CLOUDERA

Modernizing Hive database engine

Peter Vary, Marton Bod

AGENDA

Evolution of Hive

- History
- LLAP, ACID

Future: Hive-on-cloud, Unified Analytics, Iceberg

Hive ACID deep-dive

- Usage
- Behind the scenes
- Configuration

What is Apache Hive?

- Open-source data warehousing software
- Read/write/manage large data sets in distributed storage
- SQL-like query language (translated to Tez or MapReduce jobs)
- Optimal for batch processing, ETL jobs

Evolution of Hive

Some major historical milestones

- Hive CLI
  - Query compilation runs locally, submits MR jobs to Hadoop cluster
- HiveServer2
  - Client-server architecture, access control of metadata
- Separate HiveMetaStore
  - HMS in separate process, queryable by other engines
- New execution engines
  - Tez and Spark storing intermediate data in memory, saving on disk IO

LLAP and ACID

The present state of Hive

- LLAP: Improves query performance
  - LLAP daemons: Long-running executor containers
  - LLAP IO cache: Caching S3 data files in-memory and/or on local SSD

- ACID: ability to delete/update data atomically on append-only storage media (S3/HDFS)

AGENDA

Evolution of Hive

- Overview and history
- LLAP, ACID

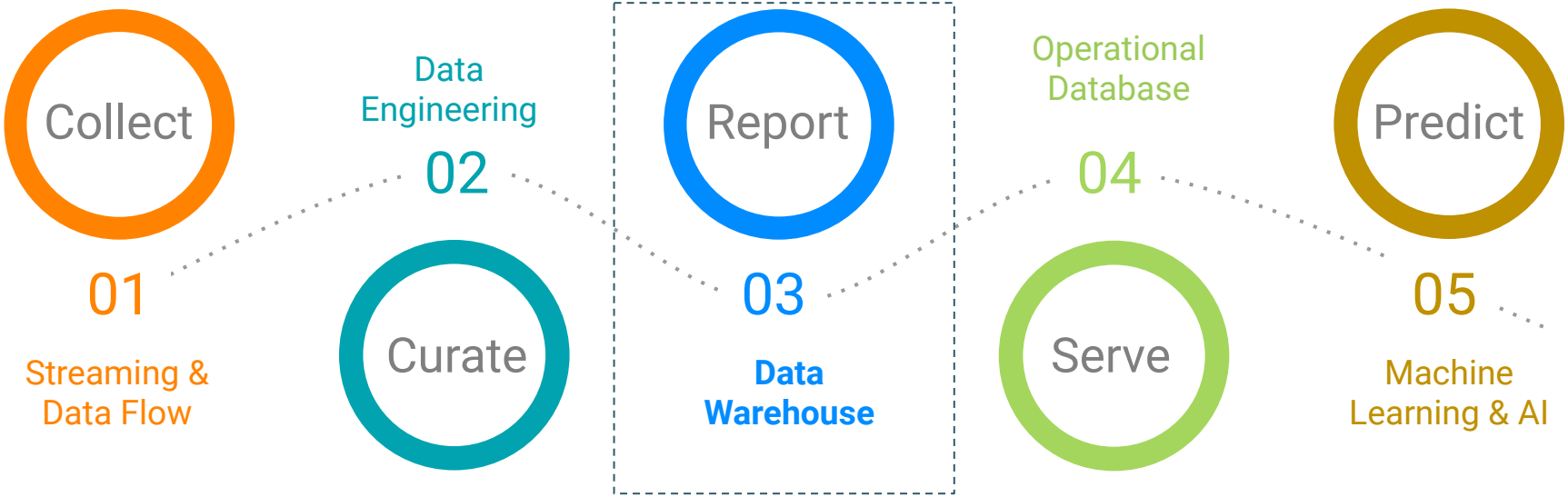
Future: Hive-on-cloud, Unified Analytics, Iceberg

Hive ACID deep-dive

- Usage
- Behind the scenes
- Configuration

Cloudera Data Platform (CDP): solutions from Edge to AI

Manage the complete data lifecycle in any cloud or datacenter

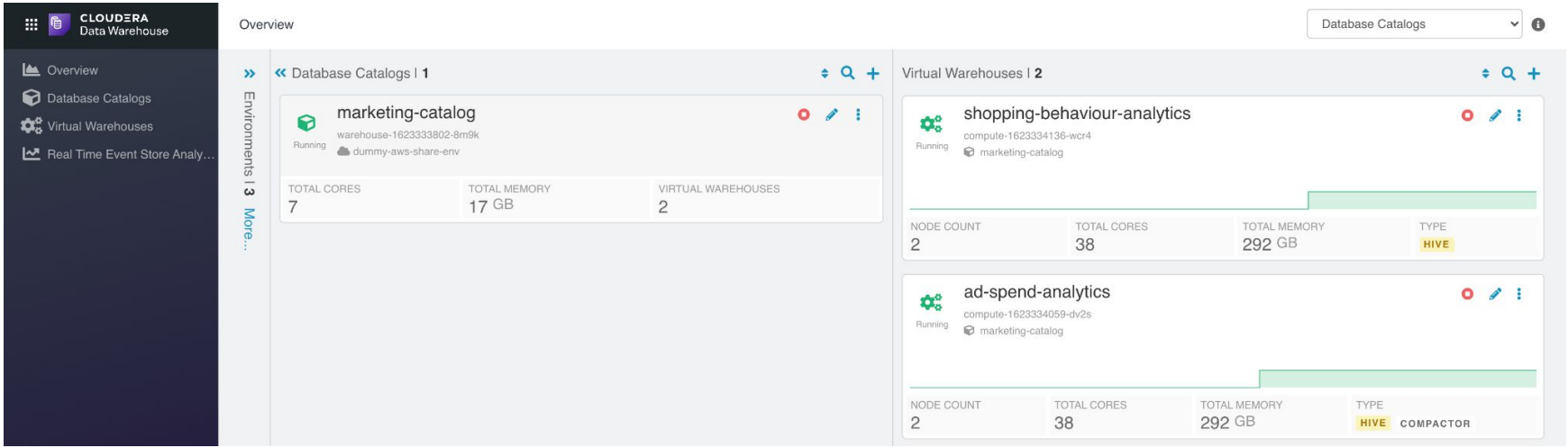


POWERED BY CLOUDERA SDX

Security | Governance | Lineage | Management | Automation

Cloudera Data Warehouse: Hive in the cloud

Containerized, autoscaling Hive resources

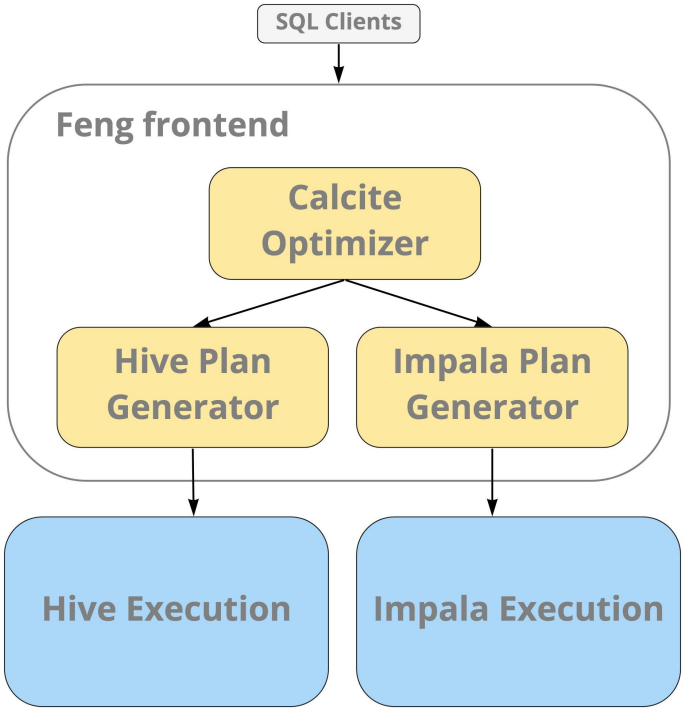


- Self-service creation of new, independent metadata catalogs and compute resources
  - Containerized Hive deployments in Kubernetes
- Autoscale up and down to cope with varying workload levels
- Compliant with the security controls associated with your data lake

Unified Analytics

A common frontend for query engines

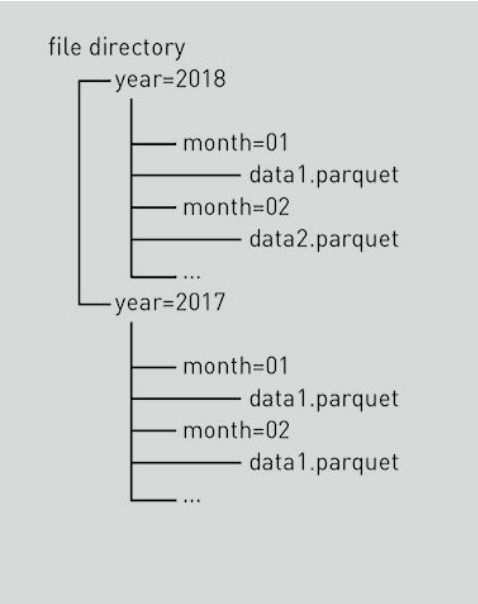
- A common SQL compiler and planner for Hive and Impala
- Automatic query routing depending on workload type and use case
- Supports optimizations e.g. materialized view rewriting, join reordering
- Result cache for repetitive queries

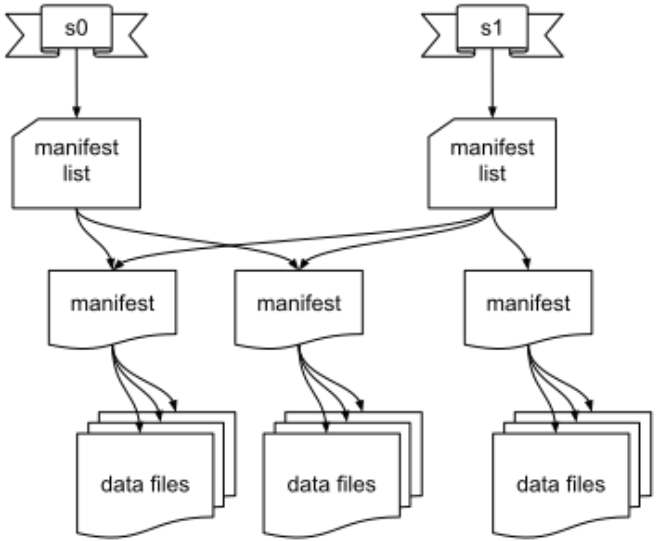


Iceberg table support

Next-generation table format

- A new way of tracking the data files of your table





Iceberg main benefits

Next-generation table format

- Concurrent reads/writes without locking
- Lower pressure on HMS (partitions are tracked in FS)
- No expensive S3 partition listings (data files are tracked in FS)
- Time travel: SELECT * FROM customers AS OF TIMESTAMP ‘2021-03-05’;
- Partition transforms (e.g. month/day/hour(timestamp)) and evolution

AGENDA

Evolution of Hive

- History
- LLAP, ACID

Future: Hive-on-cloud, Unified Analytics, Iceberg

Hive ACID deep-dive

- Usage
- Behind the scenes
- Configuration

ACID - Full transactional tables

Who does not want to update rows

- Allows modification of existing Hive table data using ACID semantics
  - ACID: Atomicity, Consistency, Isolation, Durability
  - Insert, Update, Delete, Merge
- Allows concurrent updates with long-running analytic queries
  - Support SQL Update/Delete/Merge
- Uses MVCC (Multi-Version Concurrency Control) Architecture
  - Changes are stored in Delta files
  - Reads merged base and deltas
- Not OLTP
  - Supports relatively Low rate of transactions
  - Not a replacement for MySql or HBase

SQL syntax to create / use transactional tables

SQL standard

- **CREATE TABLE T(a int,b int)** STORED AS ORC TBLPROPERTIES ('transactional'='true');
- Restrictions:
  - Managed Table
  - Table cannot be sorted
  - Currently stores the data in ORC
  - Uses AcidInputFormat/AcidOutputFormat
  - Bucketing is optional!
- If upgrading from Hive 2
  - Requires Major Compaction before Upgrading

Design

Transaction Manager and Storage layer

- Transaction Manager
  - Snapshot Isolation, each reader “locks in” a logical snapshot of the database
  - Repeatable Read without Phantom Inserts
- Storage layer enhanced to support MVCC architecture:
  - Multiple versions of each row
  - Allow concurrent readers and writers
  - Write operations do not require exclusive locks

Storing the changes

ACID V1 vs. V2

{<Original writeId>, <BucketId>, <RowId>}

- Update = delete + insert
- UPDATE acidTbl SET b = “bar“ WHERE a = 300;

Row Metadata	a	b
{1, 0, 0}	100	“abc”
{1, 0, 2}	300	“bee”
{1, 0, 1}	200	“xyz”

/delta_00001_00001/bucket_0000

Row Metadata	a	b
{2, 0, 0}	300	“bar”

/delta_00002_00002/bucket_0000

Row Metadata	a	b
{1, 0, 2}	*null*	*null*

/delete_delta_00002_00002/bucket_0000

Storing the changes

Filesystem

```
table1/
├── delta_001_001/
│   └── 0000
├── delete_delta_002_002/
│   └── 0000
├── delta_002_002/
│   └── 0000
└── delete_delta_003_003/
    └── 0000
```

```
table2/
├── base_100/
│   └── 0000
├── delta_101_103/
│   └── 0000
└── delta_104_104/
    └── 0000
```

Transactions / writes are stored in the Hive Metastore

Single source of truth

- Transaction State
  - Open, Committed, Aborted
- Reader snapshot isolation:
  - A snapshot is the state of all transactions
  - High Water Mark + List of Exception

```
table1/
├── delta_001_001/
│   └── 0000
├── delete_delta_002_002/
│   └── 0000
├── delta_002_002/
│   └── 0000
└── delete_delta_003_003/
    └── 0000
```

- Atomicity & Isolation

Compaction

Solution for the small files problem

- Compactor rewrites the table in the background
  - Major compactor merges deltas to a base - more expensive
  - Minor compaction - merges delta files into fewer deltas
- Compaction can be triggered automatically or on demand
  - There are various configurations to govern when and how it starts
  - Compactor has two implementations, Map / Reduce and Query-based
- Key design principle
  - Initiator, Worker (compactor) and Cleaner all run in background and do not affect readers/writers
  - Cleaner process – removes obsolete files
  - Requires Standalone metastore

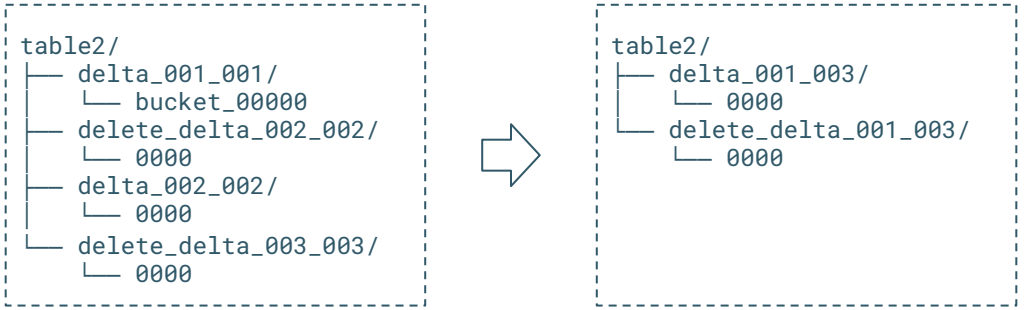
Compaction

What it does

Minor

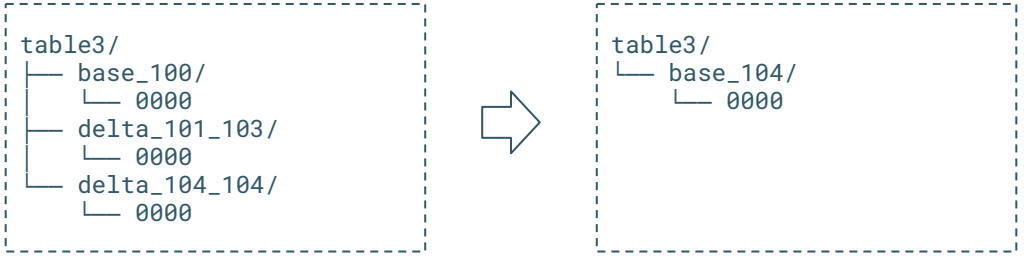
Merge files into:

- 1 delta dir
- 1 delete_delta dir



Major:

Merge files into 1 base dir



Some compaction configuration possibilities

Resource usage or performance

Config name	Default	Description
hive.compactor.initiator.on	false	Initiator and Cleaner threads on this metastore instance.
hive.metastore.runworker.in	HS2	Specifies where to run the Worker threads.
hive.compactor.worker.threads	0	How many compactor worker threads to run per instance.
hive.compactor.delta.num.threshold	10	Number of delta directories that triggers a minor compaction.
hive.compactor.delta.pct.threshold	0.1	Percentage size of the delta files relative to the base that triggers a major compaction.
hive.compactor.job.queue		Specifies the Hadoop queue name to which compaction jobs are submitted.

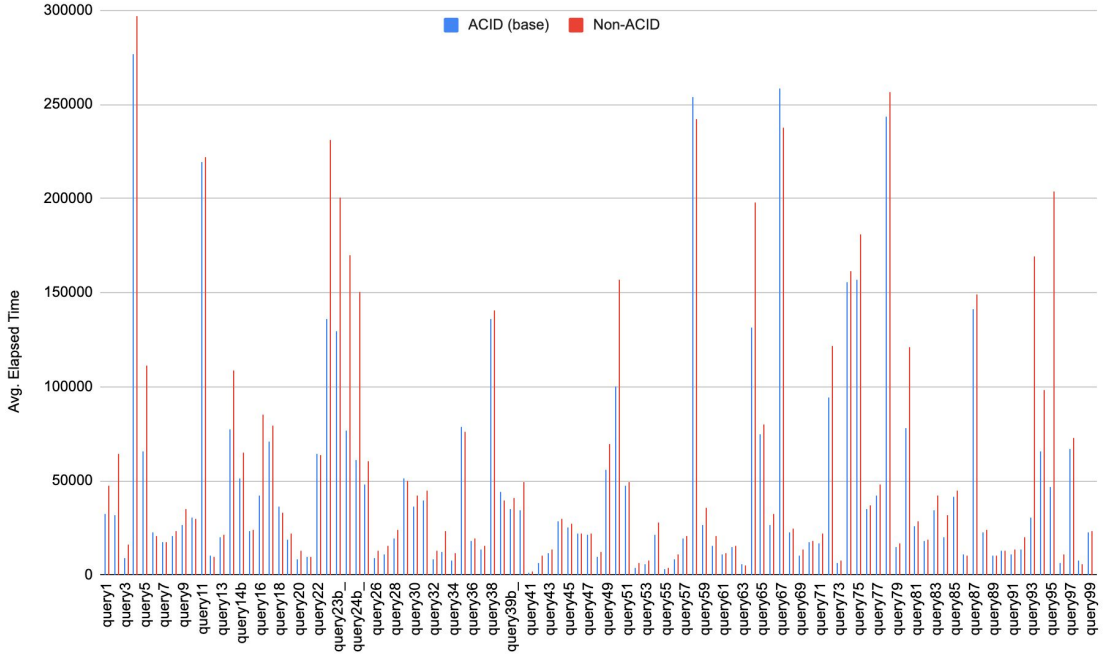
Performance numbers

System under test

- Cluster: CDP Private Cloud Base 7.1.4
- Cluster size: 10 Node
  - Intel(R) Xeon(R) Silver 4114 CPU @ 2.20GHz, 128GB RAM, 20 cores
- Engine: HiveOnTez
- Storage: HDFS
- Data size: 1 Tb
- Data pollution: 1 base / 20% updates
- Concurrency: 1 client

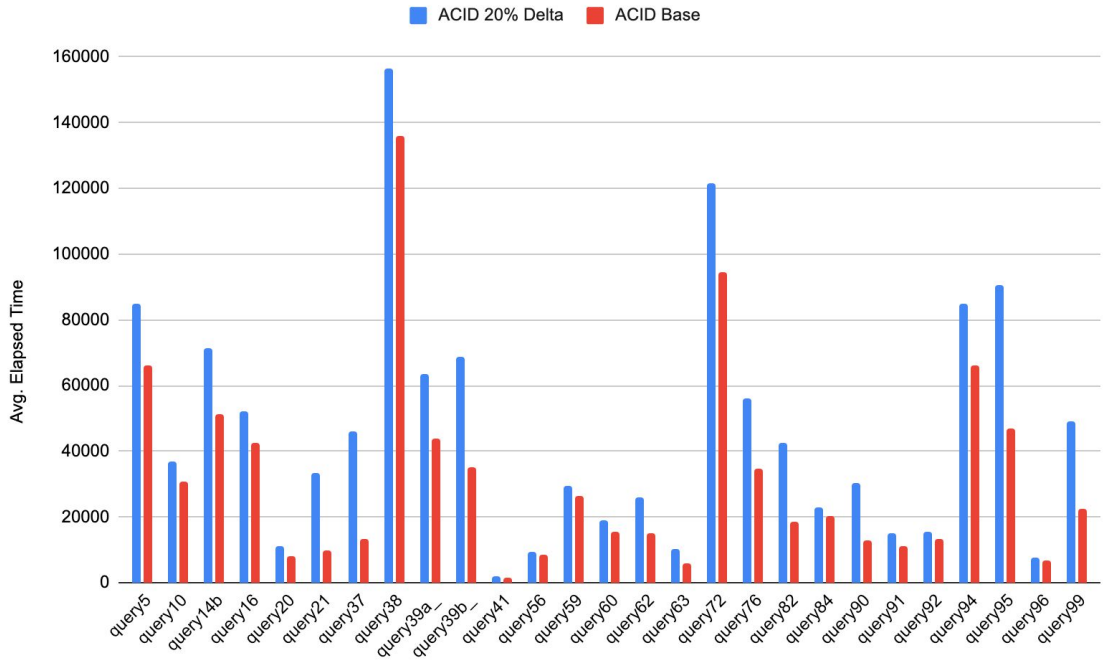
Performance numbers

ACID Base



Performance numbers

ACID 20% delta



THANK YOU

WE ARE HIRING

CLOUDERA