Lecture-9
Big Data Systems
(SEZG522/CCZG522)

BITS Pilani
Pilani Campus

Slides: Courtesy:.Prof. Anindya

innovate achieve lead

BITS Pilani

Pilani Campus

Second Semester

2024-25

Lecture-9 Contents

Hadoop ecosystem technologies

- HBase - Key-value store
- Hive - Data warehouse
- Sqoop
- Flume
- Zookeeper
- Oozie

Why HBase ?

HDFS provides sequential access to data

HBASE provides random access capability of large files in HDFS

- Hash tables used for indexing of HDFS files

Key-value store with no fixed schema as in RDBMS

- so can be used for structured and semi-structured data
- type of NoSQL database

Built for wide tables with many attributes

- de-normalized data

Column oriented storage

- Tuned for analytical queries that access specific columns

Strongly consistent because read is on latest write of a data item

Origins of idea from Google BigTable (on GFS) leading to a Hadoop project (on HDFS)

When to use HBase

- Large data volume where many distributed nodes are needed, e.g. 100s of millions of rows
- Need to lookup data in a large data store
  - that's the key aspect of HBase on HDFS
- Need linear scalability with data volume
- None of these are required : transactional guarantees, secondary indices, DB triggers, complex queries
- Can add enough commodity hardware to scale with favourable cost-performance ratio

Columnar storage

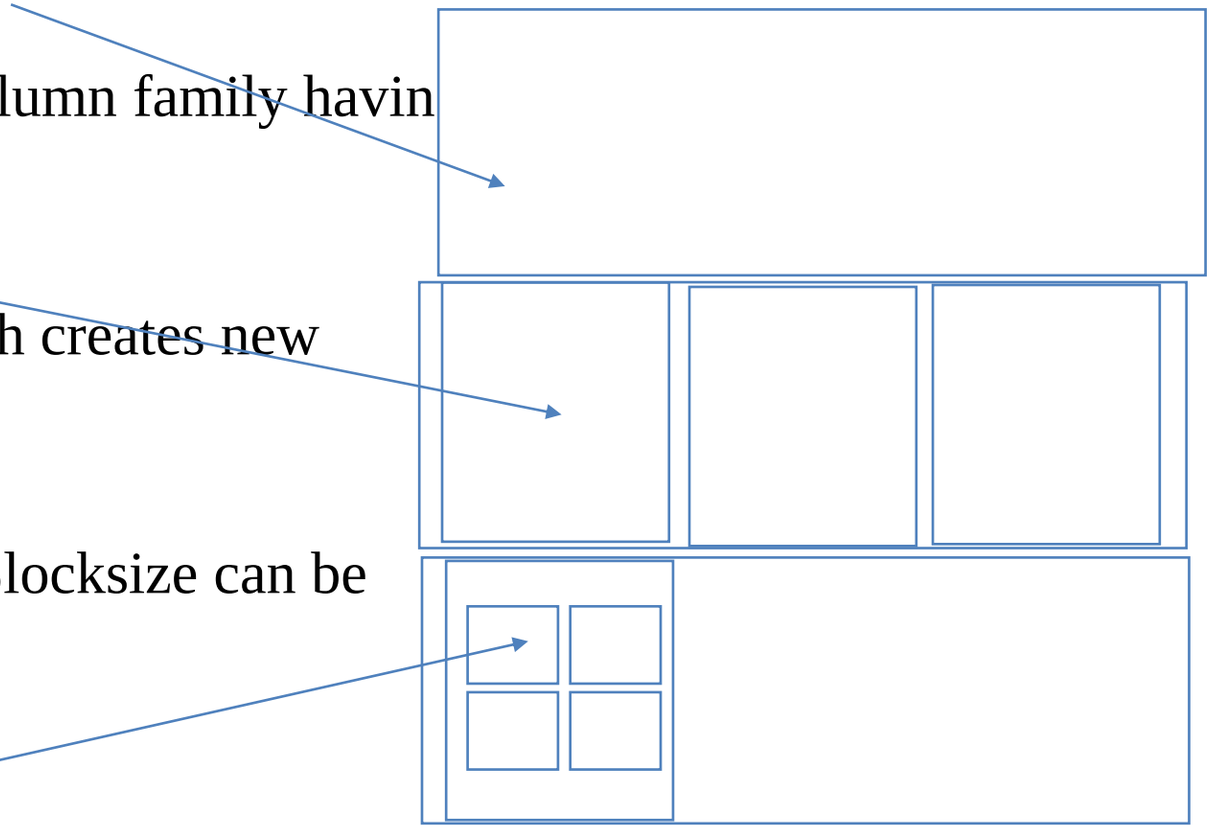
- Data in a row is a collection of column families with each column being a key-value pair
- A column values are stored together on disk
- Each cell value also has a timestamp

COLUMN FAMILIES

Row key	personal data		professional data	
empid	name	city	designation	salary
1	raju	hyderabad	manager	50,000
2	ravi	chennai	sr.engineer	30,000
3	rajesh	delhi	jr.engineer	25,000

HBase storage structure

- Region - Continuous sorted set of rows stored together (using row-key to sort).
  - ✓ HBase tables are split into regions.*
- A region has multiple Column Families with each column family havin Store
- Each Store has
  - ✓ MemStore**: Write buffer for clients. Each flush creates new StoreFile/HFile on disk.
  - ✓ StoreFiles can be compacted
- Each StoreFile has Blocks that can be compressed. Blocksize can be configured per ColumnFamily/Store level.



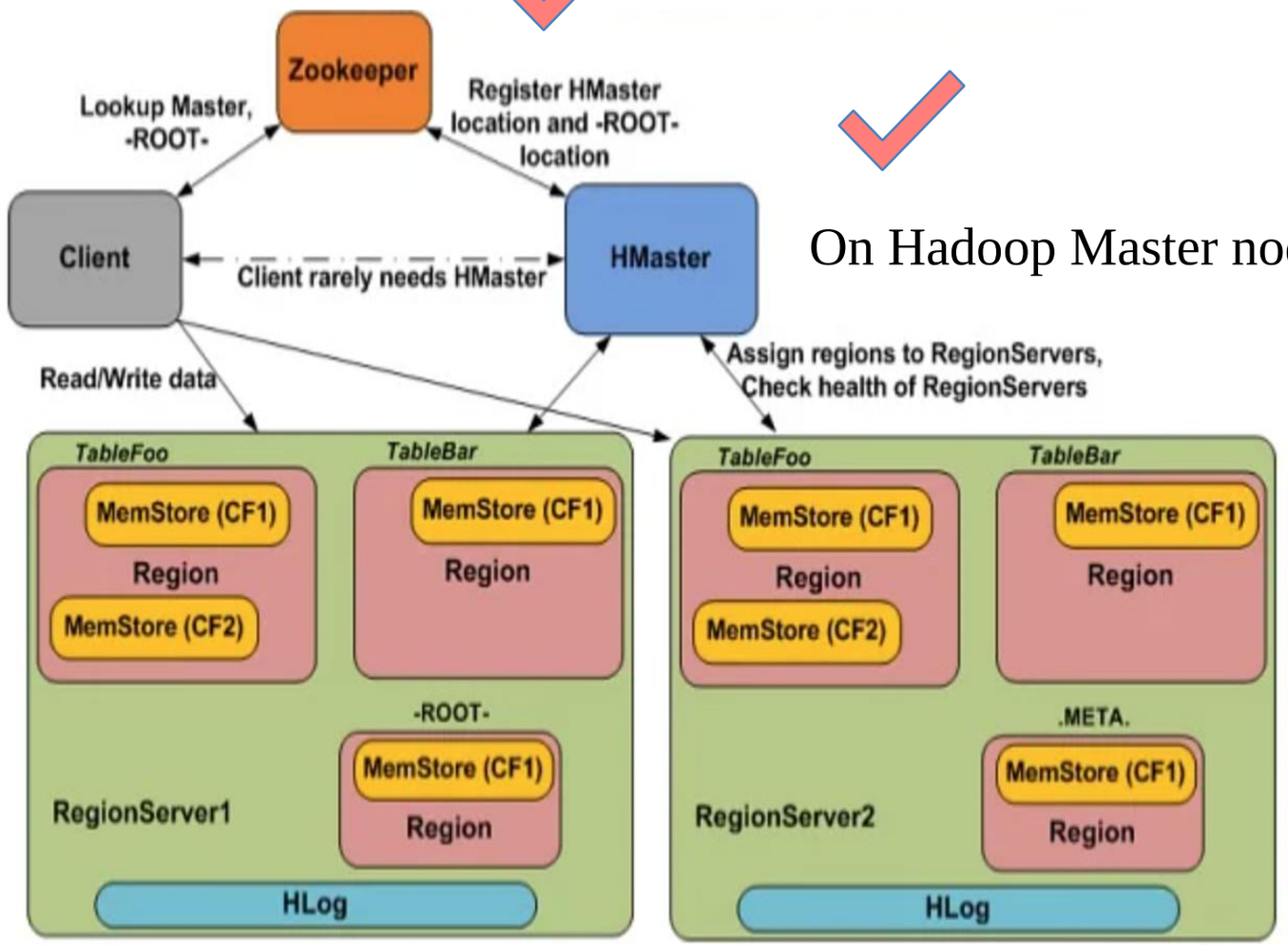
Architectural components - Master

HMaster: Single controller on master node.

- Assigns regions to RegionServers and checks health.
- Failover control
- DDL / meta-data operations

Zookeeper cluster (separate nodes):

- Client communication
- Track failures with cluster with heartbeat
- Maintain cluster config data



On Hadoop Master node

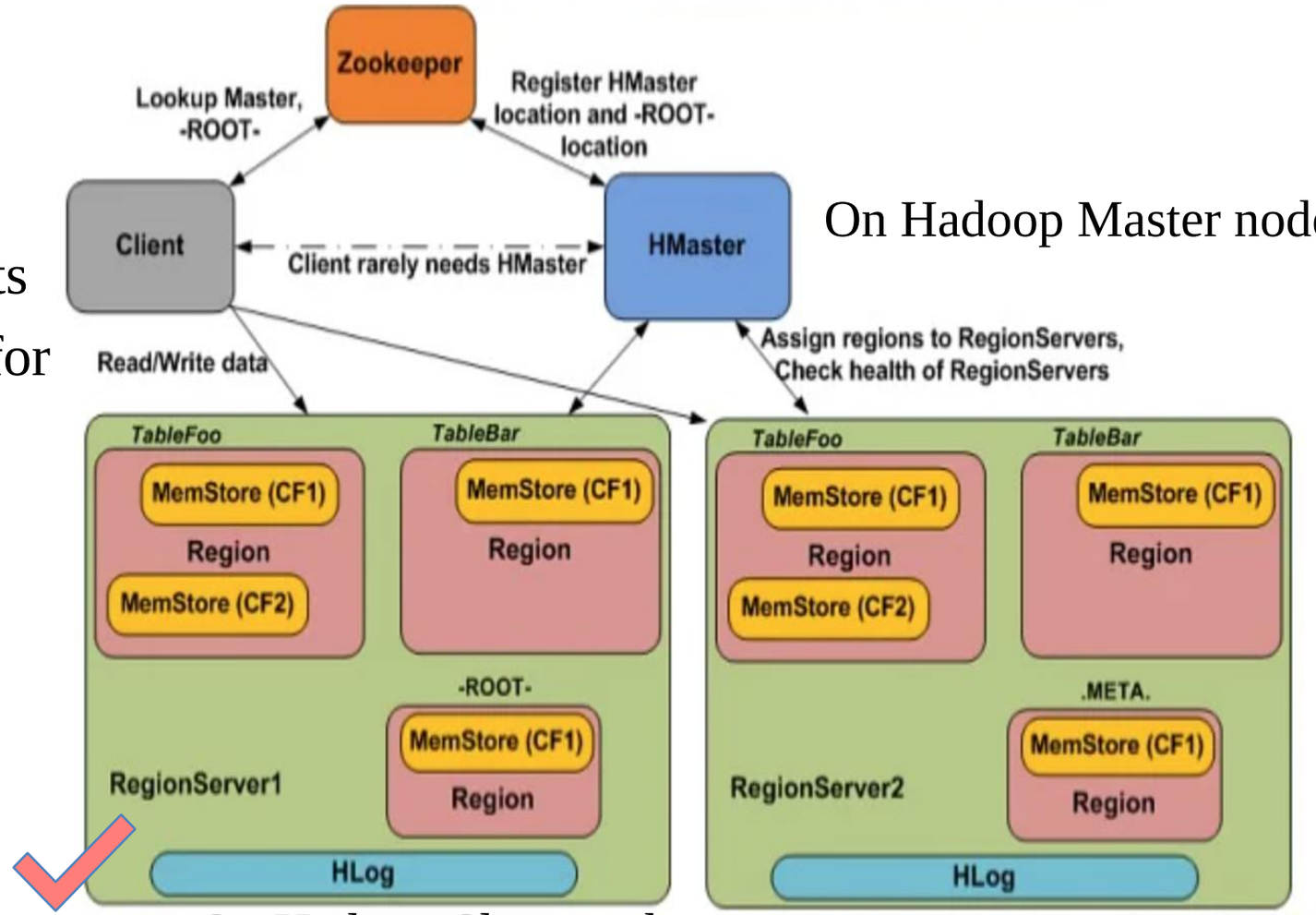
On Hadoop Slave node

CF : Column Family

Architectural components - Slaves

RegionServer: In-charge of set of regions on a slave node / HDFS DataNode

- Worker nodes handling read/write/delete requests from clients
- HLog or Write Ahead Logs (WAL) for MemStore (look in hbase/WALs/ on HDFS)



On Hadoop Master node

On Hadoop Slave node

CF : Column Family

Find HBase objects on HDFS

For data:

/hbase

/<Table> (Tables in the cluster)

/<Region> (Regions for the table)

/<ColumnFamily> (ColumnFamilies for the Region)

/<StoreFile> (StoreFiles for the ColumnFamily)

For WAL logs:

/hbase

/.logs

/<RegionServer> (RegionServers)

/<HLog> (WAL HLog files for the RegionServer)

To see list of regions and utilisation per region of a table:

hadoop fs -du /hbase/<table name>

Find HBase objects on HDFS - Example

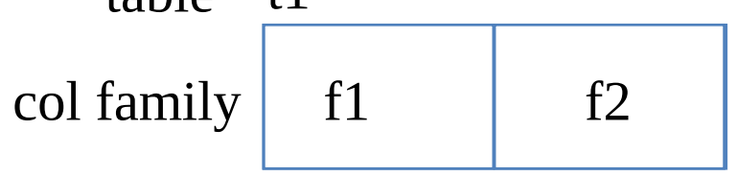
> ls -l $HOME/hbase/hbase-2.4.4/bin/tmp/hbase

```
drwxr-xr-x   6 anindya  staff   192 Jun 27 21:44 MasterData
drwxr-xr-x   3 anindya  staff    96 Jun 28 10:30 WALs
drwxr-xr-x   2 anindya  staff    64 Jun 27 21:44 archive
drwxr-xr-x   2 anindya  staff    64 Jun 27 21:44 corrupt
drwxr-xr-x   4 anindya  staff   128 Jun 27 21:44 data
-rw-r--r--   1 anindya  staff    42 Jun 27 21:44 hbase.id
-rw-r--r--   1 anindya  staff     7 Jun 27 21:44 hbase.version
drwxr-xr-x   2 anindya  staff    64 Jun 27 21:44 mobdir
drwxr-xr-x  62 anindya  staff  1984 Jun 29 01:14 oldWALs
drwx--x--x   2 anindya  staff    64 Jun 27 21:44 staging
```

Configured path for HBase files

Write Ahead Logs

Data files



> ls -l $HOME/hbase/hbase-2.4.4/bin/tmp/hbase/data/default

```
drwxr-xr-x  5 anindya  staff  160 Jun 27 21:44 t1
```

> ls -l /Users/anindya/hbase/hbase-2.4.4/bin/tmp/hbase/data/default/t1/e35e1cb65f5fd84bb4201353d1764365

```
drwxr-xr-x  3 anindya  staff  96 Jun 29 08:10 f1
drwxr-xr-x  3 anindya  staff  96 Jun 27 23:05 f2
drwxr-xr-x  3 anindya  staff  96 Jun 28 10:30 recovered.edits
```

Table

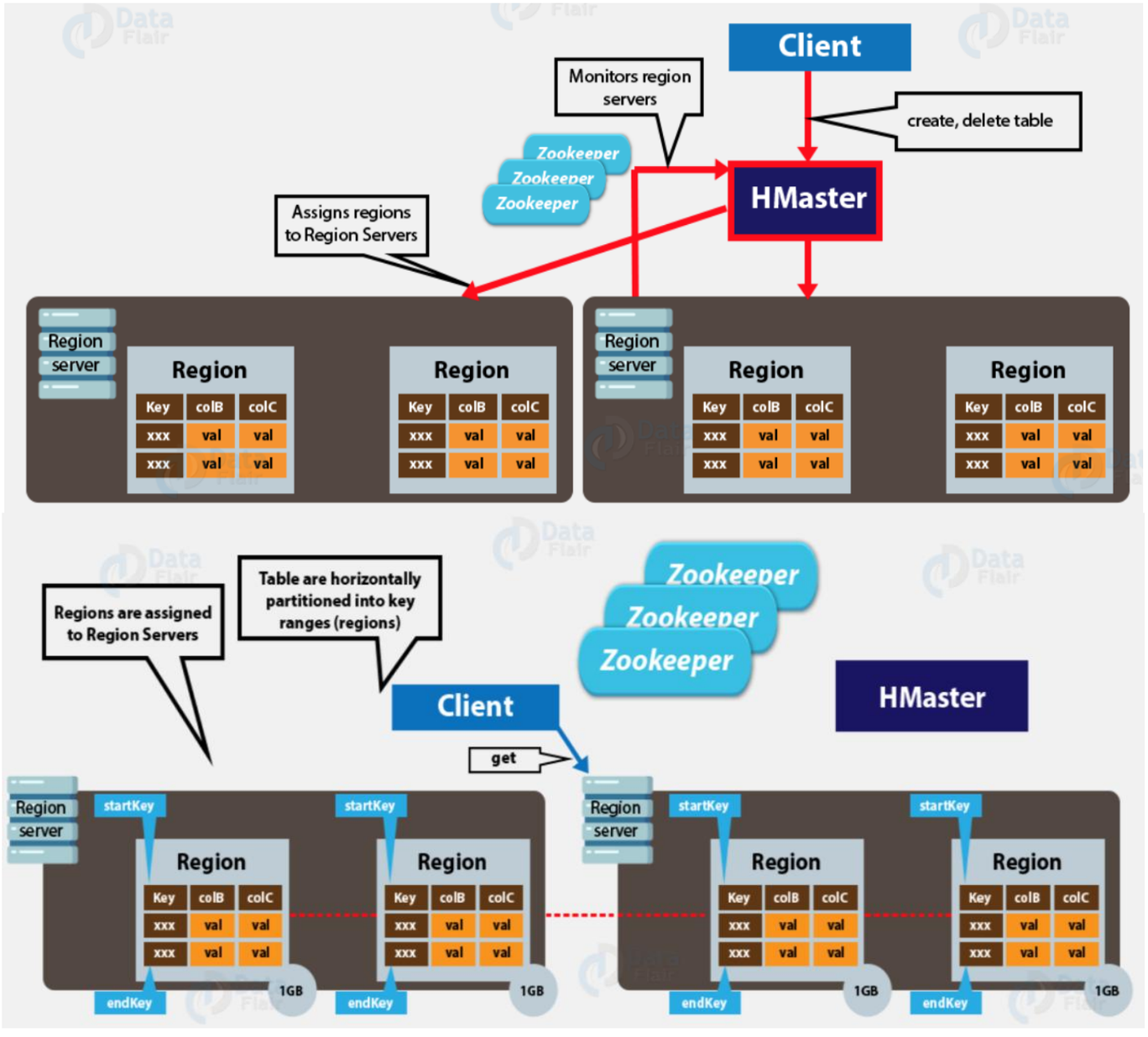
> ls -l /Users/anindya/hbase/hbase-2.4.4/bin/tmp/hbase/data/default/t1/e35e1cb6 Column families /f1

```
-rw-rw-rw-  1 anindya  staff  4891 Jun 27 22:53 01d6868c5ed4426393ab08815971b021
```

HDFS file storing a block

Client operations

- Create / Delete via HMaster
- Meta-data from HMaster
- Actual Read / Write via RegionServer



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

File compaction

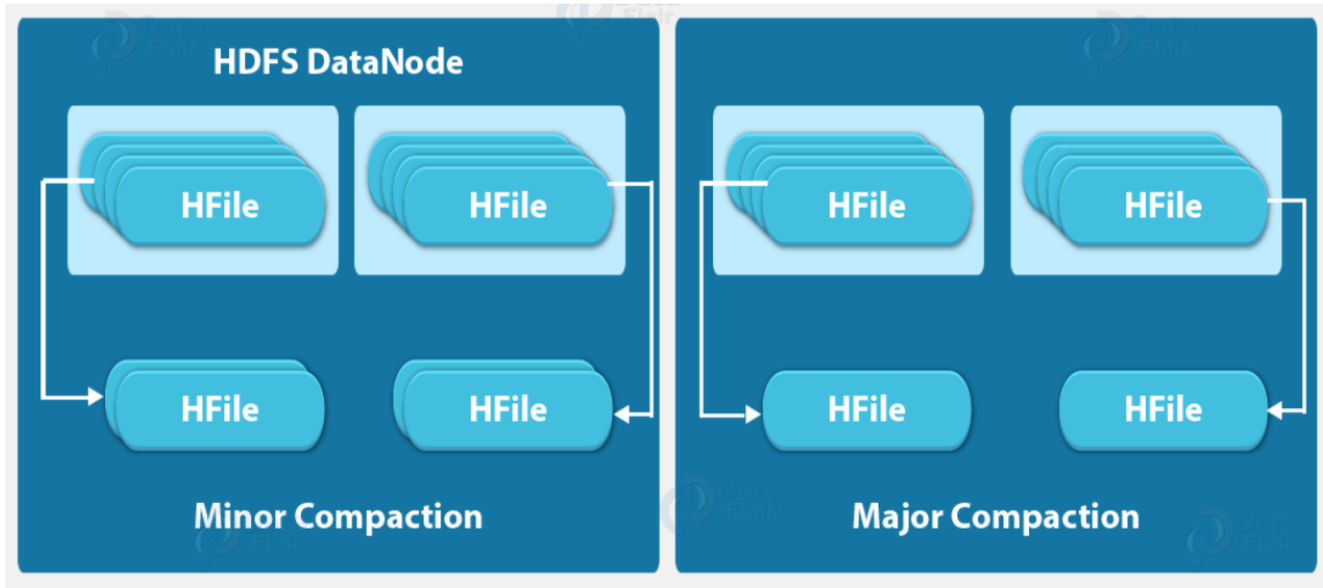
Regions get split on the same node when they grow large.

Typically RegionServers manage all regions on same node.

Load balancing, HDFS replication, or failure may have RegionServers managing regions with non-local StoreFiles.

Compaction is about

- merging StoreFiles into larger files
- merging Regions
- making sure RegionServers and corresponding region files are on same node.



Merge smaller files into larger files on same ColumnFamily on same node

Merge all files into one for sam Column Family possibly across nodes

Drops deleted cells also

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

Splitting tables into Regions

Automatic splitting happens based on size

However in some cases, user may want to control splitting

- Hot spots for data access
- For real-time / time-series data, last region is always active
- Load balancing

hbase> create 'test_table', 'f1', SPLITS=> ['a', 'e', 'i', 'o', ‘u']

- Splitting by sorted row keys starting with letters region1:a-d, region 2:e-h, …

Can also merge regions:

- hbase> merge_region region1 region2

Why Hive

Provides a way to process large structured data in Hadoop

Data Warehouse on Hadoop / HDFS

SQL like query interface

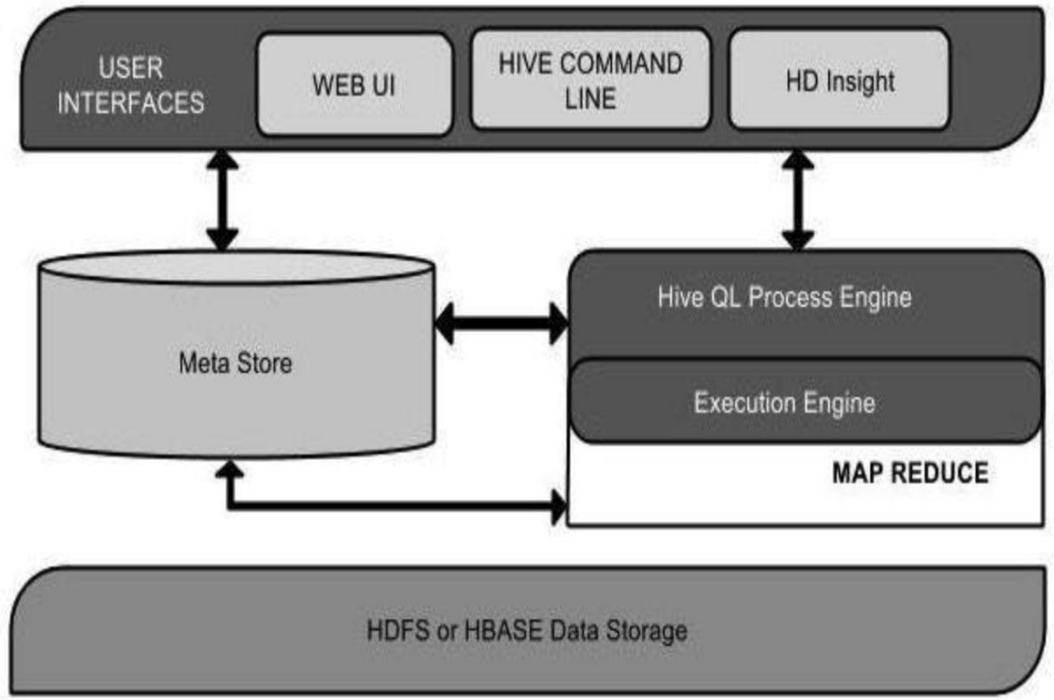
- But it is not an RDBMS with transactions
- Not for real-time queries with row level updates

Meant for OLAP type queries

Initially in Facebook and now an Apache project

Hive Architecture

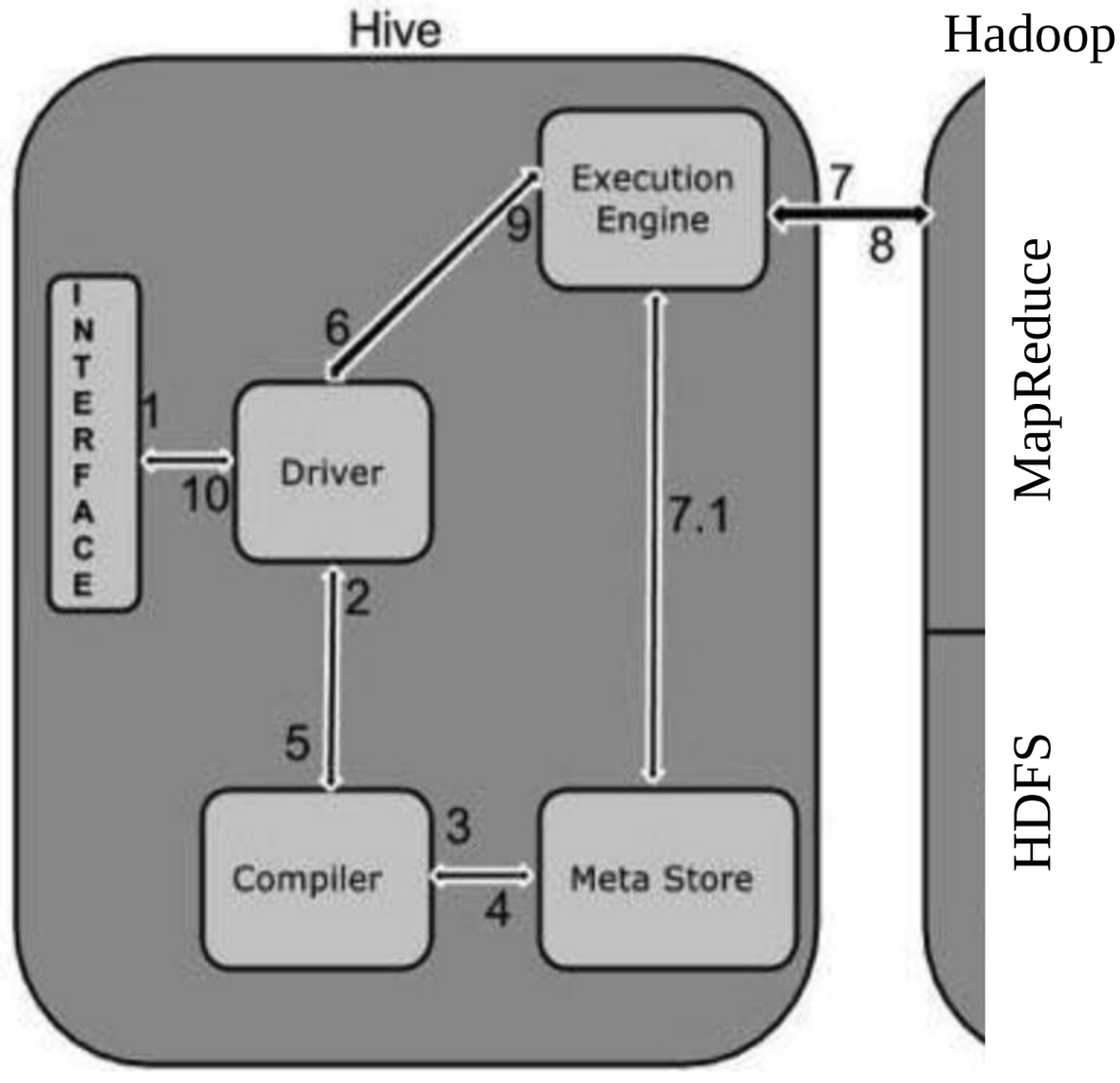
- Meta Store: Meta-data or schema is stored in a relational DB (default Derby, MySQL, ..) containing structure of tables, partitions, columns, data types, [de]-serializers to read/write data, HDFS files or HBase meta-data where data is stored.
- Provides SQL-like interface called HiveQL
- Execution engine with a Compiler (in HiveQL process engine) that consults meta-data to translate a query into MapReduce jobs
- Data in HDFS or in HBase



https://www.tutorialspoint.com/hive/hive_introduction.htm

Hive Query execution flow

1. HiveQL query sent to driver from Interface.
2. Driver sends to Compiler to create a query plan
3. Compiler builds an abstract syntax tree (AST) and then semantic analysis of the query to create a query plan graph (DAG). For this it consults the Meta Store.
4. Meta Store responds to Compiler on requests.
5. Compiler (post optimization of the DAG) responds to Driver with the final plan.
   - Possible optimizations are combining consecutive joins, splitting tasks - e.g. combiners before reducer etc.
6. Driver sends to Execution Engine that essentially executes the query plan via Hadoop MapReduce + data in HDFS or HBase and finally result is sent to Hive interface. (steps 7-10)

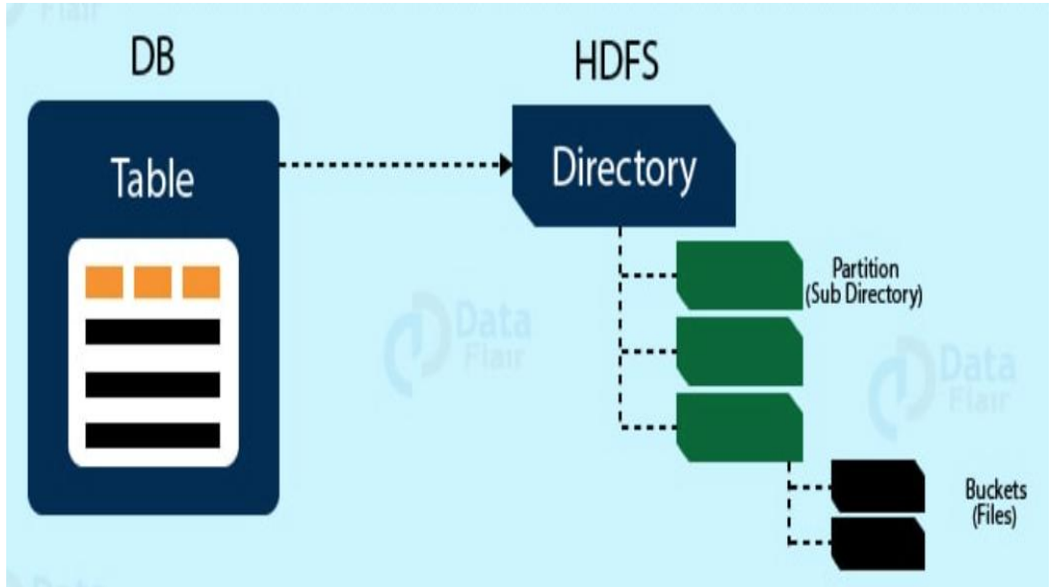


https://www.tutorialspoint.com/hive/hive_introduction.htm

Data model

Table

- Same as RDBMS tables
- Data is stored in a HDFS directory
- Managed tables: Hive stores and manages in warehouse dir
- External tables: Hive doesn't manage but records the path. So actual data can be brought in after create external table in Hive.
- Tables are split into Partitions and then into Buckets



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

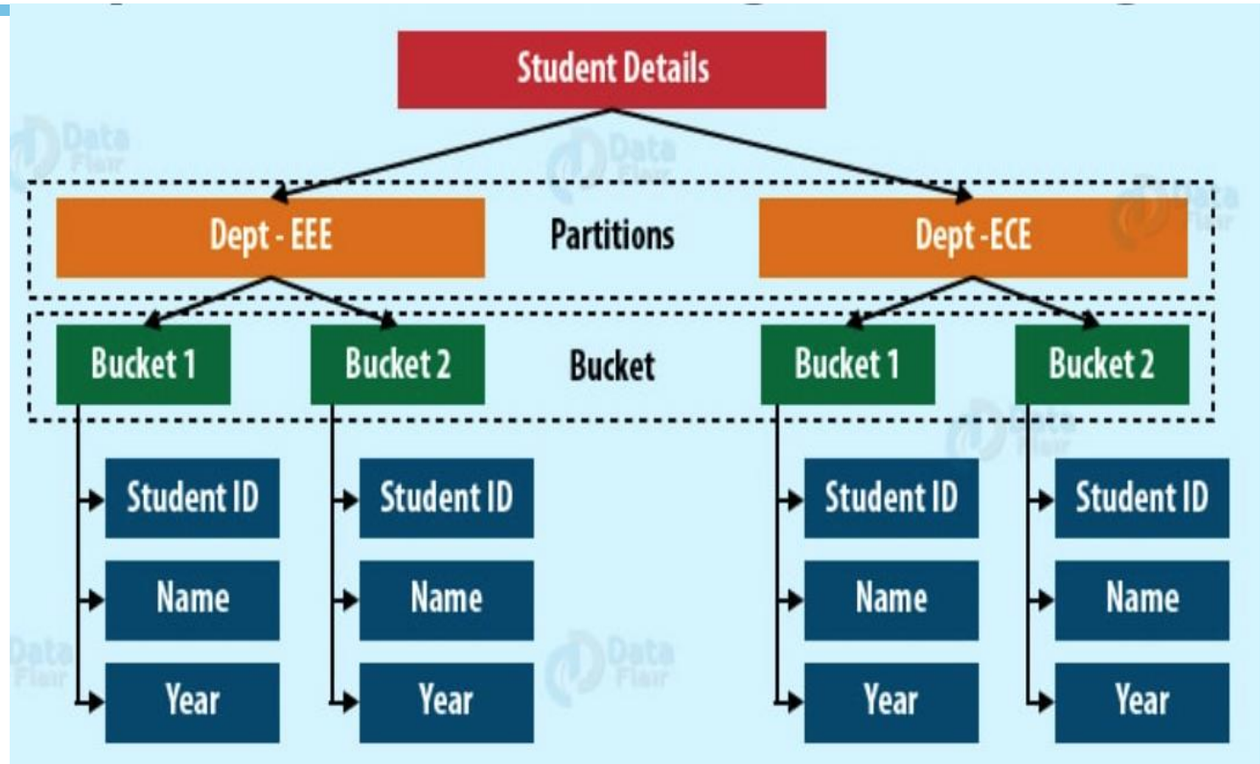
Data model

Partition

- When creating a table a key can be used to split data into partitions - implemented as separate sub-dirs with table dir on HDFS
- e.g. student data partitioned by dept id

Bucket

- Additional level of sub-division within a partition based on hash of some column to make some queries efficient
- These map to actual files on HDFS / HBase



CREATE TABLE student_details (id int, name varchar(50), year int) PARTITIONED BY (dept varchar(3)) CLUSTERED BY (year) SORTED BY (id ASC) INTO N BUCKETS

put year in same bucket but control total bucket count to keep their sizes balanced

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

Data partitioning vs bucketing

Partitioning

Distributes execution load horizontally.

- Map tasks can be given partitions

Faster execution of queries with the low volume of data takes place. E.g. search population from Vatican City returns faster than China

However,

May have too many small partitions with too many directories.

Effective for low volume data. But there some queries like group by on high volume of data take a long time to execute. For e.g., grouping population of China will take longer than Vatican City.

Bucketing

- Done when partition sizes vary a lot or there are too many partitions
- Provides faster query response within smaller segments of data. Like further indexing beyond partition keys.
- Almost equal volumes of data in each bucket — so joins at Map side will be quicker.
- Enables pre-sorting on smaller at data sets. Makes Map side merge sorts even more efficient.

However,

- Can define a number of buckets during table creation. But loading of an equal volume of data has to be done manually* by programmers.

* Like ETL - this is a Data Warehouse !

Example of Partitions and Buckets

```
CREATE TABLE bucketed_user (
        firstname VARCHAR(64),
         lastname  VARCHAR(64),
         address   STRING,
         city  VARCHAR(64),
         state  VARCHAR(64),
         post      STRING,
         phone1    VARCHAR(64),
         phone2    STRING,
         email     STRING,
         web       STRING
)
COMMENT ‘A bucketed sorted user table’
PARTITIONED BY (country VARCHAR(64))
CLUSTERED BY (state) SORTED BY (city) INTO 32
BUCKETS
STORED AS SEQUENCEFILE;
```

Create partitions based on country
Group state records into same bucket
Control total number of buckets
Store as binary file to save space

A point about Map-side joins and buckets

Employee

Andy,	700000,	5
Bob,	800000,	2
Clara,	140000,	5
David,	230000,	5

Department

2,	Sales
3,	Engineering
5,	Finance

Map

Table: Employee
Key=2, Value=(Bob, 800000,2)
…

Table: Department
Key=2, Value(2, Sales)
…

dept key used as join key for data from 2 tables

Bucketed tables helps to make these joins faster in memory with small sized files sorted, say in this case by DeptID. So MergeSort works faster.

Shuffle - Sort

Key=2, {Value=(Bob, 800000,2), Table:Employee}
{Value(2, Sales), Tab: Department}
…

Reduce

Key=2, Value=(Bob, 800000,Sales)
…

Sqoop Introduction

"SQL to Hadoop & Hadoop to SQL"

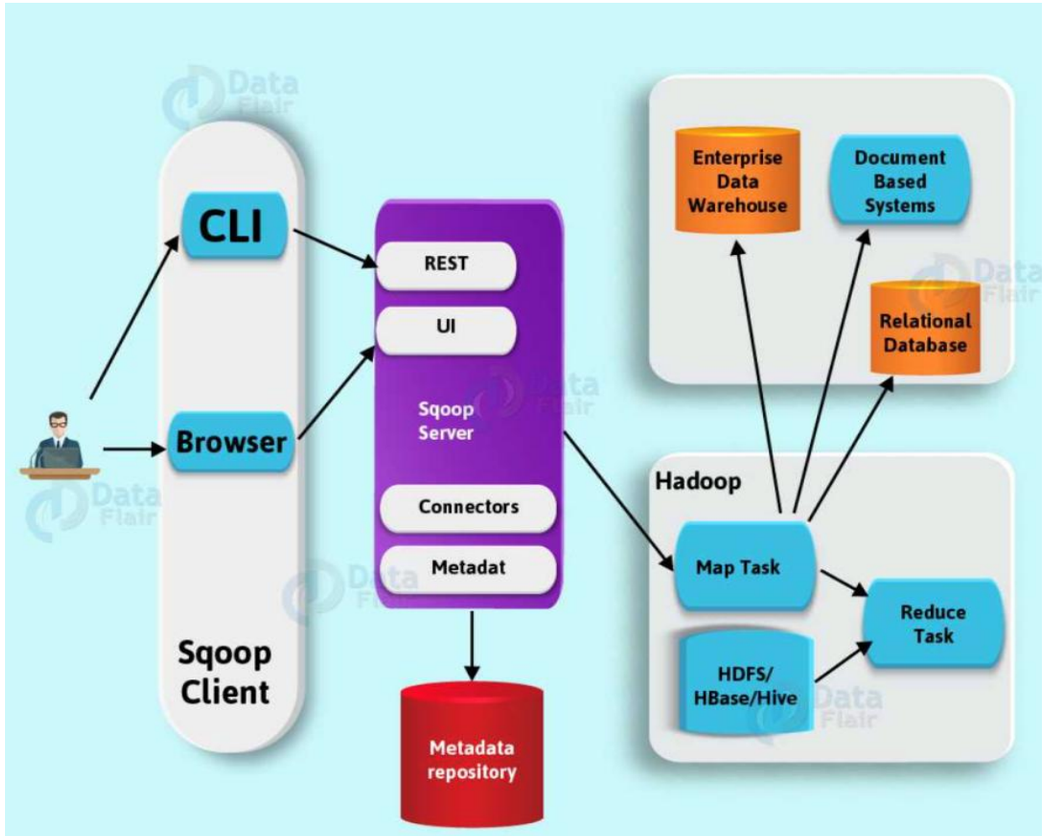
A tool in Hadoop ecosystem which is designed to transfer data between HDFS (Hadoop storage) and RDBMS

- ✔ like MySQL, Oracle RDB, SQLite, Teradata, Netezza, Postgres etc.

Efficiently transfers bulk data between Hadoop and external data stores such as enterprise data warehouses, relational databases, etc.

Supports import and export operations

- ✔ import data from relational DB such as MySQL, Oracle.
- ✔ export data from HDFS to relational DB.



Why Sqoop?

Big data developer's

- ✔ works start once the data are in Hadoop system like in HDFS, Hive or Hbase
- ✔ Performs magical stuff to find all the golden information hidden on such a huge amount of data
- ✔ Used to write to import and export data between Hadoop and RDBMS

Hence a tool was needed !

Solution came in form of Sqoop

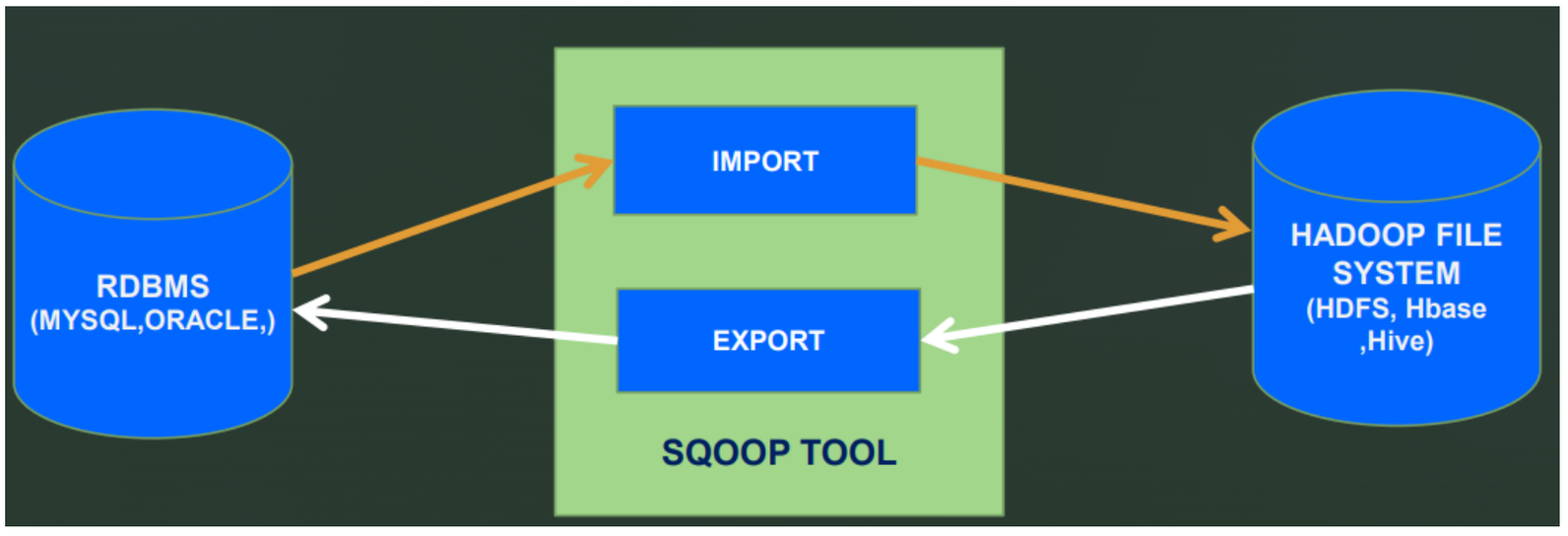
- ✔ Sqoop uses the MapReduce mechanism for its operations like import and export work and work on a parallel mechanism as well as fault tolerance.
- ✔ Developers just need to mention the source, destination and the rest of the work will be done by Sqoop
- ✔ Filled the data transfer gap between relational databases and Hadoop system

Sqoop Architecture

Source which is RDBMS like MySQL

Destination like Hbase or HDFS or Hive

Sqoop performs the operation for import and export



Features of Sqoop

Full Load
- ✔Can load the whole table by a single command
- ✔Can also load all the tables from a database using a single command

Incremental Load
- ✔Also provides the facility of incremental load where you can load parts of table whenever it is updated
- ✔Supports two types of incremental imports: 1. Append 2. Last modified

Parallel import/export
- ✔Uses YARN framework to import and export the data, which provides fault tolerance on top of parallelism

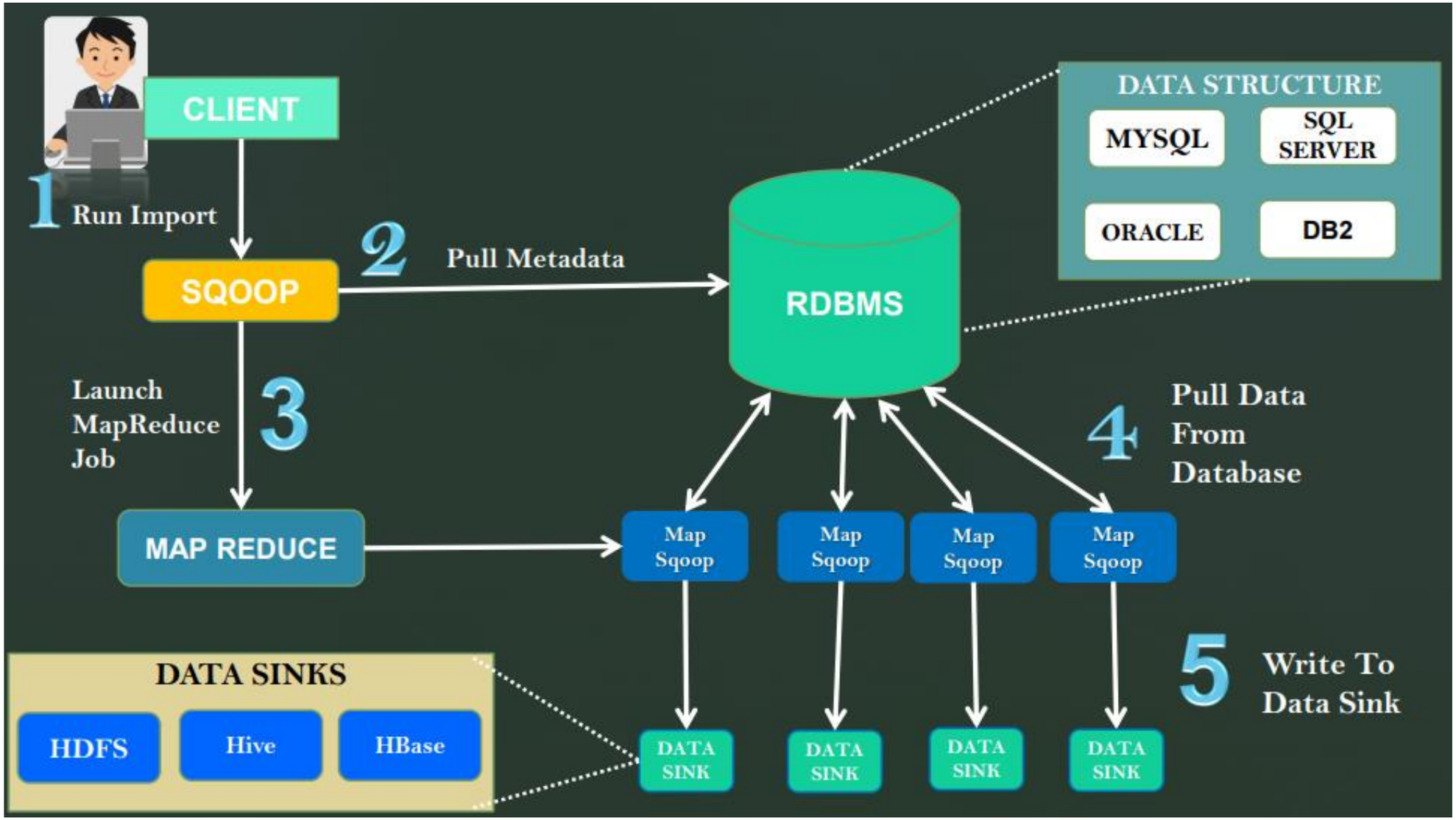
Import results of SQL query
- ✔Can also import the result returned from an SQL query in HDFS

Compression
- ✔Can compress data by using deflate(gzip) algorithm with –compress argument, or by specifying –compression-codec argument
- ✔Can also load compressed table in Apache Hive

Connectors for all major RDBMS Databases
- ✔Provides connectors for multiple RDBMS databases, covering almost the entire circumference

Load data directly into HIVE/HBase
- ✔Can load data directly into Apache Hive for analysis and also dump data in HBase, which is a NoSQL database

Five stage Sqoop Import Overview

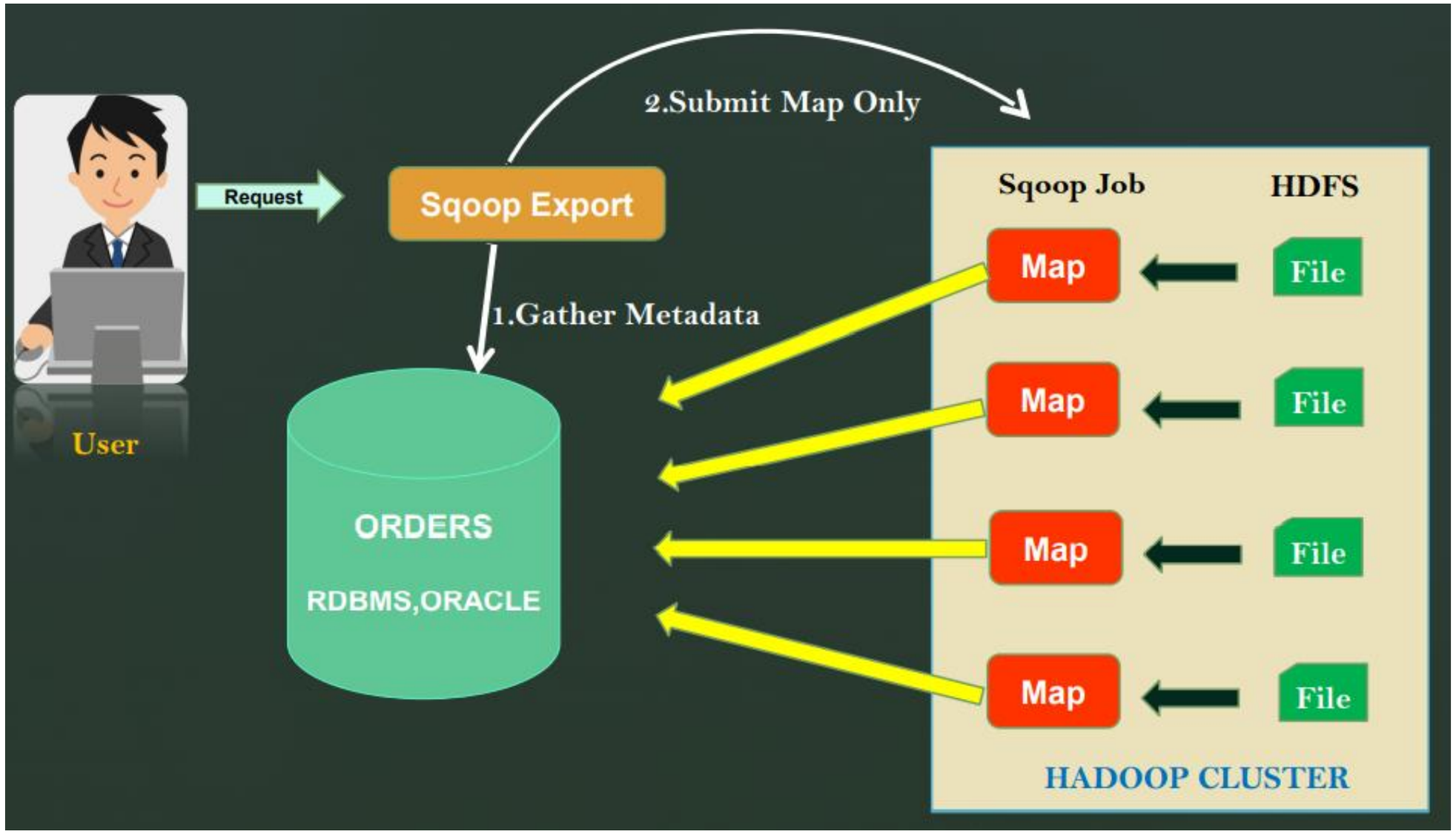


Importing Data using Sqoop

SQOOP Import Command

- ✔Import individual tables from RDBMS to HDFS
- ✔Each row in a table is treated as records in HDFS
- ✔All record are stored as text data in text files or binary files

Exporting Data using Sqoop



Sqoop Export Data

SQOOP Export Command

- ✔Export a set of files from HDFS back to RDBMS
- ✔Files given an input to SQOOP contains records called as rows in table

Limitations of Sqoop

Sqoop cannot be paused and resumed

- ✔an atomic step
- ✔If failed, need to clear things up and start again

Failures need special handling in case of partial import or export

Sqoop Export performance also depends upon the hardware configuration (Memory, Hard disk) of RDBMS server

For few databases Sqoop provides bulk connector which has faster performance

- ✔Uses a JDBC connection to connect with RDBMS based on data stores, and this can be inefficient and less performance

Flume Introduction (1)

An e-retailer generates customer browse, buy logs on its website

These logs need to be streamed into a Hadoop environment for analysis of customer behaviour by the data analytics team

Flume

- Moves logs from source application servers to HDFS files
- Performs reliable buffering and message delivery for occasional data spikes
- Scales with workload
- Can be easily managed and customised
- Parallel transfer support
- Contextual routing

Flume Introduction (2)

Why can't we simply do an HDFS 'put' ?

- It is a one time command - we need to reliably stream data
- If data is partially copied or if a failure happens in between the HDFS file is not closed and size is 0

Solution: Apache Flume

Architecture

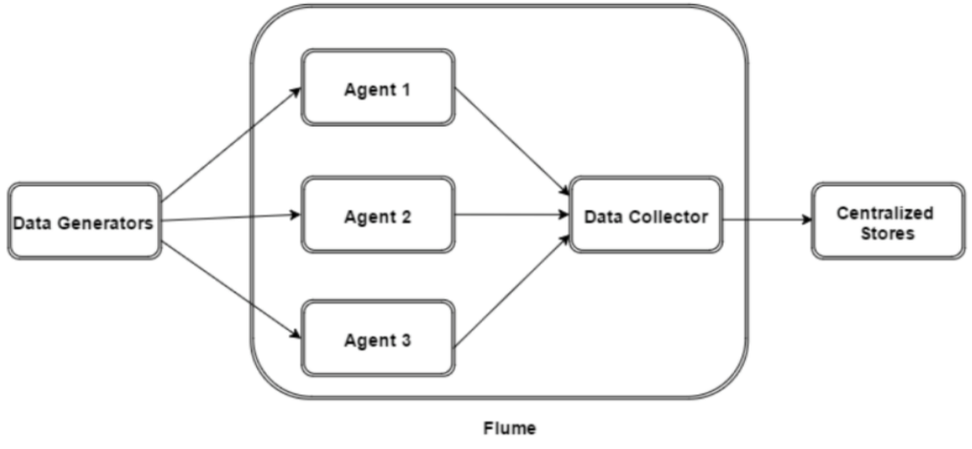
Agents move data from input to output - possible to other agents

- Contain source, channel, sink
- JVM process
- Source: Receives data from generator or another agent, specific types (e.g. Avro source)
- Sink: Consumes from channel and stores in Central store (e.g. HDFS, Hive, HBase sinks) or sends to another agent
- Channel: “Transactional” transient data store to act as a bridge between “1 or more” sources and sinks (e.g. JDBC, memory, filesystem channels)

Flume is event driven

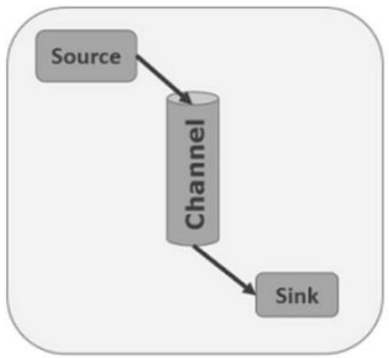
- Event: Basic unit of data transfer

Reliable messaging: For each event, 2 transactions happen. One at sender and second at receiver. Sender commits after receiver commits.



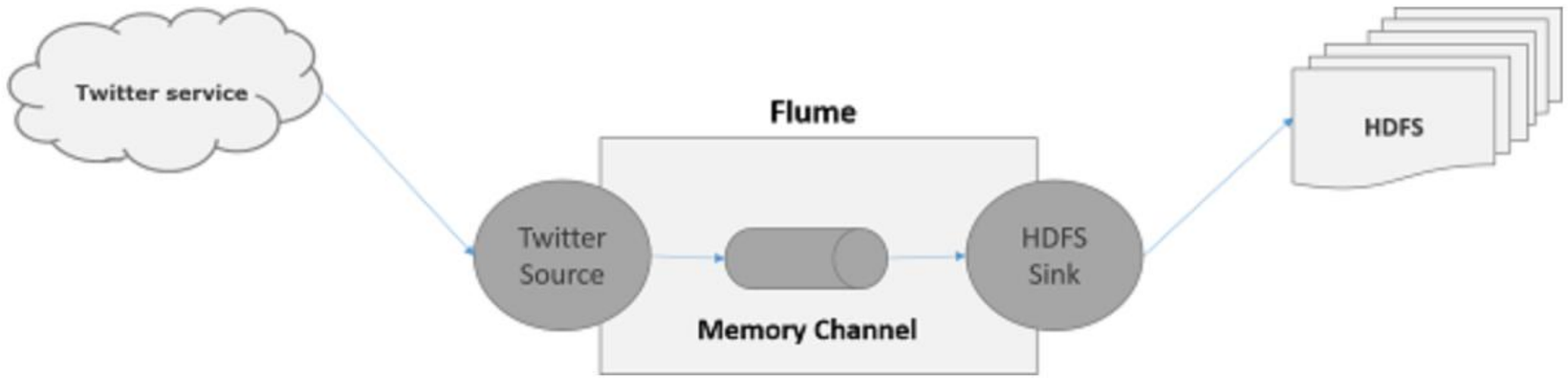
Flume

Flume event



Flume Agent

Example



Create a Twitter API based app for data source

Install a HDFS sink

Setup Flume and configure it with the app as a source and HDFS as the sink

Sqoop vs Flume

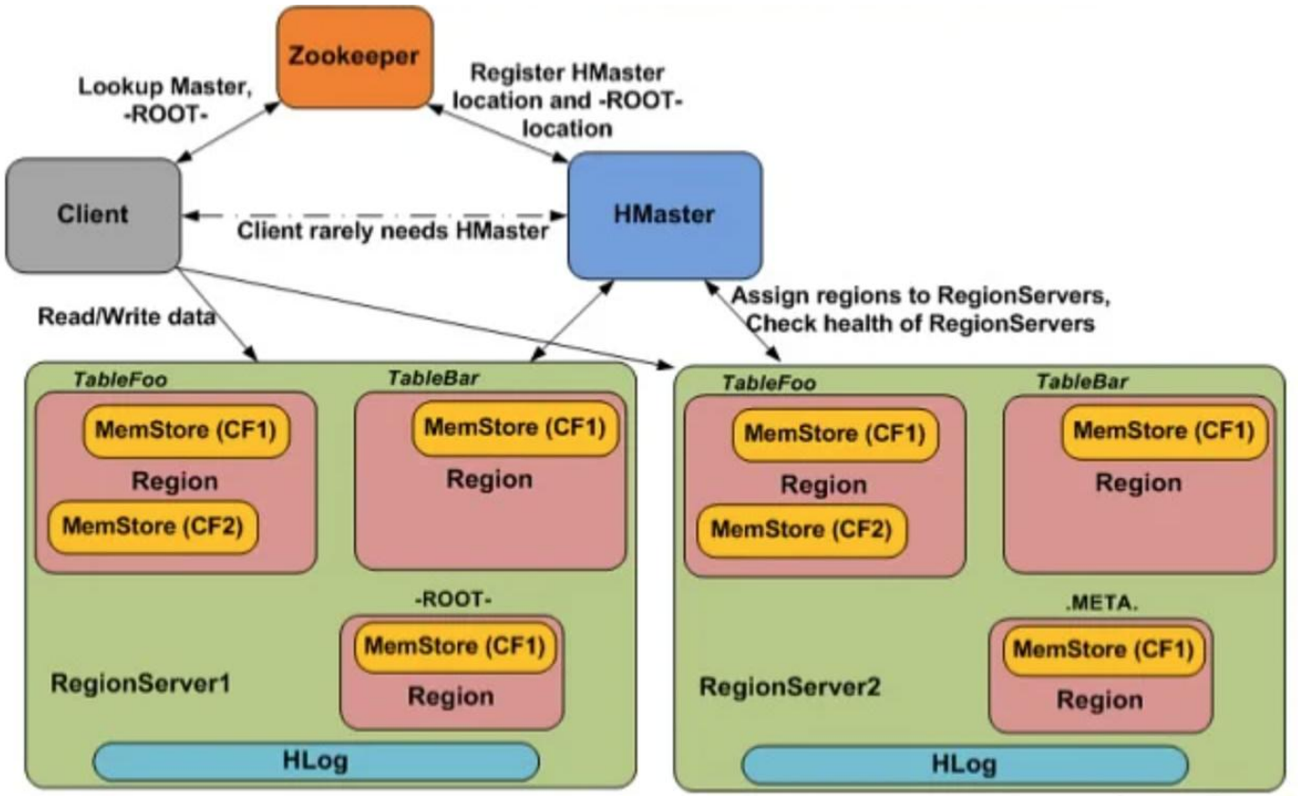
	Sqoop	Flume
Data Flow	Various RDBMS, NoSQL, can map to Hive/HBase from RDBMS	Streaming data, e.g. logs
Loading type	Not event driven	Event driven
Source integration	Connector driven	Agent driven
Usage	Structured sources	Streaming systems with semi-structured data, e.g. twitter feed, web server logs
Performance and reliability	Parallel transfer with high utilisation	Reliable transfer with aggregations at low latency

Zookeeper Introduction

Distributed systems need a coordination system that can be a common service

Example: Zookeeper for HBase cluster

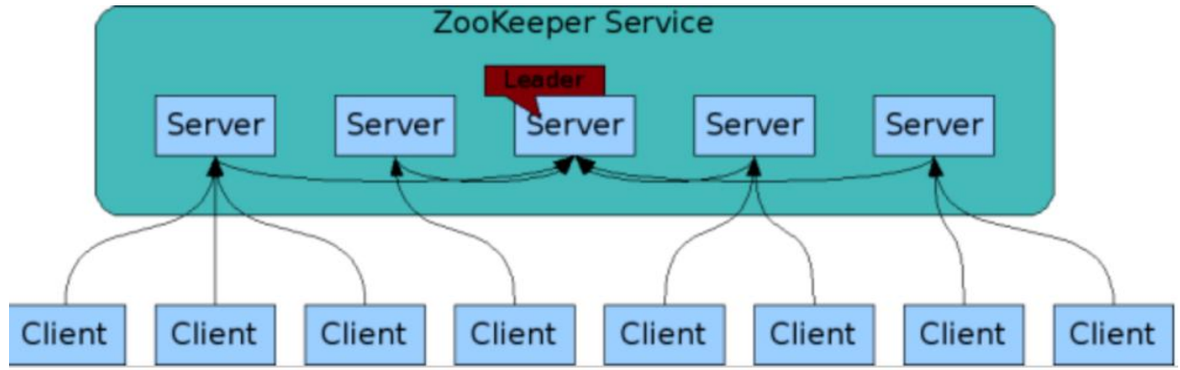
- Client needs to lookup current HMaster node
- Track failures in cluster with heartbeat and pick a new HMaster
- Maintain light weight cluster config data



cloudera

Zookeeper Architecture

- High performance, highly available, strictly ordered access
- Built in-memory and as a Java application
- Organized like an in-memory file system with dirs/files (called znodes)
- Zookeeper itself is clustered - ensemble
- Available as long as majority in ensemble is available
- Clients connect on TCP session for request, response, heartbeats, events etc.
- When a connection dies, client connects to another server or server detects client failure
- All client transactions are time stamped and ordered
- Typically meant for read-heavy workloads



Guarantees

- Sequential Consistency - Updates on a data item from a client will be applied in the order that they were sent.
- Atomicity - Updates on a data item either succeed or fail. No partial results.
- Single System Image - A client will see the same view of the service regardless of the server that it connects to.
- Reliability - Once an update has been applied on a data item, it will persist from that time forward until a client overwrites the update.
- Timeliness - The clients view of the system is guaranteed to be up-to-date within a certain time bound.

Introduction to Oozie

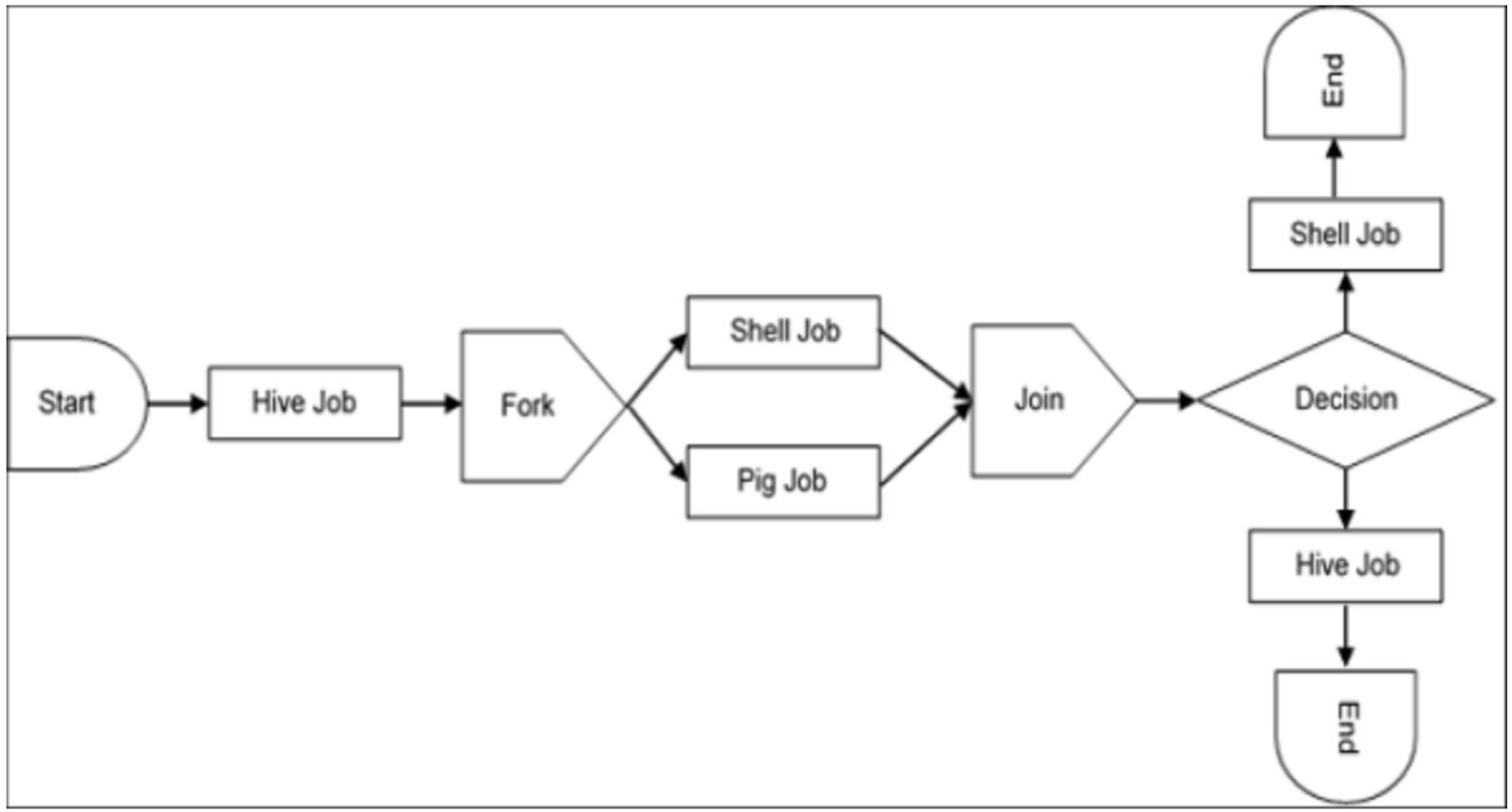
- Scheduler for Hadoop jobs
- Sequence multiple jobs using a specification input
- Support for Hive, PIG, Sqoop, or Java and shell scripts
- Built in Java as a web-application
- Uses Hadoop execution engine to execute tasks
- So load-balancing, failover is done by Hadoop
- Tasks have a callback URL to tell Oozie when done or can poll the task status

Types of jobs

Workflow jobs: In a DAG form to specify sequence of actions

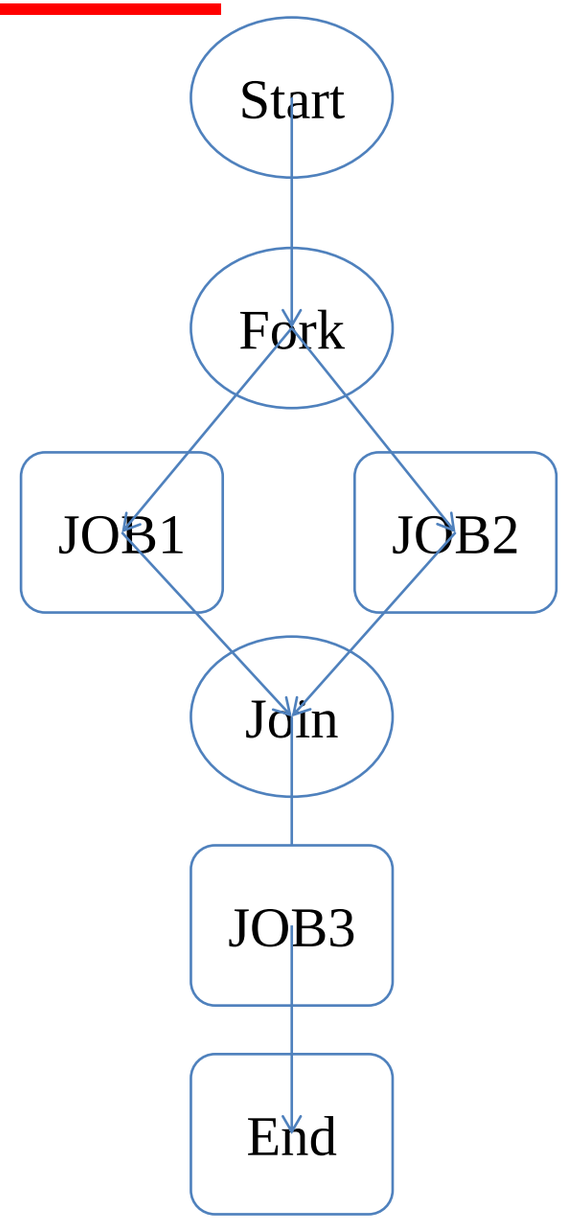
Coordinator jobs: Triggered by time and data availability

Bundle: Package of workflow and coordinator jobs



Workflow (1)

```
<workflow-app name = "simple-Workflow">
  <start to = "fork_node" />

  <fork name = "fork_node">
    <path start = “Job1"/>
    <path start = "Job2"/>
  </fork>

  <action name = "Job1">
    <parameters for running Job1 with arguments, settings, executable path>
    <ok to = "join_node" />
    <error to = "kill_job" />
  </action>

  <action name = "Job2">
    <parameters for running Job1 with arguments, settings, executable path>
    <ok to = "join_node" />
    <error to = "kill_job" />
  </action>

  <join name = "join_node" to = "Job3"/>

  <action name = "Job3">
    <parameters for running Job1 with arguments, settings, executable path>
     <ok to = "end" />
    <error to = "kill_job" />
  </action>

  <kill name = "kill_job">
    <message>Job failed</message>
  </kill>
  <end name = "end" /
</workflow-app>
```



Example job spec :

```
<hive xmlns = "uri:oozie:hive-action:0.4">
 <job-tracker>xyz.com:8088</job-tracker>
 <name-node>hdfs://rootname</name-node>
 <script>hdfs_path_of_script/Copydata.hive</script>
 <param>database_name</param>
</hive>
```

Workflow (2)

```
<workflow-app xmlns = "uri:oozie:workflow:0.4" name = "simple-Workflow">
  <start to = "Decision1" />

  <decision name = "Decision1">
    <switch>
      <case to = "Job1">${fs:exists('/test/abc') eq 'false'}
        </case>
      <default to = "Job2" />
    </switch>
  </decision>

  <action name = "Job1">
    <action details>
    <ok to = "Job3" />
    <error to = "kill_job" />
  </action>

  <action name = "Job2">
    <action details>
    <ok to = "end" />
    <error to = "kill_job" />
  </action>

  <action name = “Job3">
    <action details>
    <ok to = "end" />
    <error to = "kill_job" />
  </action>

  <kill name = "kill_job">
    <message>Job failed</message>
  </kill>
  <end name = "end" />
</workflow-app>
```