HBASE

Ques: HBASE and HDFS

HDFS	HBASE
It is a distributed file system suitable for storing large files	It is a database built on top of the HDFS.
It does not support fast single record lookups.	It offers fast lookups for larger tables.
It provides high latency batch processing; no concept of batch processing.	It provides low latency access to individual rows of billion of records ~~of fast lugang~~
It only provides sequential access of data.	HBase internally uses hashed table and provide random access, and it stores the data in indexed HDFS files for faster lookups.

Ques: Storage mechanism in HBase.

HBase is a column oriented database and the tables in it are sorted by rank. The table schema only defines column families, which are the key-value pairs. A table can have multiple columns families and each column family can have any number of column Then column values are stored contiguously on the d Each cell value of the table is timestamped. In short, in an HBase:

- Tables is a collection of lines
- Series is a collection of column families.
- column family is a collection of columns.
- column is a collection of key-value pairs.

Below is an example of table in HBase.

Rowid	Column family			Column family			Column family		
	col1	col2	col3	col1	col2	col3	col1	col2	col3

Ques: HBase and RDBMS

HBase	RDBMS
HBase is less schematic, it does not have the concept of fixed column schema. ~~it does not have the concept~~ only defines columns families.	An RDBMS is governed by its schema, which describes the entire structure of the tables.
It is built for wide tables. HBase is scalable horizontally.	It is thin and designed for small tables. Hard to scale.
There are no transactions in HBase.	RDBMS is transactional.
It has de-normalized data	It will have normalized data.
It is good for semi-structured as well as structured data.	It's good for structured data.

Properties of HBase

- HBase is linearly scalable.
- It has automatic failure support
- It provides consistent read and writes.
- It integrates with Hadoop, both as a source and a target
- It has simple Java API for client
- It provides data replication through clusters.

Application of HBase

- It is used when it is necessary to write heavy applicati
- HBase is used when we need to quickly provide random access to available data.
- Companies like Facebook, Twitter, Yahoo and Adobe use HBase internally.

Ques: Where to use HBase?

- Apache HBase is used to have random, real-time read/write access to large data.
- It houses very large tables on top of clusters of standard hardware.
- Apache HBase is a non-relational database modeled after the google Bigtable. Bigtable acts on top of google file system, also Apache HBase works on top of Hadoop and HDFS.

HBASE - SHELL

Ques: HBase Shell

HBase contains a shell using which it can communicate with HBase. HBase uses the Hadoop file system to store its data. It will have a master server and region server. The data store will be in the form of region. These regions are sharing up and being stored in the region server.

The master server manages these region servers and all of these tasks take place on HDFS. Below are some of the commands supported by HBase Shell.

Ques: General commands

* status - Provides the status HBase, for example the number servers.
- Version - Provides the version HBase utility used.
- table_help - Provide help for table reference commands.
- whoami - Provides information about the user.

Data definition language

create - Creates a table
list - List all tables in HBase
disable - Disables a table
is_disabled - Verifies if a table is disabled
is enable - Enable a table
is_enabled - Verifies if a table is possible.
describe - Provides the description of a table
age - Changes a table
exists - Verifies if a table exists.
drop - Drop a table of HBase
drop_all - Drop the tables fitting the "regex" given in the command.

Java Admin API - Prior to all the above commands, Java offers an Admin API to achieve DDL functionalities through programming. Under org.apache.hadoop.hbase.client package, HBase Admin and HTable Descriptors are 2 major class in this package that provide DDL functionalities

HBase – Overview of Architecture and Data Model

Introduction

HBase is a column-oriented database that's an open-source implementation of Google's Big Table storage architecture. It can manage structured and semi-structured data and has some built-in features such as scalability, versioning, compression and garbage collection.

Since its uses write-ahead logging and distributed configuration, it can provide fault-tolerance and quick recovery from individual server failures. HBase built on top of Hadoop / HDFS and the data stored in HBase can be manipulated using Hadoop's MapReduce capabilities.

Let's now take a look at how HBase (a column-oriented database) is different from some other data structures and concepts that we are familiar with **Row-Oriented vs. Column-Oriented data stores.** As shown below, in a row-oriented data store, a row is a unit of data that is read or written together. In a column-oriented data store, the data in a column is stored together and hence quickly retrieved.

Row ID	Customer	Product	Amount
101	John White	Chairs	$400.00
102	Jane Brown	Lamps	$500.00
103	Bill Green	Lamps	$150.00
104	Jack Black	Desk	$700.00
105	Jane Brown	Desk	$650.00
106	Bill Green	Desk	$900.00

Row-oriented data stores –

- Data is stored and retrieved one row at a time and hence could read unnecessary data if only some of the data in a row is required.
- Easy to read and write records
- Well suited for OLTP systems
- Not efficient in performing operations applicable to the entire dataset and hence aggregation is an expensive operation
- Typical compression mechanisms provide less effective results than those on column-oriented data stores

Column-oriented data stores –

- Data is stored and retrieved in columns and hence can read only relevant data if only some data is required
- Read and Write are typically slower operations
- Well suited for OLAP systems
- Can efficiently perform operations applicable to the entire dataset and hence enables aggregation over many rows and columns
- Permits high compression rates due to few distinct values in columns

Introduction Relational Databases vs. HBase

When talking of data stores, we first think of Relational Databases with structured data storage and a sophisticated query engine. However, a Relational Database incurs a big penalty to improve performance as the data size increases. HBase, on the other hand, is designed from the ground up to provide scalability and partitioning to enable efficient data structure serialization, storage and retrieval. Broadly, the differences between a Relational Database and HBase are:

Relational Database –

- Is Based on a Fixed Schema
- Is a Row-oriented datastore
- Is designed to store Normalized Data
- Contains thin tables
- Has no built-in support for partitioning.

HBase –

- Is Schema-less
- Is a Column-oriented datastore
- Is designed to store Denormalized Data
- Contains wide and sparsely populated tables
- Supports Automatic Partitioning

HDFS vs. HBase

HDFS is a distributed file system that is well suited for storing large files. It's designed to support batch processing of data but doesn't provide fast individual record lookups. HBase is built on top of HDFS and is designed to provide access to single rows of data in large tables. Overall, the differences between HDFS and HBase are

HDFS –

- Is suited for High Latency operations batch processing
- Data is primarily accessed through MapReduce
- Is designed for batch processing and hence doesn't have a concept of random reads/writes

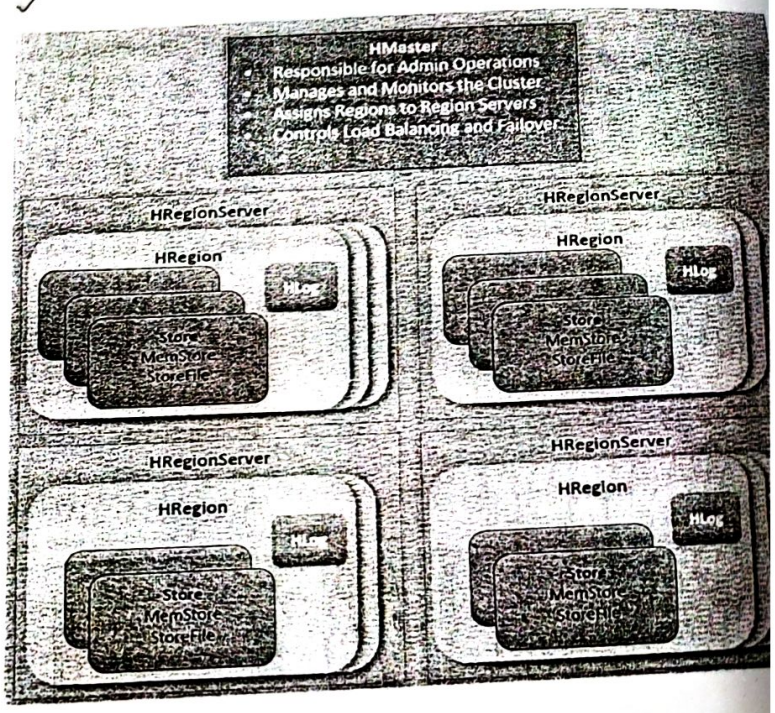
HBase –

- Is built for Low Latency operations
- Provides access to single rows from billions of records
- Data is accessed through shell commands, Client APIs in Java, REST, Avro or Thrift

HBase Architecture

The HBase Physical Architecture consists of servers in a Master-Slave relationship as shown below. Typically, the HBase cluster has one Master node, called HMaster and multiple Region Servers called HRegionServer. Each Region Server contains multiple Regions – HRegions.

Just like in a Relational Database, data in HBase is stored in Tables and these Tables are stored in Regions. When a Table becomes too big, the Table is partitioned into multiple Regions. These Regions are assigned to Region Servers across the cluster. Each Region Server hosts roughly the same number of Regions.



The HMaster in the HBase is responsible for

- Performing Administration
- Managing and Monitoring the Cluster
- Assigning Regions to the Region Servers
- Controlling the Load Balancing and Failover

On the other hand, the HRegionServer perform the following work

- Hosting and managing Regions
- Splitting the Regions automatically
- Handling the read/write requests
- Communicating with the Clients directly

Each Region Server contains a Write-Ahead Log (called HLog) and multiple Regions. Each Region in turn is made up of a MemStore and multiple StoreFiles (HFile). The data lives in these StoreFiles in the form of Column Families (explained below). The MemStore holds in-memory modifications to the Store (data).

The mapping of Regions to Region Server is kept in a system table called .META. When trying to read or write data from HBase, the clients read the required Region information from the .META table and directly communicate with the appropriate Region Server. Each Region is identified by the start key (inclusive) and the end key (exclusive)

HBase Data Model

The Data Model in HBase is designed to accommodate semi-structured data that could vary in field size, data type and columns. Additionally, the layout of the data model makes it easier to partition the data and distribute it across the cluster. The Data Model in HBase is made of different logical components such as Tables, Rows, Column Families, Columns, Cells and Versions.

Customer Id	Name	City	Product	Amount
101	John White	Los Angeles, CA	Chairs	$400.00
102	Jane Brown	Atlanta, GA	Lamps	$200.00
103	Bill Green	Pittsburgh, PA	Desk	$500.00
104	Jack Black	St. Louis, MO	Bed	$1600.00

Column Families

Tables – The HBase Tables are more like logical collection of rows stored in separate partitions called Regions. As shown above, every Region is then served by exactly one Region Server. The figure above shows a representation of a Table.

Rows – A row is one instance of data in a table and is identified by a *rowkey*. Rowkeys are unique in a Table and are always treated as a byte[].

Column Families – Data in a row are grouped together as Column Families. Each Column Family has one more Columns and these Columns in a family are stored together in a low level storage file known as HFile. Column Families form the basic unit of physical storage to which certain HBase features like compression are applied. Hence it's important that proper care be taken when designing Column Families in table.

The table above shows Customer and Sales Column Families. The Customer Column Family is made up 2 columns – Name and City, whereas the Sales Column Families is made up to 2 columns – Product and Amount.

Columns – A Column Family is made of one or more columns. A Column is identified by a Column Qualifier that consists of the Column Family name concatenated with the Column name using a colon – example: columnfamily:columnname. There can be multiple Columns within a Column Family and Rows within a table can have varied number of Columns.

Cell – A Cell stores data and is essentially a unique combination of *rowkey*, Column Family and the Column (Column Qualifier). The data stored in a Cell is called its value and the data type is always treated as byte[].

Version – The data stored in a cell is versioned and versions of data are identified by the timestamp. The number of versions of data retained in a column family is configurable and this value by default is 3.

HBase is a NoSQL database commonly referred to as the Hadoop Database, which is open-source and is based on Google's Big Table white paper. HBase runs on top of the Hadoop Distributed File System (HDFS), which allows it to be highly scalable, and it supports Hadoop's map-reduce programming model. HBase permi two types of access: random access of rows through their row keys and offline or batch access through map-reduce queries.