Hive

Kaustuv Kunal

https://github.com/kaustuvkunal/Hive-Tutorial

Outline

1. Introduction
2. Versions Evolution
3. Architecture & Components
4. Client-Server Architecture
5. Installation
6. Hive Basics (DDL,DML,QL)
7. Acid Transaction
8. Storage Format
9. Indexing, Logging & UDF
10. Future work & Alternative
11. References

What is Hive

- Data warehouse of Hadoop ecosystem
- Developed by Facebook
- Initially created to build an abstraction over MapReduce for SQL developer
- Used for structured data & require schema
- Access file store in HDFS , HBase and in other data storage systems (e.g. AWS S3)
- Hive is not designed for online transaction processing

Hive Features

- Hive is Schema on Write
- Only open source Hadoop SQL with transactions i.e. Insert/Update/Delete/Merge
- Hive is best used for traditional data warehousing tasks and is not designed for online transaction processing
- Hive is designed to maximize scalability, performance, extensibility, fault-tolerance and loose-coupling with its input formats

OLAP, OLTP & hive

- Online Analytical Process: Supports hundreds to thousand of transaction per-second. More of a querying system. Tables not normalized examples are Hive, Cassandra. Slow changing tables.

- Online Transactional Process: Supports millions & billions of transactions per second. More of database modifying system. Tables normalized, example Oracle MySQL. Requires high integrity. Fast changing tables.

Hive is OLAP

CAP & Hive

CAP theorem says, A distributed database system can only have 2 of the below 3 properties:

1. **Consistency** : All nodes see the same data at the same time. *read* operation will return the value of the most recent *write*. entire transaction gets rolled back if there is an error during any stage in the process.
2. **Availability** : Every request gets a response on success/failure. the system remains operational 100% of the time.
3. **Partition Tolerance** : A system that is partition-tolerant can sustain any amount of network failure that doesn't result in a failure of the entire network. Data records are sufficiently replicated across combinations of nodes and networks to keep the system up through intermittent outages.

A data warehouse can be either CA or CP or AP . OLAP are generally AP and **Hive is also an AP data warehouse.**

Hive Evolution

- 2008 – Facebook open sources hive
- 2010 – Apache Hive initial version released
- 2015 – Hive 1.x.y release
- 2016 – Hive 2.x.y release
- 2018 - Hive 3.x.y release

Hive Versions

Version 0.X:

- Provided view instead of file view (In HDFS)
- Translated SQL to MapReduce
- Mostly ETL, Big batches, High Startup time

Hive 1.X

- Stable
- Backwards compatible
- Bug fixes

Version-2

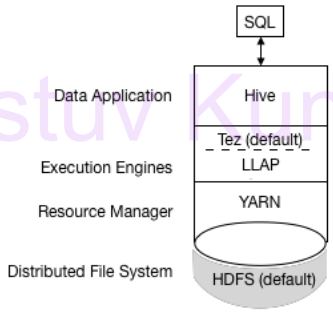
- Tez and Spark execution engines option.
- HBase can store Hive Metadata i.e. HBase Meta-store provide scaling support but lacks transaction addressing via Apache omid hence removed in hive- 3.0
- ACID transaction support.
- HPLSQL : an open source tool implements procedural SQL language for Apache Hive, Spark-SQL, Impala as well as any other SQL-on-Hadoop implementations, NoSQL and RDBMS. http://www.hplsql.org/why
- Live Long And Process (LLAP)
- Cost based Optimization
- Hadoop 2+, java6+ supported
- Does not require Hive server 2, can run HS2 embedded in Beeline

Version-3

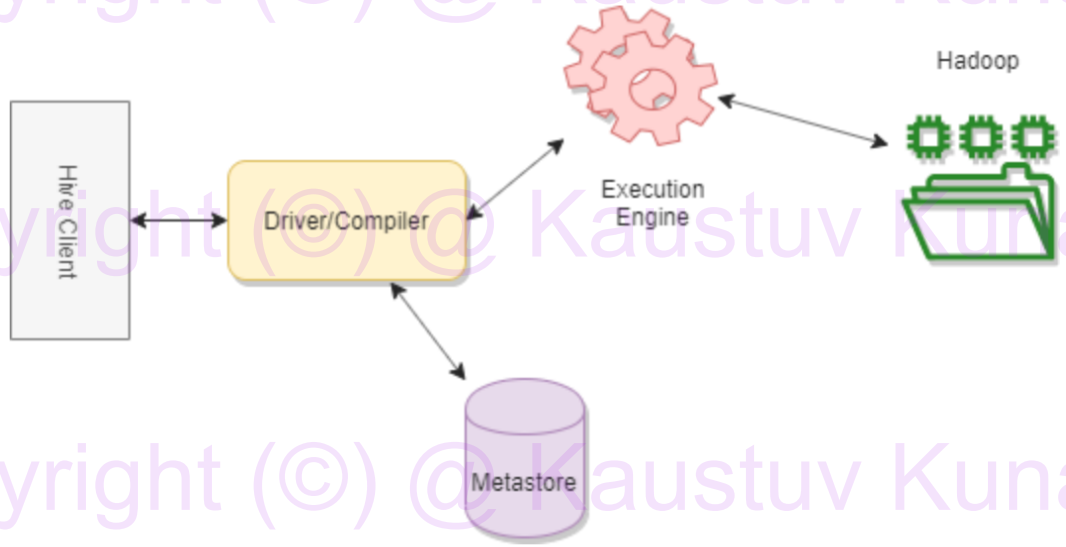
- Optimized for S3/WASB/GCP
- Support for JDBC/Kafka/Druid
- Apache Tez replaces MapReduce as the default Hive execution engine
- MapReduce engine is deprecated
- In ACID side,
  - ✓ ACID Storage has been removed
  - ✓ Added MERGE support (CDC can regularly merged into fact table with upsert functionality)
  - ✓ Removed restrictions so tables no longer be bucketed
  - ✓ Non ORC based table supported (Insert and Select only)
- In SQL side,
  - ✓ Materialized view
  - ✓ LLAP workload management
  - ✓ Query result caching
  - ✓ Hive on Kubernetes

Hive (Version-3) Flow

- Hive compiles the query
- Tez executes the query
- YARN allocates resources for applications across the cluster and enables authorization for Hive jobs in YARN queues
- Hive updates the data in HDFS or the Hive warehouse, depending on the table type
- Hive returns query results over a JDBC connection



Hive Architecture

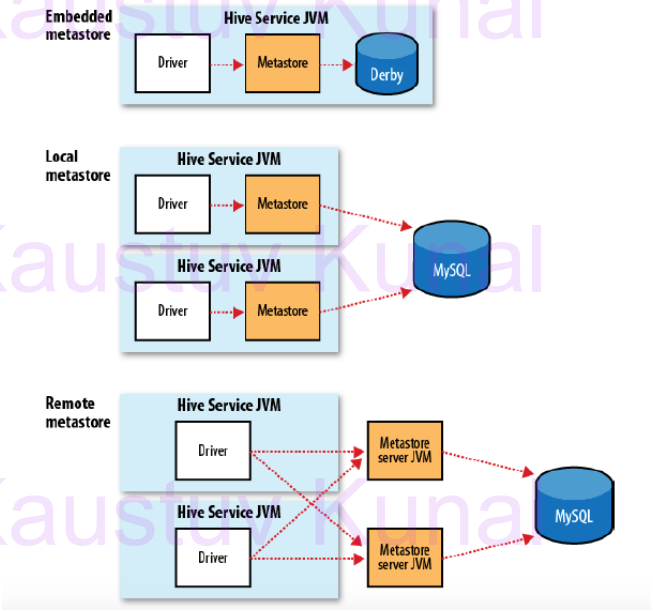


Hive Components

1. **UI**: Interface to submit queries, supports both command line interface and web based GUI
2. **Driver**: Component which receives the queries
3. **Compiler**: Component that parses the query, does semantic analysis on different query blocks and query expressions. Generates an execution plan with the help of the table and partition metadata looked up from the metastore
4. **Execution Engine**: Component which executes the execution plan created by the compiler. The plan is a DAG of stages. The execution engine manages the dependencies between these different stages of the plan and executes these stages on the appropriate system components
5. **Metastore**: Component that stores all the structure information of the various tables and partitions in the warehouse including column and column type information, the serializers and deserializers necessary to read and write data and the corresponding HDFS files where the data is stored

Meta-Store

- Central repository of Hive metadata
- Default hive uses derby metadata and embedded JVM allows only single session at a time
- **Standalone meta-store** : metastore service still runs in the same process as the Hive service, but connects to a database running in a separate process
- **Remote meta-store** :one or more meta-store servers run in separate processes to the Hive service, better manageability and security because the database tier can be completely firewalled off, and clients no longer need the database credentials



Execution Engine

- **MapReduce** : MapReduce is a purely batch framework. Queries using it may experience higher latencies (tens of seconds), even over small datasets.
- **Apache Tez** : Apache Tez is designed for interactive query, and has substantially reduced overheads versus MapReduce.
- **Apache Spark** :Apache Spark is a cluster computing framework that's built outside of MapReduce, but on top of HDFS, with a notion of composable and transformable distributed collection of items called Resilient Distributed Dataset (RDD) which allows processing and analysis without traditional intermediate stages that MapReduce introduces.

Note *: Users are free to switch back and forth between these frameworks at any time*

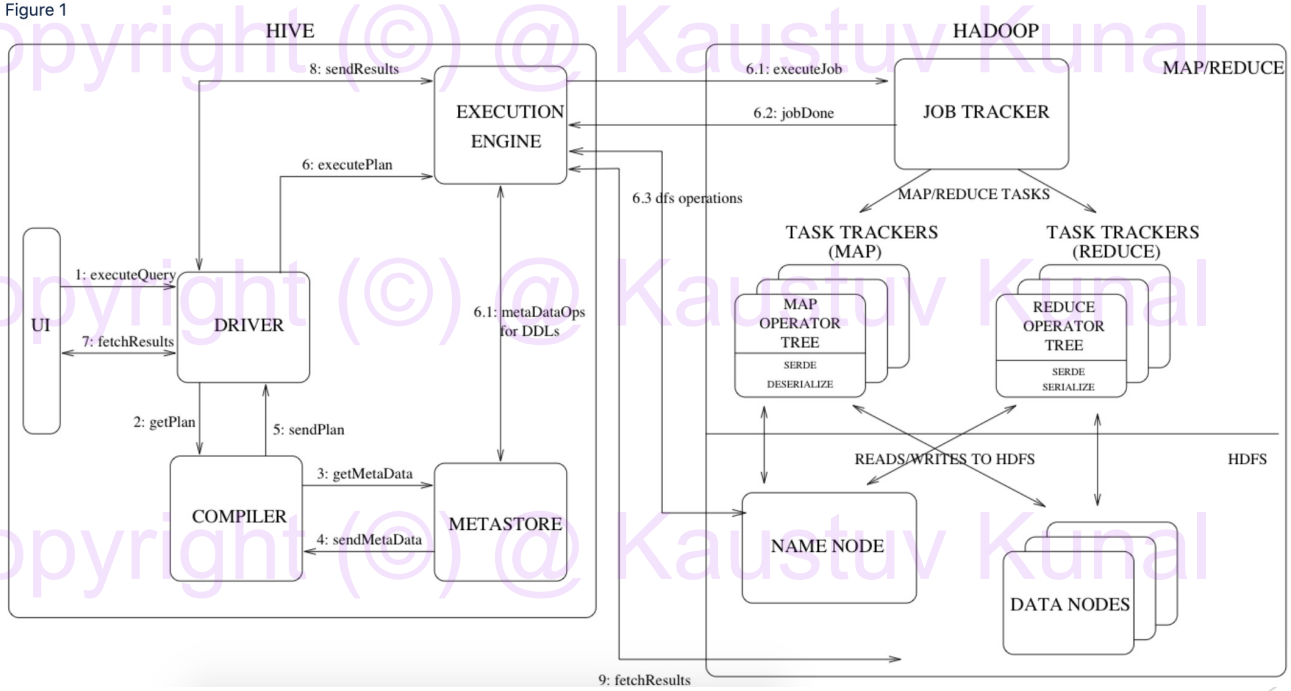
Execution Engine

- Tez and Spark can avoid replication overhead by writing the intermediate output to local disk, or even store it in memory (at the request of the Hive planner).
- The execution engine is controlled by the `hive.execution.engine` property.
- Set Hive to use Tez as follows:

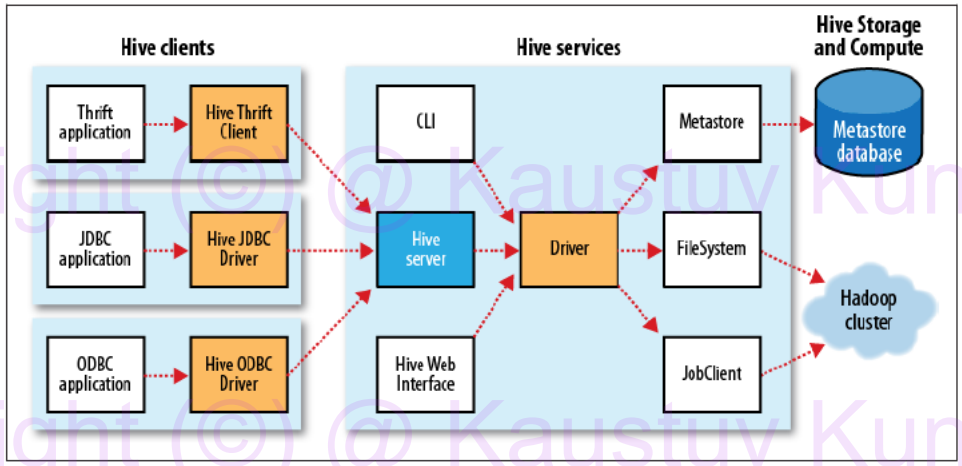
hive> SET hive.execution.engine=tez;

Hive Control Flow

Figure 1



Hive Client-Service Architecture



Hive Services

- **CLI**: Command-line interface to Hive (the shell This is the default service)
- **Hiveserver2** : Improves original Hive server by supporting authentication and multi-user concurrency. Applications using the Thrift, JDBC, and ODBC connectors need to run a Hive server to communicate with Hive.
- **Beeline**: Command-line interface to Hive that works in embedded mode(shell) or by connecting to a Hive server 2 process using JDBC
- **Jar** : The Hive equivalent of Hadoop jar
- **Metastore**: By default, the metastore is run in the same process as the Hive service. Using this service, it is possible to run the metastore as a standalone (remote) process
- **HWI**: Hive web interface. Component removed after hive 2.2.0

Hive Clients

If Hive is run as a server (hive --service hiveserver2), there are a number of different mechanisms for connecting to it from applications, primary are:

- **Thrift Client** : for accessing hive via Python and Ruby applications
- **JDBC Driver**: for accessing hive via java (beeline)
- **ODBC Driver**: for accessing via various business applications that support ODBC protocol

Hive Installation Steps

1. Download a stable hive release
2. Set Environment variable
3. Create temp and warehouse directory in HDFS
4. Set properties of *hive-env.sh* file

Refer: https://github.com/kaustuvkunal/Hive-Tutorial/blob/master/Hive-Installation.md

Setup MySQL as Hive Metastore

Default hive metastore is derby which has only single session support hence for most practical purpose MySQL metastore is used.

1. Create metastore database in MySql
2. Put mysql-connector-java-(mysql-version).jar into HIVE_HOME/lib/
3. Create the Initial database schema
4. Create MySql user account for hive user
5. Configure hive-default .xml

Refer: https://github.com/kaustuvkunal/Hive-Tutorial/blob/master/mysql-metastore-setup.md

Upgrading Hive Version

- Upgrade the Meta-Store schema by running the appropriate schema upgrade scripts located in the scripts/metastore/upgrade directory.
- Upgrade scripts for MySQL, Postgre-SQL, Oracle, Microsoft SQL Server, and Derby databases. If using a different database for your Metastore you will need to provide your own upgrade script.

Hive Tables

- **Managed Tables** : Reside inside hive warehouse data. Hive manages it. If deleted, data and metadata both gets deleted. When you load data into a managed table, it is moved into Hive's warehouse directory.
- **External Tables** : Reside outside hive warehouse directory . Delete command only deletes meta data from metastore.

Refer : https://github.com/kaustuvkunal/Hive-Tutorial/blob/master/hive-commnads.md

Partitioning

- Partitioning is optimization & storage technique in Hive.
- It store table data by a column values.
- If we partitioned by date column, then records for the same date would be stored in the same partition(hdfs-folder) and queries on that particular partition column will run much efficiently.
- Partitions are defined at table creation time using the PARTITIONED BY clause.
- Table may be partitioned in multiple dimensions (sub partitioning).
- When we load data into a partitioned table, the partition values are specified explicitly.
- However, partitions may be added to or removed from a table after creation, using an ALTER TABLE statement.
- `SHOW PARTITIONS parttiion_tablename` command depicts partitions in a table.

Partitioning Types

- Static Partitioning : Specify the partition column value explicitly in each load statement. Example LOAD DATA INPATH '/hdfs path of the file' INTO TABLE t1 PARTITION(country="US”)
- Dynamic Partitioning : Allow us not to specify partition column value each time. Example ‘INSERT INTO TABLE t2 PARTITION(country) SELECT * from T1;

Bucketing

- Bucketing imposes sampling of tables
- CLUSTERED BY (id) INTO 4 BUCKETS; (for random sampling)
- CLUSTERED BY (id) SORTED BY (id ASC) INTO 4 BUCKETS; (sorted and useful in joining)
- Two tables that are bucketed on the same columns which include the join columns can be efficiently implemented as a map-side join
- Bucketed tables allow more efficient sampling than non-bucketed tables and may later allow for time saving operations such as map-side joins. However, the bucketing specified at table creation is not enforced

Views

- Views are virtual table
- May also be used to restrict user's access to particular subsets of tables
- When we create a view, the query is not run; it is simply stored in the metastore called not materialized to disk.
- So every time you called view its created by a mr job.
- To store views in disc use create table as command

Hive DDL

- Create (Database, Schema, Table, View, Function, Index)
- Drop (Database, Schema, Table, View, Index)
- Truncate (Table)
- Alter(Database, Schema, Table, View)
- Msck (Repair Table [add/drop/sync partitions])
- Show (Database, Schema, Table, TBLPROPERTIES, Views, Partitions, Functions, Index, columns, create table)
- Show (Database, Schema, Table, TBLPROPERTIES, Views, Partitions, Functions, Index, columns, create table)
- Describe (Database, Schema, Table, View,)
- Locks (table, partition)

Hive DML

- Loading files into tables
- Inserting data into Hive Tables from queries
- Writing data into the file system from queries
- Inserting values into tables from SQL

Refer : https://github.com/kaustuvkunal/Hive-Tutorial/blob/master/hive-commnads.md

Hive Data Types

Category	Type	Description	Literal examples
Primitive	BOOLEAN	True/false value	TRUE
	TINYINT	1-byte (8-bit) signed integer, from -128 to 127	1Y
	SMALLINT	2-byte (16-bit) signed integer, from -32,768 to 32,767	1S
	INT	4-byte (32-bit) signed integer, from -2,147,483,648 to 2,147,483,647	1
	BIGINT	8-byte (64-bit) signed integer, from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	1L
	FLOAT	4-byte (32-bit) single-precision floating-point number	1.0
	DOUBLE	8-byte (64-bit) double-precision floating-point number	1.0
	DECIMAL	Abitrary-precision signed decimal number	1.0
	STRING	Unbounded variable-length character string	'a', "a"
	VARCHAR	Variable-length character string	'a', "a"
	CHAR	Fixed-length character string	'a', "a"
	BINARY	Byte array	Not supported
	TIMESTAMP	Timestamp with nanosecond precision	1325502245000, '2012-01-02 03:04:05.123456789'
	DATE	Date	'2012-01-02'
Complex	ARRAY	An ordered collection of fields. The fields must all be of the same type.	array(1, 2)[a]
	MAP	An unordered collection of key-value pairs. Keys must be primitives; values may be any type. For a particular map, the keys must be the same type, and the values must be the same type.	map('a', 1, 'b', 2)
	STRUCT	A collection of named fields. The fields may be of different types.	struct('a', 1, 1.0)[b], named_struct('col1', 'a', 'col2', 1, 'col3', 1.0)
	UNION	A value that may be one of a number of defined data types. The value is tagged with an integer representing its data type in the union (by 0-index).	create_union(1, 'a', 63)

Sorting Options in Hive-QL

ORDER BY : Guarantees global ordering, but does this by pushing all data through just one reducer. This is basically unacceptable for large datasets. You end up one sorted file as output.

SORT BY : Orders data at each of N reducers, but each reducer can receive overlapping ranges of data. You end up with N or more sorted files with overlapping ranges.

DISTRIBUTE BY : Ensures each of N reducers gets non-overlapping ranges of x, but doesn't sort the output of each reducer. You end up with N or unsorted files with non-overlapping ranges.

CLUSTER BY x: Ensures each of N reducers gets non-overlapping ranges, then sorts by those ranges at the reducers. This gives you global ordering, and is the same as doing (DISTRIBUTE BY x and SORT BY x). You end up with N or more sorted files with non-overlapping ranges.

Group By: Groups the data by select column-id and use to perform operations like sum, avg.

Property Precedence Hierarchy

1. The Hive SET command
2. The command-line -hiveconf option
3. hive-site.xml and the Hadoop site files (core-site.xml, hdfs-site.xml, mapredsite.xml, and yarn-site.xml)
4. The Hive defaults and the Hadoop default files (core-default.xml, hdfs-default.xml,mapred-default.xml, and yarn-default.xml)

Hcatalog & WebHCat

- HCatalog is a table storage management tool for Hadoop. It exposes the tabular data of Hive metastore to other Hadoop applications. It enables users with different data processing tools (Pig, MapReduce) to easily write data onto a grid. It ensures that users don't have to worry about where or in what format their data is stored.

- WebHCat is the REST API for Hcatalog . It provides a service that you can use to run Hadoop MapReduce (or YARN), Pig, Hive jobs or perform Hive metadata operations using an HTTP (REST style) interface.

Joins

- Inner joins
- Outer joins (left outer join and right outer join)
- Semi joins
- Map joins

Refer : https://github.com/kaustuvkunal/Hive-Tutorial/blob/master/hive-commnads.md

Map -Join

- For join Query optimization
- Let small table load into memory for faster execution
- Can be done explicitly by mentioning /*MAPJOIN(table_name)*/ or set properties in configuration files to automatically does that

Semi join

- Selecting of table part which is not in other table

ACID

1. **Atomic** :Each transaction is treated as a single unit either 0 or 1.
2. **Consistency:** Ensures that a transaction can only bring the database from one valid state to another. This prevents database corruption by an illegal transaction, but does not guarantee that a transaction is *correct*. Once an application performs an operation the results of that operation are visible to it in every subsequent operation)
3. **Isolate :**Concurrent execution of transactions leaves the database in the same state that would have been obtained if the transactions were executed sequentially. An incomplete operation by one user does not cause unexpected side effects for other users.
4. **Durable :**Once a transaction has been committed, it will remain committed even in the case of a system failure.

With Addition of transactions in Hive 0.13 it is now possible to provide full ACID semantics at the row level.

Why Transactions Added in Hive?

- Overcome Dirty reads : System see data written after they had started their queries
- Inserts and update requirement of individual records due to dimension/Schema change
- Delete requirement (*INSERT*, *UPDATE*, and *DELETE*.)
- Bulk update (SQL Merge)

Enabling ACID Support

Add transactional properties in hive-site

- ✓ hive.support.concurrency – true
- ✓ hive.enforce.bucketing – true (Not required as of Hive 2.0)
- ✓ hive.exec.dynamic.partition.mode – nonstrict
- ✓ hive.txn.manager – org.apache.hadoop.hive.ql.lockmgr.DbTxnManager
- ✓ hive.compactor.initiator.on – true (for exactly one instance of the Thrift metastore service)
- ✓ hive.compactor.worker.threads – a positive number on at least one instance of the Thrift metastore service

Hive Transaction Support

Supports for :

- ✓ Insert
- ✓ Delete
- ✓ Update
- ✓ Merge

Refer: https://github.com/kaustuvkunal/Hive-Tutorial/blob/master/Hive-Installation.md

Transactions basics

How transaction works when HDFS does not supports file update ?

- Data for the table or partition is stored in a set of base files.
- New records, updates, and deletes are stored in delta files.
- A new set of delta files is created for each transaction (or in the case of streaming agents such as Flume or Storm, each batch of transactions) that alters a table or partition.

- At read time the reader merges the base and delta files, applying any updates and deletes as it reads.
- Any partitions (or tables) written with an ACID aware writer will have a directory for the base files and a directory for each set of delta files.

Compactor

Set of background processes running inside the Metastore to support ACID system, major process are:

- Initiator : Discovering which tables or partitions are due for compaction.
- Worker: Each Worker handles a single compaction task. A compaction is a MapReduce job.
- Cleaner : Process that deletes delta files after compaction.
- AcidHouseKeeperService : Aborts timeout transaction and release resource.

Delta File Compaction

- As operations modify the table more and more delta files are created and need to be compacted to maintain adequate performance. There are two types of compactions, minor and major
  - **Minor compaction** takes a set of existing delta files and rewrites them to a single delta file per bucket.
  - **Major compaction** takes one or more delta files and the base file for the bucket and rewrites them into a new base file per bucket. Major compaction is more expensive but is more effective
- After a compaction the system waits until all readers of the old files have finished and then removes the old files
- *SHOW COMPACTIONS* : displays information about currently running compaction and recent history

Transaction Limitations

- *BEGIN*, *COMMIT*, and *ROLLBACK not supported yet*
- Only ORC file format is supported but anything implementing format
- Tables must be bucketed to make use of these features
- External tables cannot be made ACID tables
- Reading/writing to an ACID table from a non-ACID session is not allowed the
- LOAD DATA... statement is not supported with transactional tables

Future work on Transaction

- Support for BEGIN/COMMIT/ROLL BACK
- Smarter compaction
- User defined primary keys
- Better monitoring/alert
- Finer grained Concurrency management/conflict detection

Storage

- Row Format
  - Row format is defined by a SerDe, a portmanteau word for a Serializer-Deserializer
  - Default row delimiter is Control-A character.
  - Default collection item delimiter is a Control-B character
  - The default map key delimiter is a Control-C character
- File Format
  - Default file format is A **plain-text file**, but there are row-oriented and column-oriented binary formats available too.
  - There is no means for escaping delimiter characters in Hive, so it is important to choose ones that don't occur in data fields

Other Storage Formats

- ORC Files
- Sequence Files
- RC Files
- Parquet files
- Avro-data files

ORC

- Default File format supported by Hive Transactions
- Optimized Row Columnar (ORC)
- Designed to store hive efficiently and to perform read , write faster
- ORC Advantages :
  - ✓ A single file as the output of each task hence reduces the Name node's load
  - ✓ Light-weight indexes stored within the file
  - ✓ Block-mode compression based on data type
  - ✓ Concurrent reads of the same file using separate Recordreaders

Sequence Files

- Hadoop flat files, stores values in binary key-value pairs
- Easily splitable and mergeable

RC Files

- Record Columnar File
- Binary format: Flat file consisting of binary key/value pairs
- Columnar storage : Partitions rows horizontally into row splits, and then it vertically partitions each row split in a columnar way
- Ensure data in the same row are located in the same node

Parquet Files

- A compressed, efficient columnar data representation

- Supports very efficient encoding schemes, fast processing and is splitable.

- Parquet file consists of a header followed by one or more blocks, and terminated by a footer
- all the file metadata is stored in the footer

Avro File

Avro is serialization format , it provides,

- ✓ Rich data structures
- ✓ A compact, fast, binary data format

- ✓ A container file, to store persistent data
- ✓ Remote procedure call

- ✓ Simple integration with dynamic languages
- ✓ Schema at top of file

Hive Indexes

Hive supported below two types of indexes

1. **Compact** : Compact indexes store the HDFS block numbers of each value, rather than each file offset, so they don't take up much disk space but are still effective for the case where values are clustered together in nearby rows.

1. **Bitmap** :Bitmap indexes use compressed bit sets to efficiently store the rows that a particular value appears in, and they are usually appropriate for low-cardinality columns (such as gender or country).Bitmap index stores(value, list of row as bitmap) while compact Index stores (value, block-id)

Why Indexes are not Recommended in Hive ?

- ORC has build in Indexes which allow the format to skip blocks of data during read, they also support Bloom filters.
- Indexing Is Removed since 3.0
- Alternative to indexing is materialized view.
- Using columnar file formats (Parquet, ORC) – they can do selective scanning; they may even skip entire files/blocks.
- Indexes unsupported for Tez execution engine

Hive logging

- logging configuration is in *conf/hive-log4j.properties* and you can edit this file to customize logging

UDF(User Defined Function)

- Allow user to extend Hive-QL
- Implemented in java, acted like a built in function(include online help)
- Extend UDF, write one or more evaluate method with a Hadoop writable return type.
- Hive's SQL can also be extended with user code via user defined functions (UDFs), user defined aggregates (UDAFs), and user defined table functions (UDTFs).
- UDF Types
  - ✓ UDF : Input single row and output single row
  - ✓ UDAF : For multiple input rows creates a single output row
  - ✓ UDTF : Operates on a single row and produces multiple rows

Hive Alternatives

- Presto from Facebook
- Apache Drill
- Spark SQL
- Impala from Cloudera

Reference

- Facebook hive paper
- https://docs.hortonworks.com/HDPDocuments/HDP3/HDP-3.0.0/hive-overview/content/hive-apache-hive-3-architecturural-overview.html
- The Definitive guide 4th edition (chapter 17)
- http://hive.apache.org