CSE 4/587
Data Intensive Computing

Dr. Eric Mikida
epmikida@buffalo.edu
208 Capen Hall

Dr. Shamshad Parvin
shamsadp@buffalo.edu
313 Davis Hall

References

https://cse.buffalo.edu/~bina/cse487/spring2018/

https://lintool.github.io/MapReduceAlgorithms/MapReduce-book-final.pdf

https://static.googleusercontent.com/media/research.google.com/en//archive/mapreduce-osdi04.pdf

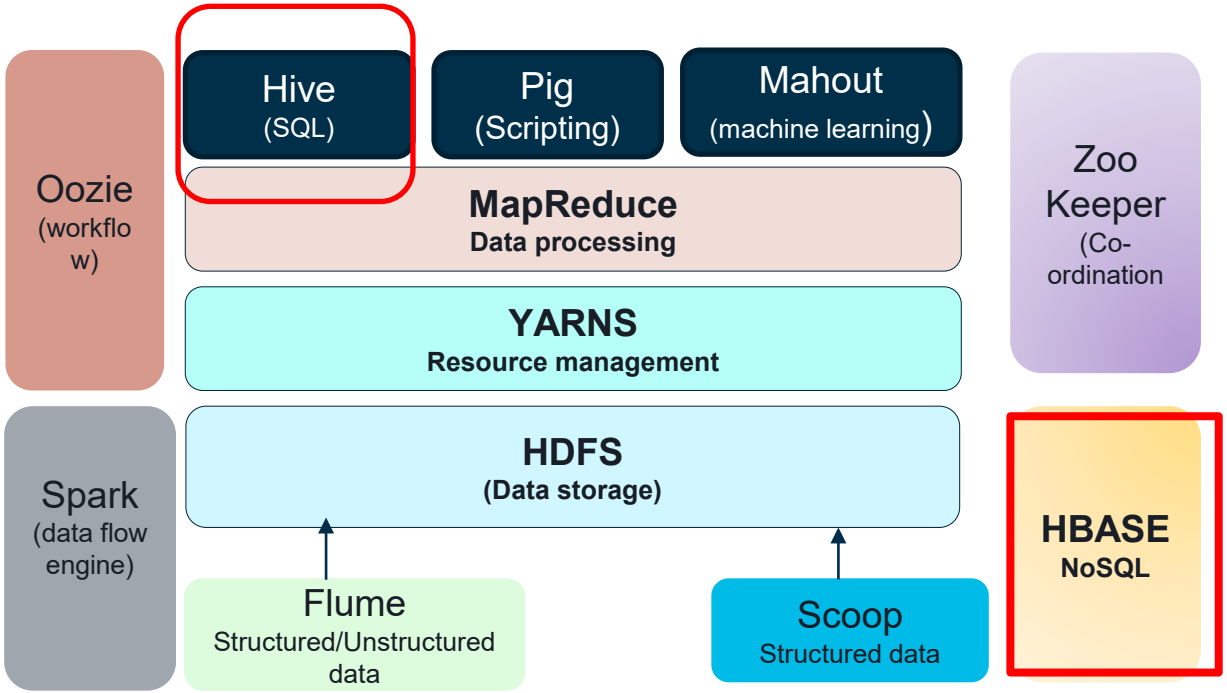
https://mapr.com/blog/in-depth-look-hbase-architecture/

https://data-flair.training/blogs/

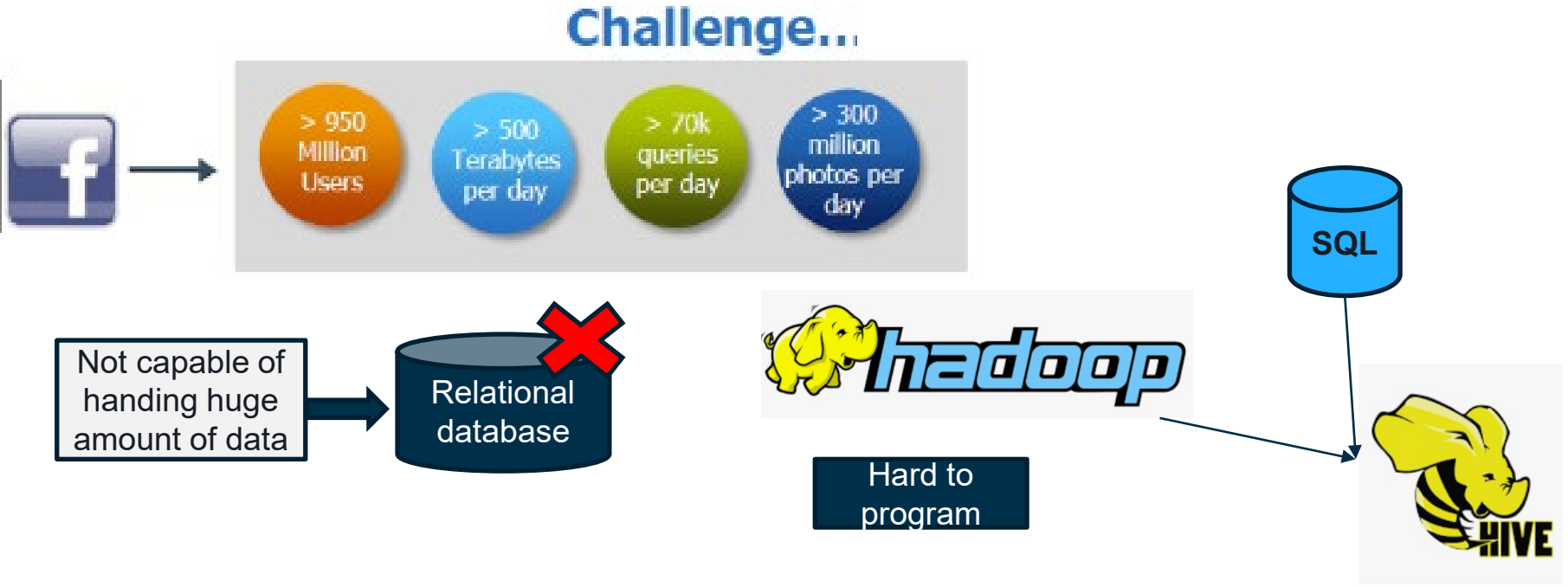
https://www.edureka.co/blog/

https://www.simplilearn.com/

Apache Ecosystem



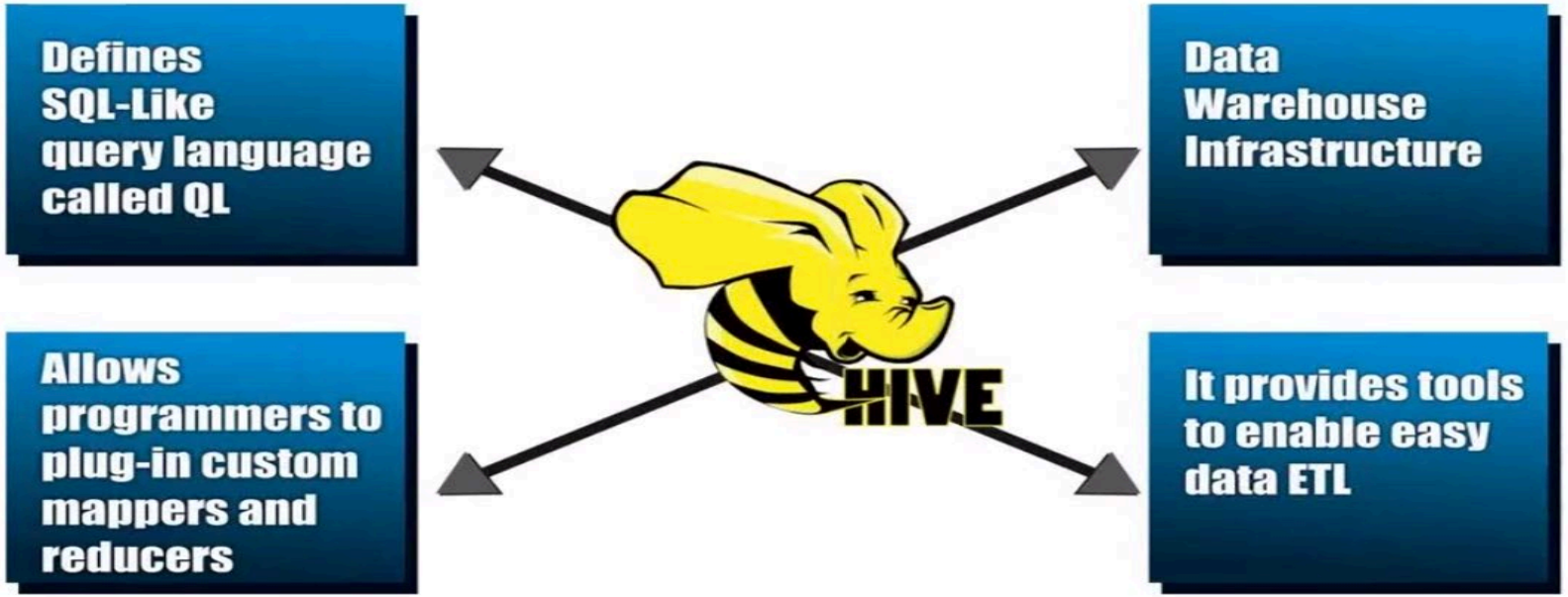
Why HIVE?


Challenge...
> 950 Million Users
> 500 Terabytes per day
> 70k queries per day
> 300 million photos per day
Not capable of handing huge amount of data
Relational database
hadoop
Hard to program
SQL
HIVE

What is HIVE?

- Hive, an open source peta-byte scale date warehousing framework based on Hadoop, was developed by the Data Infrastructure Team at Facebook.
- Apache Hive is a distributed, fault-tolerant data warehouse system built on top of Hadoop to provide easy data storage and analysis .
- Hive supports queries in a SQL-like declarative language (**HiveQL**), which are compiled into MapReduce jobs that are executed using Hadoop

Hive Key Principles



Advantages of Hive

- Useful for people who aren't from a programming background as it eliminates the need to write complex MapReduce program.
- **Extensible** and **scalable** to cope up with the growing volume and variety of data, without affecting performance of the system.
- It is as an efficient ETL (Extract, Transform, Load) tool.
- Hive supports any client application written in Java, PHP, Python, C++ or Ruby by exposing its **Thrift server**

Hive

- With Hive, now the following task can be performed:
- Tables can be portioned and bucketed
- Schema flexibility and evolution
- Hive tables can be defined directly in the HDFS
- Extensible – Types, Formats, Functions and scripts

Where to Use Hive?

- Apache Hive can be used in the following places:
- Data Mining
- Log Processing
- Document Indexing
- Customer Facing Business Intelligence
- Predictive Modelling

HIVE Architecture

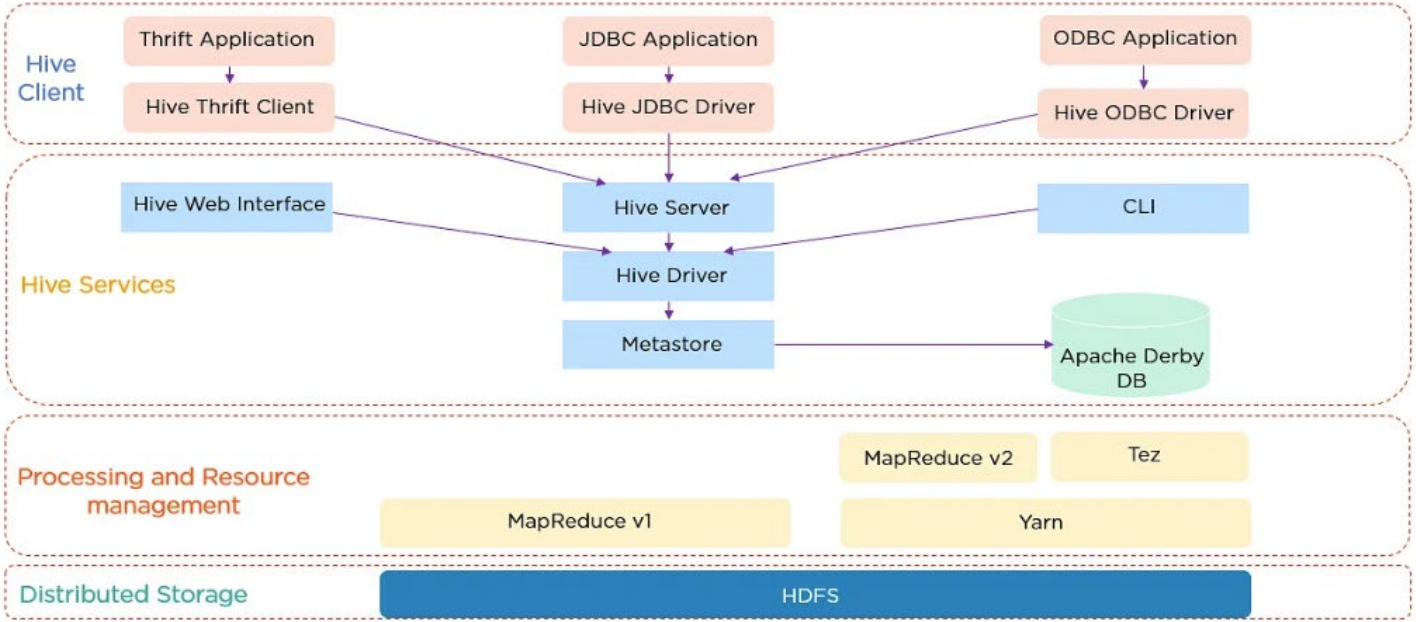
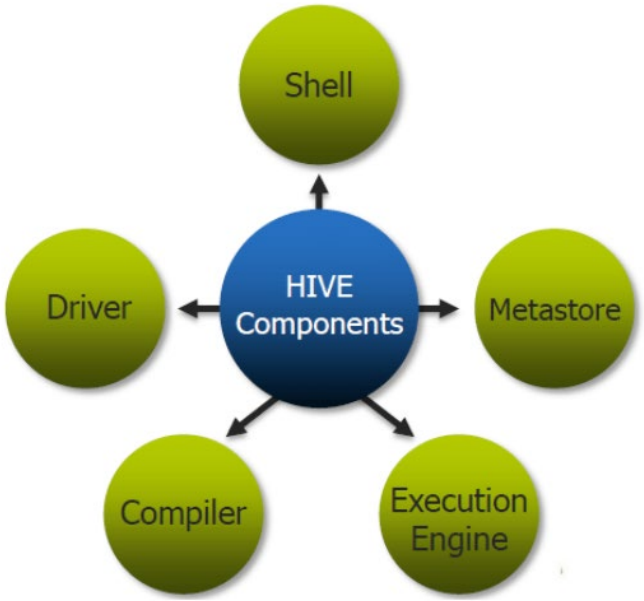


Fig: Architecture of Hive

HIVE Components

Major Components:

- Metastore
- Driver
- Compiler
- Executer/engine
- CLI, UI and Thrift server



Metastore

Metastore acts as the system catalog for Hive

- Stores information about the tables, partitions, schemas, columns, etc.
- It stores the data in a traditional RDBMS format

Without Metastore, it is not possible to impose a structure on hadoop files

- Information stored in the Metastore must be backed up regularly
- Backup server replicates the data, and retrieves it in case of data loss

Driver and Compiler

The **Driver** is the component that manages the lifecycle of a HiveQL statement as it moves through Hive.

- Driver also maintains a session handle and any session statistics

The **Compiler** performs the compilation of the HiveQL query

- The metadata stored in the Metastore is used by the compiler to generate the execution plan

Compiler

Similar to compilers in traditional databases, the Hive compiler processes HiveQL statements in the following steps:

1. Convert the query to an Abstract Syntax Tree (AST)
2. Check for compatibility and compile-time errors
3. Convert the AST to a Directed Acyclic Graph (DAG)
4. Execute tasks in order of their dependencies
    a. Each task is only executed if all of its pre-reqs have been executed

Executor

The **Executor** executes tasks in the order of their dependencies

1. An MR task first serializes its part of the plan into a plan.xml file.
2. This file is added to the job cache for the task and instances of ExecMapper and ExecReducers are spawned using Hadoop
3. Each of these classes deserializes the plan.xml and executes the relevant part of the operator DAG

CLI, UI, and Thrift Server - provide a user interface for an external user to interact with Hive. Thrift server in Hive allows external clients to interact with Hive over a network, similar to the JDBC or ODBC protocols:

Client Components

CLI, **UI**, and **Thrift Server** each provide a user interface for an external user to interact with Hive

- Thrift Server in Hive allows external clients to interact with Hive over a network, similar to the JDBC or ODBC protocols.

Hive Data Model

Data in Hive organized into :

- Tables
- Partitions
- Buckets

Hive Data Model Contd.

- Tables

- Analogous to relational tables
  - Each table has a corresponding directory in HDFS
  - Data serialized and stored as files within that directory

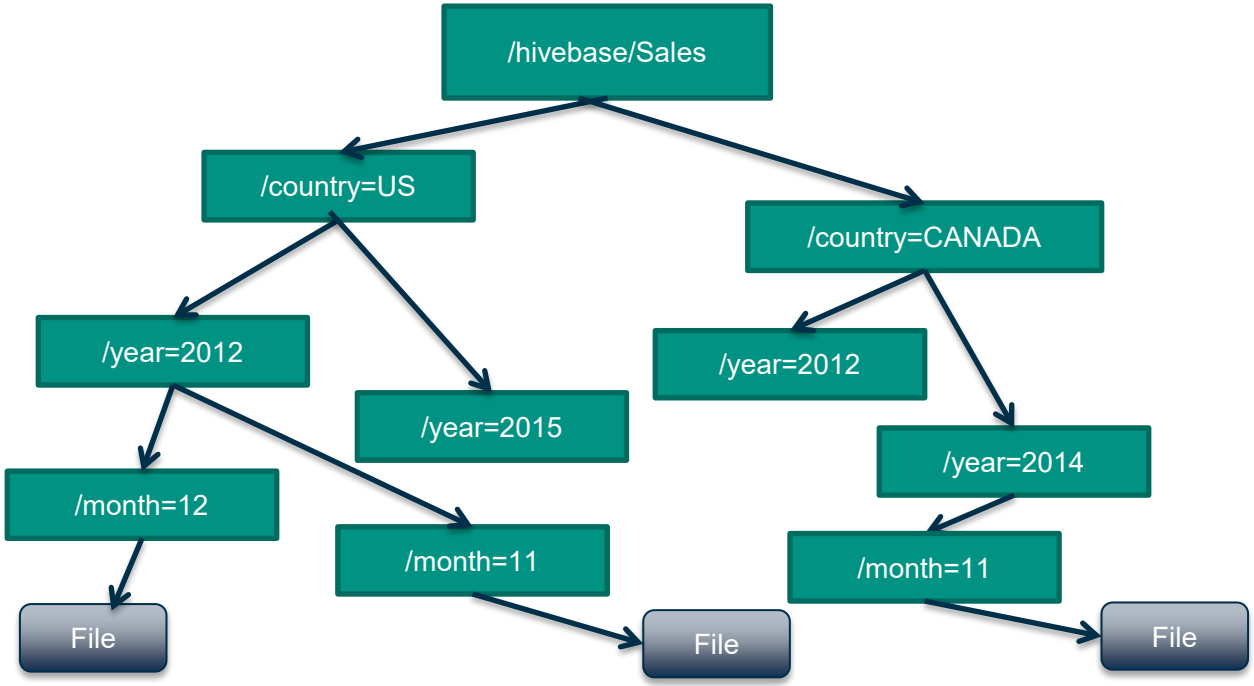
- - Users can specify custom serialization –deserialization schemes (**SerDe's**)

Hive Data Model Contd.

- Partitions
- Each table can be broken into partitions
- Partitions determine distribution of data within subdirectories

Example -
CREATE_TABLE Sales (sale_id INT, amount FLOAT)
PARTITIONED BY (country STRING, year INT, month INT)
So each partition will be split out into different folders like
Sales/country=US/year=2012/month=12

Hierarchy of Hive Partitions



Hive Data Model Contd.

- Buckets
- Data in each partition divided into buckets
- Based on a hash function of the column
- **H(column) mod NumBuckets = bucket number**
- Each bucket is stored as a file in partition directory

HIVE Data types

Hive structures data into well-understood database concepts
ie: tables, rows, columns, partitions

It supports primitive types:

- Integers (signed) – bigint(8 bytes), int(4 bytes), smallint(2 bytes), tinyint(1 byte).
- Floating point numbers – float(single precision), double(double precision)
- String

Hive also supports:

- Associative arrays: map<key-type, value-type>
- Lists: list<element type>
- Structs: struct<file name: file type…>

SerDe: ser*ialize and* ***de****serialize*
API is used to move data in
and out of tables

Query Language

HiveQL comprises of a subset of **SQL** and some extensions

From SQL: from clause subqueries, inner, left outer, right outer and outer joins, cartesian products, group by and aggregations, union all, create table, select and many functions on primitive and complex types ***make the language very SQL like***

MetaData browsing capabilities like show tables and describe are also present

Query Language

HiveQL comprises of a subset of **SQL** and some extensions

From SQL: from clause subqueries, inner, left outer, right outer and outer joins, cartesian products, group by and aggregations, union all, create table, select and many functions on primitive and complex types ***make the language very SQL like***

MetaData browsing capabilities like show tables and describe are also present

This enables anyone familiar with SQL to start a hive cli(command line interface) and begin querying the system right away

HiveQL

DDL :

- **CREATE DATABASE**
- **CREATE TABLE**
- **ALTER TABLE**
- **UPDATE TABLE**

DML:

- **LOAD TABLE**
- **INSERT**

QUERY:

- **SELECT**
- **GROUP BY**
- **JOIN**
- **UNION**

Limitations of Hive:

- Not designed for online transaction processing.
- Provides acceptable latency for interactive data browsing.
- Does not offer real-time queries and row level updates.
- Latency for Hive queries is generally very high.

HBase

- Open-source implementation of Google's **Big Table**
  - A ton of semi-structured data is created every day
- **Apache HBase** created as part of the Hadoop project
  - It is built on top of HDFS
  - Provides fault-tolerant way of storing large quantities of sparse data
  - It is a distributed non relational database

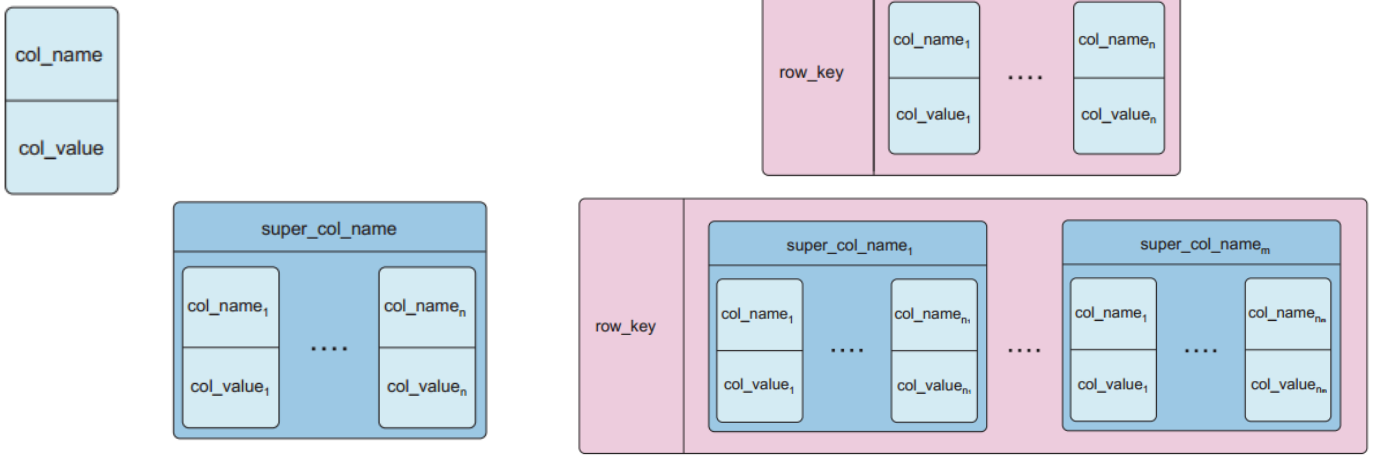
HBase

HBase is a column-oriented database

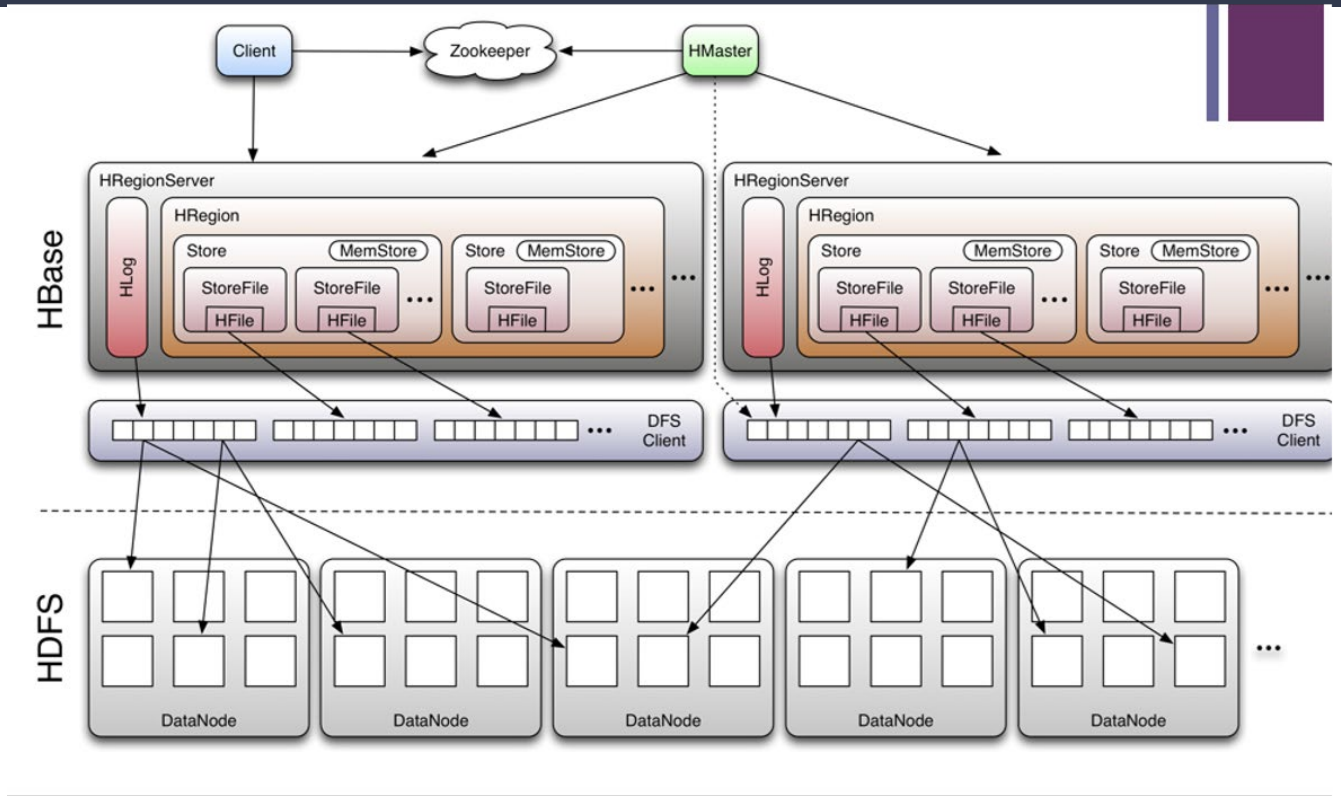
- Column-oriented databases store records in a sequence of columns i.e. the entries in a column are stored in contiguous locations on disks

HBase

- Based on Google's BigTable paper
- Like column oriented relational databases (store data in column order) but with a twist
- Tables similarly to RDBMS, but handle semi-structured
- Data model:
  - Collection of Column Families
  - Column family = (key, value) where value = set of **related** columns (standard, super)
  - indexed by *row key*, *column key* and *timestamp*



HBase Architecture



HBase Architecture

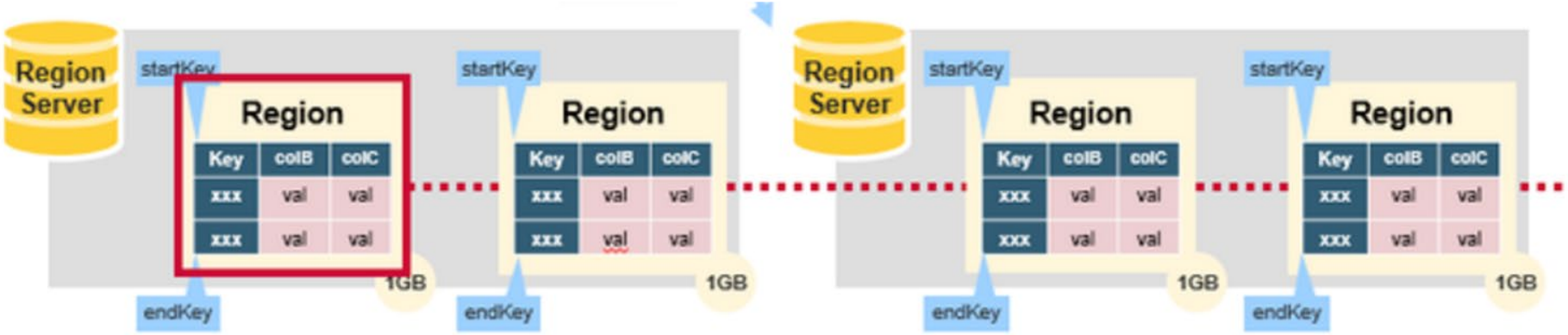
The three main components of Hbase architecture are:

- HMaster server
- Region server
- Regions

HBase Tables are divided horizontally by row key range into ***Regions***

- A region contains all rows in the table between the region's start and end key
- Regions are assigned to the nodes in the cluster, called ***Region Servers***, and these serve data for reads and writes

HBase Architecture



- A **Region** has a default size of 256MB (can be configured based on need)
- A group of regions is served to the clients by a **Region Server**
- A Region Server can serve approximately 1000 regions to the client.

HBase Architecture

The Hadoop **DataNode** stores the data that the Region Server is managing

- All HBase data is stored in HDFS files
- Region Servers are collocated with the HDFS DataNodes
    - Enables data locality (putting the data close to where it is needed)

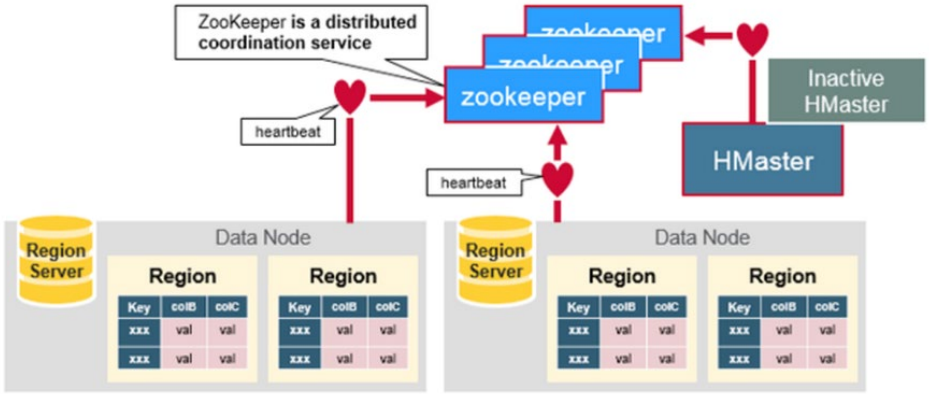
HMaster is responsible for region assignment, coordinating the region servers, and re-assigning regions for recovery or load balancing

- Monitors all RegionServer instances in the cluster (listens for notifications from ***ZooKeeper***)
- It is also responsible for creating, deleting, and updating tables

HBase Architecture

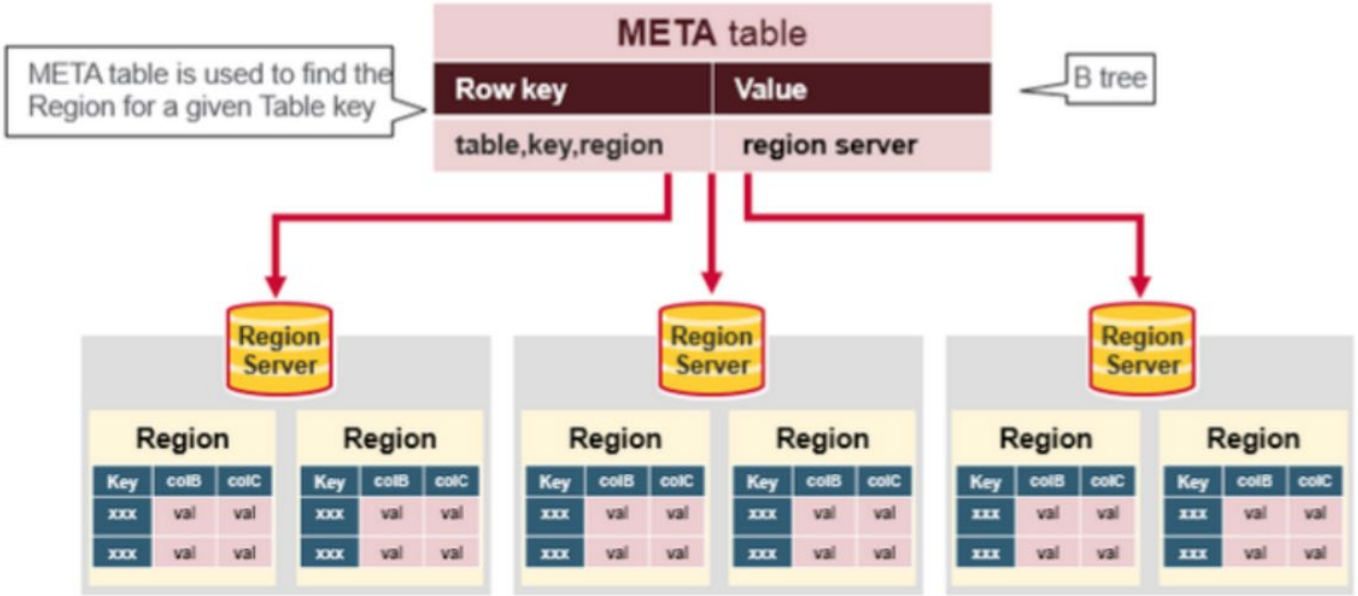
HBase uses **ZooKeeper** as a distributed coordination service to maintain server state in the cluster

- ZooKeeper maintains which servers are alive and available, and provides server failure notification.
- Zookeeper uses consensus to guarantee common shared state.



HBase Architecture

The **META table** is an HBase table that keeps a list of all regions in the system



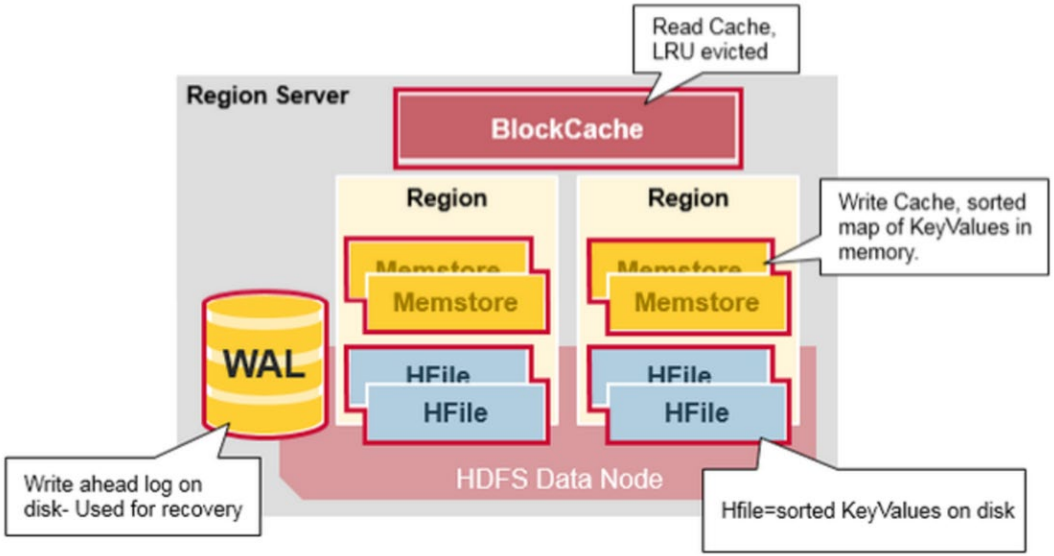
HBase Architecture

The RegionServer has 4 main components:

- **Write Ahead Log (WAL)** is a file on the DFS used to store data that hasn't yet been persisted to permanent storage; used for recovery in the case of failure.
- **BlockCache** is the read cache
  - Stores frequently read data in memory
  - Least Recently Used (LRU) data is evicted when full
- **MemStore** is the write cache
  - Stores new data which has not yet been written to disk
  - Data gets sorted before writing to disk
  - One MemStore per column family per region
- **HFiles** store the rows as sorted KeyValues on disk

HBase Architecture

RegionServer Architecture



HBase Summary

- Scales automatically
- Integrated with HDFS and Hadoop
- Capability of handling semi-structured data
- Provides fast random access to available data
- Provides JAVA and other APIs