CSE 4/587
Data Intensive Computing

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

Day 19
HIVE

Announcements and Feedback

- Phase 2 deadline extended to Mon 11/14 @ 11:59pm
  - Phase 1 feedback will be returned today
- Midterm Exam papers will be returned today
  - Sayantan graded Question 1
  - Smarana graded Question 2
  - Enjamamul graded Question 5
  - All other questions can come to me

Hive

- Size of data sets being analyzed in industry is growing rapidly; traditional warehousing solutions prohibitively expensive
- Hadoop/MR is being used in companies like Yahoo, Facebook etc. to store and process extremely large data sets on commodity hardware
- MR programming model is very low level and requires developers to write custom programs which are hard to maintain and reuse
- **Hive** is an open-source data warehousing solution built on top of Hadoop
- Hive supports queries in a SQL-like declarative language (**HiveQL**), which are compiled into MapReduce jobs that are executed using Hadoop

HiveQL and MetaStore

- The **HiveQL** language includes:
  - a type system with support for tables containing primitive types
  - collections like arrays and maps, and nested compositions of the same
  - Support in I/O libraries to query data in custom formats
  - Enables users to plug in custom MR scripts into queries
- Hive's system catalog, **Metastore**, contains schemas and statistics
  - Useful in data exploration, query optimization and query compilation
- In Facebook, the Hive warehouse contains tens of thousands of tables

Data Models

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…>

Data Models

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***

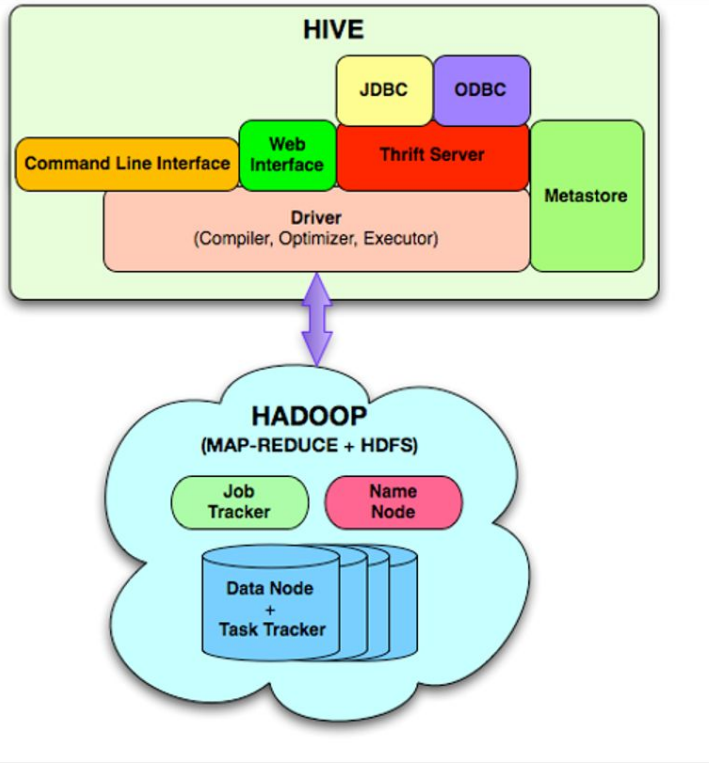
MetaData browsing capabilities like show tables and describe are also present

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

HIVE Architecture

Major Components:

- Metastore
- Driver
- Compiler
- Executer
- 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; each task is only executed if all of its pre-reqs have been executed.

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 Use Cases

- Hive is useful for ***summarizing data***, and ***analytics***
- Hive is ***scalable*** and Hive queries are ***simple*** since it is similar to SQL
- Hive is useful for processing ***structured data stored in Hadoop***

- Hive is not good for online transactions
  - Latency of Apache Hive queries are high

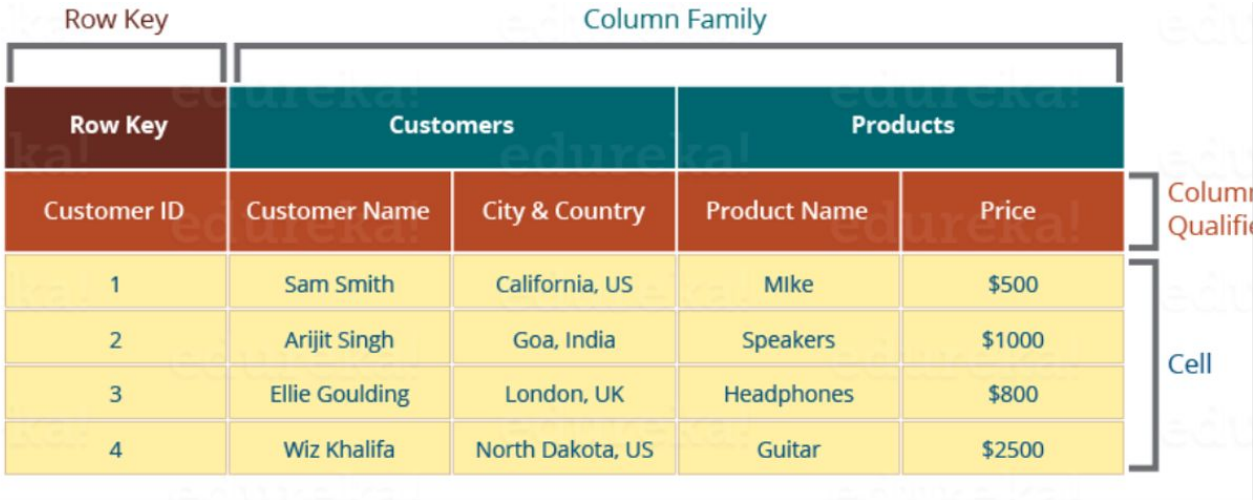
HBase

- 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

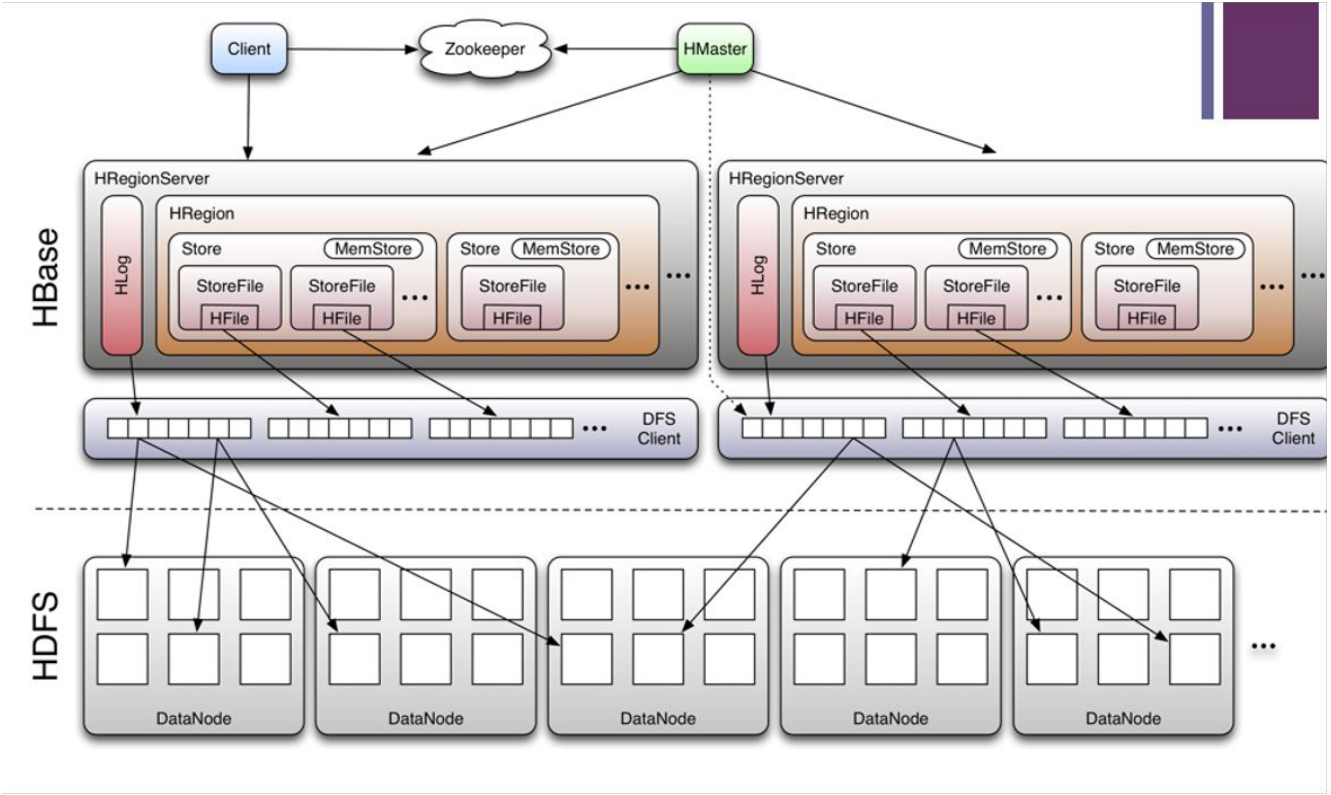
HBase

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 Architecture



HBase Architecture

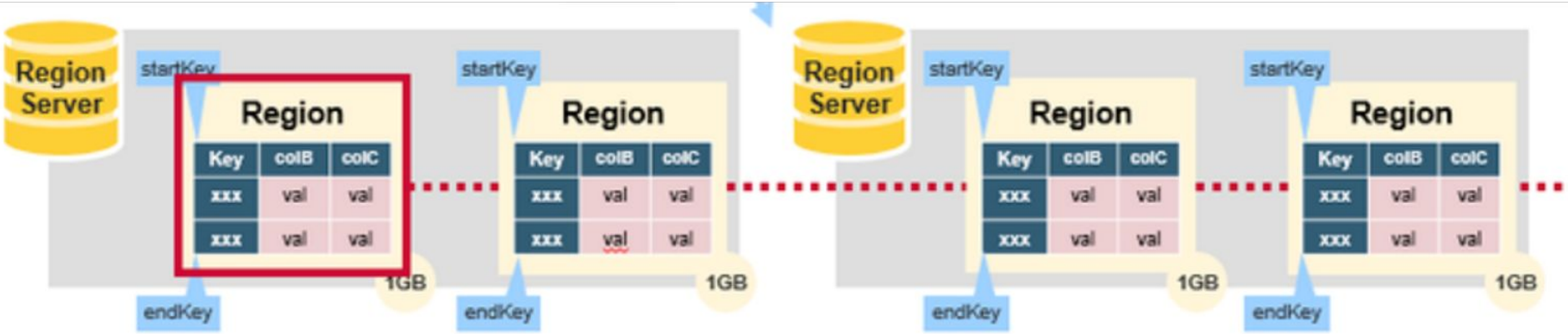
The three main components of Hbase architecture are:

- 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)

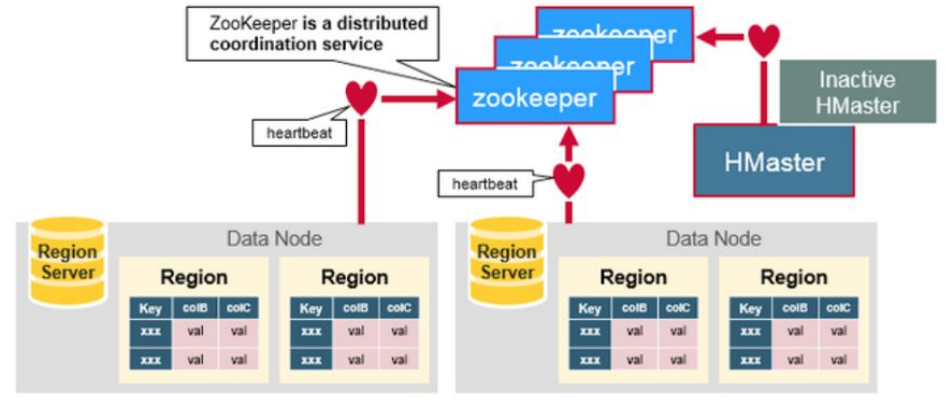
HMaster is responsible for region assignment, coordinating the region servers, and re-assigning regions for recovery or load balancing

- 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.



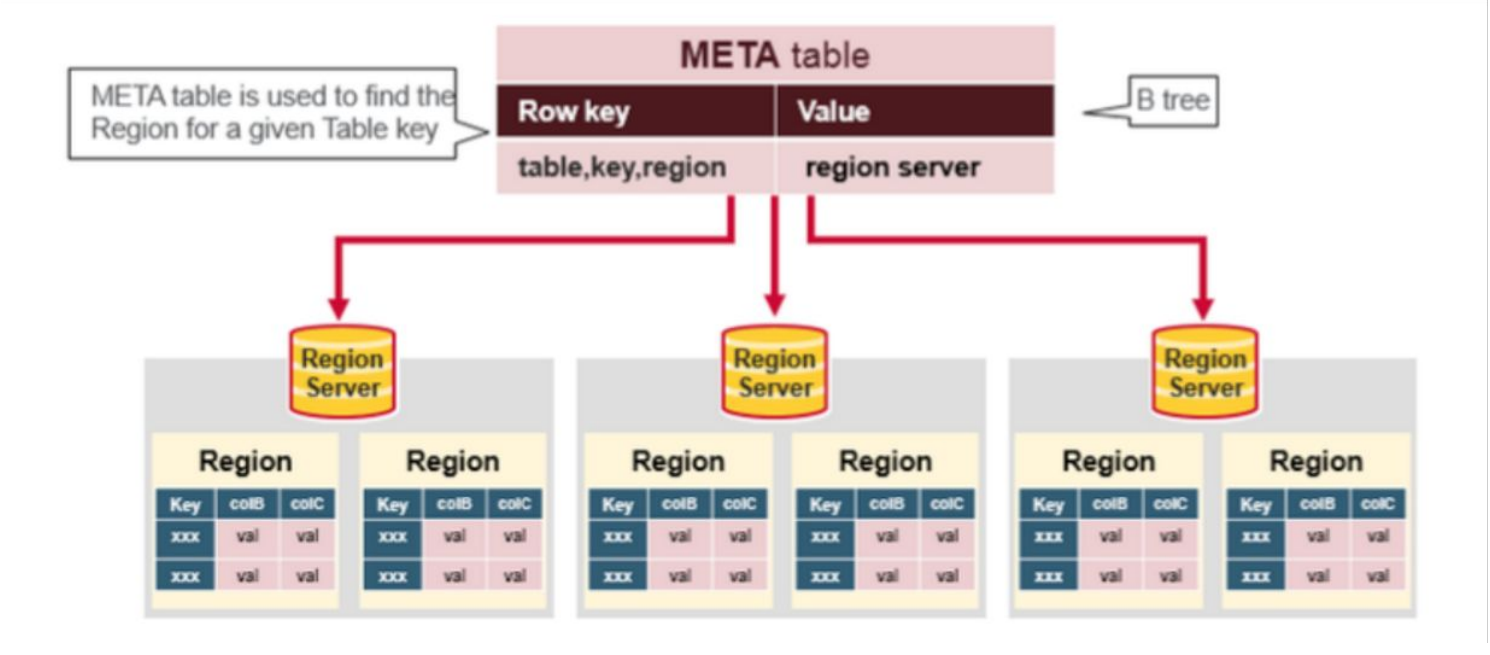
HBase Architecture

There is a special HBase Catalog table called the META table, which holds the location of the regions in the cluster

- ZooKeeper stores the location of the META table.
- First time a client reads or writes to HBase, the client gets the RegionServer that hosts the META table from ZooKeeper
- The client will query the META server to get the RegionServer corresponding to the row key it wants to access
    - The client caches this information along with the META table location
    - For future reads, the client uses the cache to retrieve the META location and previously read row keys
    - It will then get the row from the corresponding RegionServer

HBase Architecture

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



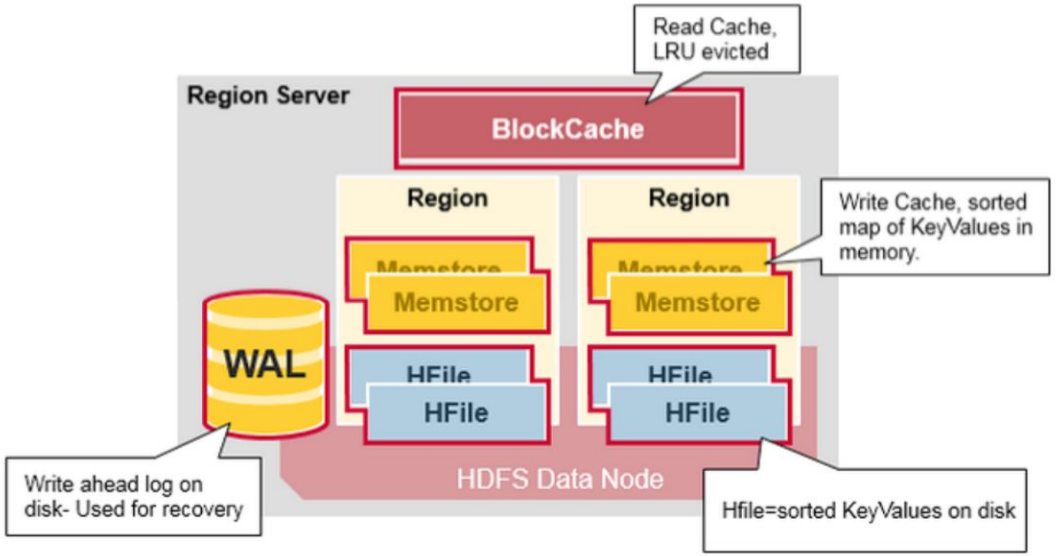
HBase Architecture

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 Architecture

- Initially there is one region per table
- When a region grows too large, it splits into two child regions
- Both child regions, each with ½ of the original region, are opened in parallel on the same RegionServer, and then the split is reported to the HMaster
  - For load balancing reasons, the HMaster may schedule for new regions to be moved off to other servers.
- HDFS replicates the WAL and HFile blocks
  - HFile block replication happens automatically
  - HBase relies on HDFS to provide the data safety as it stores its files
  - When data is written in HDFS replicates the blocks as previously discussed

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

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/