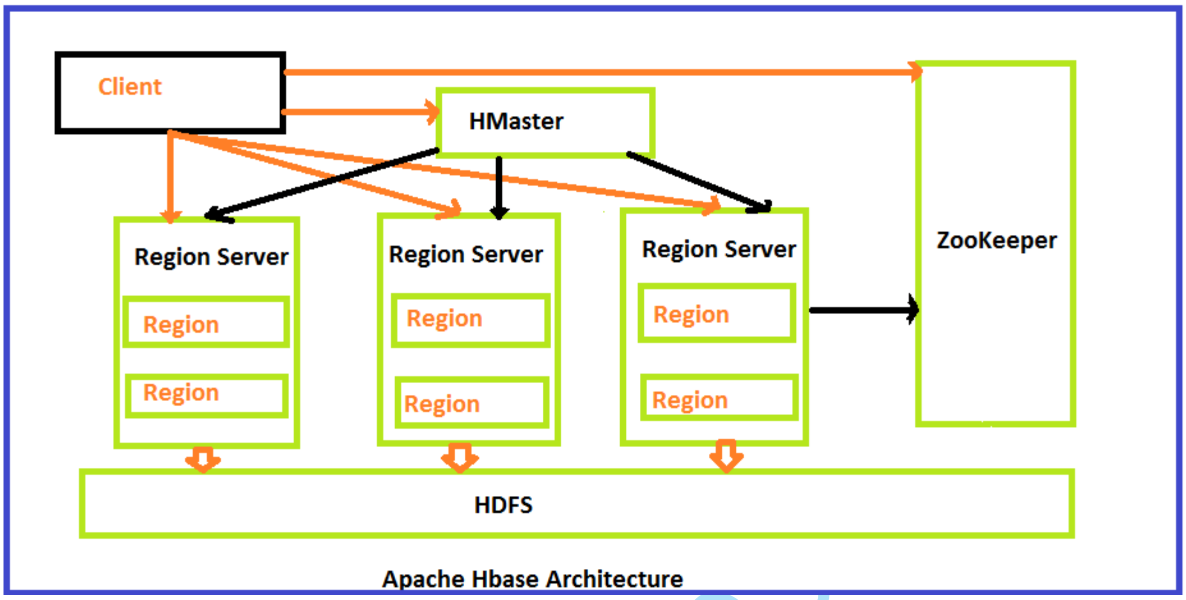
sID	product	location	available
1	chair	Boston	15
2	chair	Ohio	6
3	chair	Denver	9

sID	product
1	chair
2	chair
3	chair

sID	location
1	Boston
2	Ohio
3	Denver

sID	available
1	15
2	6
3	9

HBase Architecture:



Apache Hbase Architecture

- HBase architecture consists mainly of four components:
    - HMaster
    - HRegionserver
    - HRegions
    - Zookeeper
    - HDFS

HMaster

- Plays a vital role in terms of performance and maintaining nodes in the cluster.
- It acts as a monitoring agent to monitor all Region Server instances present in the cluster and acts as an interface for all the metadata changes.
- HMaster assigns regions to region servers.
- Plays a vital role in terms of performance and maintaining nodes in the cluster.
- When a client wants to change any schema and to change any Metadata operations, HMaster takes responsibility for these operations.

HBase Region Servers

- When HBase Region Server receives writes and read requests from the client, it assigns the request to a specific region, where the actual column family resides. However, the client can directly contact with HRegion servers, there is no need of HMaster mandatory permission to the client regarding communication with HRegion servers. The client requires HMaster help when operations related to metadata and schema changes are required.

- HMaster can get into contact with multiple HRegion servers and performs the following functions.
    - Hosting and managing regions
    - Splitting regions automatically
    - Handling read and writes requests
    - Communicating with the client directly

HBase Regions

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

ZooKeeper

- HBase Zookeeper is a centralized monitoring server which maintains configuration information and provides distributed synchronization.
- Distributed synchronization is to access the distributed applications running across the cluster with the responsibility of providing coordination services between nodes.
- If the client wants to communicate with regions, the server's client has to approach ZooKeeper first.

HBase Data Model

- HBase Data Model is a set of components that consists of Tables, Rows, Column families, Cells, Columns, and Versions.
- HBase tables contain column families and rows with elements defined as Primary keys.

HBase Shell Commands

General Commands

Command Name	Syntax	Description
Status	status status 'simple' status 'summary' status 'detailed'	This command will give details about the system status like a number of servers present in the cluster, active server count, and average load value.
Version	version	This command will display the currently used HBase version in command mode.
Table help	table_help	It will provide different HBase shell command usages and its syntaxes.
Whoami	whoami	This command "whoami" is used to return the current HBase user information from the HBase cluster.

Tables Managements commands

These commands will allow programmers to create tables and table schemas with rows and column families.

Command Name	Syntax	Example
Create	create <tablename>, <columnfamilyname>	hbase(main):001:0> create 'education' ,'personal'
List	list	hbase(main):001:0> list
Describe	describe <table name>	hbase(main):010:0>describe 'education'

Disable	disable <tablename>	hbase(main):011:0>disable 'education'
Enable	enable <tablename>	hbase(main):012:0>enable 'education'
Show Filters	show_filters	hbase(main):012:0> show_filters
Drop	drop <table name>	hbase(main):017:0>drop 'education'

Alter	**Syntax**	**Example**
Add a Column Family	alter 'table_name', {NAME => 'new_column_family'}	alter 'mytable', {NAME => 'new_cf'} alter 'edu', 'guru99_1', {NAME => 'guru99_2', IN_MEMORY => true}, {NAME => 'guru99_3', VERSIONS => 5} Explanation: • edu: This is the name of the table being altered, in this case, it's the edu table. • 'guru99_1': This is the name of the column family (guru99_1) within the edu table that is being modified. • {NAME => 'guru99_2', IN_MEMORY => true}: This part of the command is adding or modifying the configuration of the guru99_2 column family within the guru99_1 column family. It sets the IN_MEMORY attribute to true, indicating that the data for this column family should be stored in memory. • {NAME => 'guru99_3', VERSIONS => 5}: This part is adding or modifying the configuration of the guru99_3 column family within the guru99_1 column family. It sets the VERSIONS attribute to 5,

		indicating that a maximum of 5 versions of each cell in this column family should be stored.

Modify a Column Family	alter 'table_name', {NAME => 'column_family', {ATTRIBUTE => 'attribute_name', VALUE => 'new_value'}}
	
	• table_name: The name of the table you want to alter.
	• column_family: The name of the column family you want to modify.
	• ATTRIBUTE: The attribute you want to modify (e.g., TTL, VERSIONS, etc.).
	• attribute_name: The specific attribute within the column family.
	• VALUE: The new value you want to set for the specified attribute.
		alter 'mytable', {NAME => 'cf1', TTL => '2592000'}
		
		alter 'mytable', {NAME => 'cf1', VERSIONS => '3'}
		
		alter 'mytable', {NAME => 'cf1', BLOCKSIZE => '65536'}
		
		alter 'mytable', {NAME => 'cf1', IN_MEMORY => 'true'}
		
		alter 'mytable', {NAME => 'cf1', DATA_BLOCK_ENCODING => 'FAST_DIFF'}

Delete a Column Family	alter 'table_name', {NAME => 'column_family', METHOD => 'delete'}	alter 'mytable', {NAME => 'cf1', METHOD => 'delete'}

Modify Table Name	alter 'old_table_name', NAME => 'new_table_name'	alter 'mytable', NAME => 'newtable'

Note :

- In Apache HBase, when altering a column family, you can modify various attributes to customize the behavior of that column family. Here are some common attributes that you might want to modify:

1. Time-to-Live (TTL): Attribute Name: TTL

   Example: This sets the Time-to-Live for the cf1 column family in the mytable table to 30 days (2592000 seconds).

   alter 'mytable', {NAME => 'cf1', TTL => '2592000'}

2. Maximum Version: Attribute Name: VERSIONS

   Example: This limits the column family cf1 to store a maximum of 3 versions of each cell.

   alter 'mytable', {NAME => 'cf1', VERSIONS => '3'}

3. Block Size: Attribute Name: BLOCKSIZE

   Example: This sets the block size (in bytes) for the cf1 column family to 64 KB.

   alter 'mytable', {NAME => 'cf1', BLOCKSIZE => '65536'}

Data Manipulation commands

Command Name	Syntax	Example
Count	count <'tablename'>, CACHE =>1000	hbase> count 'guru99', CACHE=>1000 • The command will retrieve the count of a number of rows in a table. • Current count is shown per every 1000 rows by default. • Count interval may be optionally specified. • Default cache size is 10 rows.
Put	put <'tablename'>,<'rowname'>,<'columnvalue'>,<'value'>	hbase> put 'guru99', 'r1', 'c1', 'value', 10 • Here we are placing values into table “guru99” under row r1 and column c1. • It will put a cell ‘value’ at defined or specified table or row or column.

Get	get <'tablename'>, <'rowname'>, {< Additional parameters>} Here <Additional Parameters> include TIMERANGE, TIMESTAMP, VERSIONS and FILTERS.	hbase> get 'guru99', 'r1', 'c1' • For table "guru99′ row r1 and column c1 values will display using this command. hbase> get 'guru99', 'r1', {TIMERANGE => [ts1, ts2]} • For table "guru99″row 1 values in the time range ts1 and ts2 will be displayed using this command.
Delete	delete <'tablename'>,<'row name'>,<'column name'>	hbase(main):)020:0> delete 'guru99', 'r1', 'c1". • The above execution will delete row r1 from column family c1 in table "guru99."
Truncate	truncate <tablename>	hbase>truncate 'edu' • After truncate of an hbase table, the schema will

		present but not the records.
Scan	scan <'tablename'>, {Optional parameters}	hbase>scan 'guru99' • It shows “guru99” table with column name and values • It consists of three row values r1, r2, r3 for single column value c1 • It displays the values associated with rows.

HBase Create Table with Java API

Question

Using Apache HBase, answer the following:

1. Create a table with the name 'Student' by specifying 2 column families 'subjects' and 'address'.

2. Add 3 columns under 'subjects' column family (subject1,subject2 and subject3) and add corresponding values to it. Add 2 columns under 'address' column family (city and state) and add corresponding values.

3. Show how one can fetch data of city column.

4. Add new column family 'personal'.

5 . Drop table 'student.

- create 'Student', 'subjects', 'address'

- put 'Student', '1', 'subjects:subject1', 'Math'

put 'Student', '1', 'subjects:subject2', 'Science'

put 'Student', '1', 'subjects:subject3', 'History'

put 'Student', '1', 'address:city', 'ExampleCity'

put 'Student', '1', 'address:state', 'ExampleState'

- get 'Student', '1', {COLUMN=>'address:city'}

- alter 'Student', {NAME=>'personal', VERSIONS=>1}

- disable 'Student'

drop 'Student'