Cloudera Runtime 1.0.0

Managing Apache Hive

Date published: 2020-11-30
Date modified: 2025-10-22

CLOUDERA

Legal Notice

Contents

ACID operations in Cloudera Data Warehouse

Apache Hive supports ACID (atomicity, consistency, isolation, and durability) v2 transactions at the row level without any configuration. Knowing what this support entails helps you determine the table type you create.

By default, managed tables are ACID tables. You cannot disable ACID transactions on managed tables, but you can change the Hive default behavior to create external tables by default to mimic legacy releases. Application development and operations are simplified with strong transactional guarantees and simple semantics for SQL commands. You do not need to bucket ACID v2 tables, so maintenance is easier. With improvements in transactional semantics, advanced optimizations, such as materialized view rewrites and automatic query cache, are available. With these optimizations, you can deploy new Hive application types.

A Hive operation is atomic. The operation either succeeds completely or fails; it does not result in partial data. A Hive operation is also consistent: After an application performs an operation, the results are visible to the application in every subsequent operation. Hive operations are isolated. Your operations do not cause unexpected side effects for other users. Finally, a Hive operation is durable. A completed operation is preserved in the event of a failure.

Hive operations are atomic at the row level instead of the table or partition level. A Hive client can read from a partition at the same time another client adds rows to the partition. Transaction streaming rapidly inserts data into Hive tables and partitions.

Viewing transactions in Cloudera Data Warehouse

As Administrator, you can view a list of open and aborted transactions.

Procedure

Enter a query to view transactions.
SHOW TRANSACTIONS
The following information appears in the output:

- Transaction ID
- Transaction state
- Hive user who initiated the transaction
- Host machine or virtual machine where transaction was initiated

Viewing transaction locks

As a Hive administrator, you can get troubleshooting information about locks on a table, partition, or schema.

Procedure

1. Enter a Hive query to check table locks.

```
SHOW LOCKS mytable EXTENDED;
```

2. Check partition locks.

```
SHOW LOCKS mytable PARTITION(ds='2018-05-01', hr='12') EXTENDED;
```

3. Check schema locks.

```
SHOW LOCKS SCHEMA mydatabase;
```

Output generally appears as follows.

- Database name
- Table name
- Partition, if the table is partitioned
- Lock state:

  - Acquired - transaction initiator hold the lock
  - Waiting - transaction initiator is waiting for the lock
  - Aborted - the lock has timed out but has not yet been cleaned
- Lock type:

  - Exclusive - the lock cannot be shared
  - Shared_read - the lock cannot be shared with any number of other shared_read locks
  - Shared_write - the lock may be shared by any number of other shared_read locks but not with other shared_write locks
- Transaction ID associated with the lock, if one exists
- Last time lock holder sent a heartbeat
- Time the lock was acquired, if it has been acquired
- Hive user who requested the lock
- Host machine or virtual machine on which the Hive user is running a Hive client
- Blocked   By ID - ID of the lock causing current lock to be in Waiting mode, if the lock is in this mode

Related Information

Apache wiki transaction configuration documentation

Data compaction

As administrator, you need to manage compaction of delta files that accumulate during data ingestion. Compaction is a process that performs critical cleanup of files.

Hive creates a set of delta files for each transaction that alters a table or partition. By default, compaction of delta and base files occurs at regular intervals. Compactions occur in the background without affecting concurrent reads and writes.

There are two types of compaction:

- Minor

  Rewrites a set of delta files to a single delta file for a bucket.
- Major

  Rewrites one or more delta files and the base file as a new base file for a bucket.

  Carefully consider the need for a major compaction as this process can consume significant system resources and take a long time. Base and delta files for a table or partition are compacted.

  You can configure automatic compactions or do manual compactions. Start a major compaction during periods of low traffic. You use an ALTER TABLE statement to start compaction manually. A manual compaction either returns the accepted compaction request ID or shows the ID (and current state) of a compaction request for the very same target. The request is stored in the COMPACTION_QUEUE table.

The compactor initiator must run on only one HMS instance at a time.

Compaction tasks

Compaction in Hive goes hand-in-hand with Hive ACID. Compaction is not, however, necessarily required for Hive ACID. You need to understand when you want, or do not want, compaction to occur.

If you confine your ACID operations tables to full reloads and some delta merges, the performance is steady. As tables are always rewritten (dropped and recreated), there is no need for compaction. Consider disabling compaction by

Compaction occurs for the following reasons:

- You explicitly trigger compaction on a table or partition.
- Automatic compaction finds something to compact.

You run an ALTER TABLE statement to start explicit compaction. Automatic compaction happens without your intervention when Hive periodically crawls through the transaction information, finds tables/partitions affected and those fit for the pre-defined conditions, and marks them for compaction. Conditions are based on the number of deltas, amount of change, and so on.

Compaction of sorted tables is not supported.

In a data pipeline, creating staging or temporary tables can significantly increase the compaction throughput. Avoid compaction of these tables.

Starting compaction manually

You manually start compaction when automatic compaction fails for some reason. You can start compaction by running a Hive statement.

About this task

You can run compaction pseudo-synchronously using the AND WAIT clause. Compaction actually occurs asynchronously, but seems synchronous. The compaction request is recorded and queued, and remains in a waiting cycle, querying the status of the compaction in the background until a failure, success, or timeout occurs. The hive.compactor.wait.timeout (default: 300s) property sets the timeout.

Start compaction using a query

You use the following syntax to issue a query that starts compaction:

```
  ALTER TABLE tablename [PARTITION (partition_key='partition_value' [,...])]
 COMPACT 'compaction_type'
```

Required role: DWAdmin

Before you begin

- Files you are compacting must be in the ORC format.
- Compaction must be enabled (initiator hive.compactor.initiator.on=true)

1. Run a query to start a major compaction of a table.

```
ALTER TABLE mytable COMPACT 'major'
```

Use the COMPACT 'minor' clause to run a minor compaction. ALTER TABLE compacts tables even if the
NO_AUTO_COMPACTION table property is set.

2. Start compaction in a pseudo-synchronous way.
ALTER TABLE mydb.mytable PARTITION (mypart='myval') COMPACT 'MAJOR' AND WAIT;

Viewing compaction progress

You view the progress of compactions by running Hive queries.

About this task
The 'SHOW COMPACTIONS' statement is not authorized. Every user can view compactions and can see the current
state of compactions.

Procedure

Enter the query to view the progress of compactions.
SHOW COMPACTIONS;

- Unique internal ID
- Database name
- Table name
- Partition name
- Major or minor compaction
- Compaction state:

  - Initiated - waiting in queue
  - Working - currently compacting
  - Ready for cleaning - compaction completed and old files scheduled for removal
  - Failed - the job failed. Details are printed to the metastore log.
  - Succeeded
  - Attempted - initiator attempted to schedule a compaction but failed. Details are printed to the metastore log.
- Thread ID
- Start time of compaction
- Duration
- Job ID - ID of the submitted MapReduce job

Monitoring compaction

You can query the SYS database to monitor compactions.

About this task

Querying the COMPACTIONS table provides information about completed and queued compactions.

Run a query to find failed compactions.

```
SELECT * FROM SYS.COMPACTIONS WHERE C_STATE IN ("failed", "did not initiate"
);
```

Disabling automatic compaction

You can disable automatic compaction of a particular Hive ACID table by setting a Hive table property. By default, compaction is enabled, so you must enter an ALTER TABLE command to disable it.

About this task

Compaction of a full ACID table is skipped under the following conditions:

- Another compaction is either running or already initiated for the target.
- The compaction target table is already dropped.
- Table is explicitly configured to be skipped by the auto-compaction

Compaction of an insert-only, ACID table is skipped if hive.compactor.compact.insert.only is set to false (turned off). The default is true. Although you can disable automatic compaction, tables still can be compacted if you explicitly request compaction.Disabling automatic compaction does not prevent you from performing manual compaction.

The compaction auto-initiator can be disabled on service instance level (disabled by default). You can independently enable or disable compaction workers on service instance level. Compaction merges only the bucket files with the same index and keeps the same number of bucket files in base. Compaction does not rebalance the rows between the buckets.

Procedure

At the Hive JDBC client prompt, in the database of the target table, alter the TBLPROPERTIES.

```
ALTER TABLE my_t SET TBLPROPERTIES ('NO_AUTO_COMPACTION'='true');
```

Configuring compaction using table properties

You see how to configure compaction using table properties and learn about the advantage of using this method of configuration.

About this task

You can configure compaction using table properties. Using table properties, you can group all table and partition compactions into a specific queue to match your use case. You can also size the compactor job based on the tables parameters like size and compression.

Procedure

Set table properties to adjust the compaction initiator properties.

```
ALTER TABLE mydb.mytable
SET TBLPROPERTIES (
'compactorthreshold.hive.compactor.delta.pct.threshold'='0.2f',
'compactorthreshold.hive.compactor.delta.num.threshold'='20');
```

These properties change thresholds, as the names imply: the deltas/base size percentage override threshold and the number of deltas threshold.

Configuring the compaction check interval

You need to know when and how to control the compaction process checking, performed in the background, of the file system for changes that require compaction.

When you turn on the compaction initiator, consider setting the hive.compactor.check.interval property. This property determines how often the initiator should search for possible tables, partitions, or compaction. By default, the value for this property is set to 300 seconds. Decreasing hive.compactor.check.interval has the following effect:

• Reduces the time it takes for compaction to be started for a table or partition that requires compaction.
• Requires several calls to the file system for each table or partition that has undergone a transaction since the last major compaction, resulting in increases to the load on the filesystem.

The compaction initiator first checks the completed transactions (COMPLETED_TXN_COMPONENTS), excluding those that already have completed compactions, searching for potential compaction targets. The search of the first iteration includes all transactions. Further searching of iterations are limited to the time-frame since the last iteration.

To configure the compaction check interval, set the hive.compactor.check.interval. For example:

From the Cloudera Data Warehouse service, go to the corresponding Database Catalog CONFIGURATIONS Metastore and set the value for the hive.compactor.check.interval property under the hive-site configuration file.

Query vectorization

You can use vectorization to improve instruction pipelines for certain data and queries and to optimize how Hive uses the cache. Vectorization processes batches of primitive types on the entire column rather than one row at a time.

Unsupported functionality on vectorized data

Some functionality is not supported on vectorized data:

• DDL queries
• DML queries other than single table, read-only queries
• Formats other than Optimized Row Columnar (ORC)

Supported functionality on vectorized data

The following functionality is supported on vectorized data:

• Single table, read-only queries

  Selecting, filtering, and grouping data is supported.
• Partitioned tables
• The following expressions:

  • Comparison: >, >=, <, <=, =, !=
  • Arithmetic plus, minus, multiply, divide, and modulo
  • Logical AND and OR
  • Aggregates sum, avg, count, min, and max

Supported data types

You can query data of the following types using vectorized queries:

• tinyint
• smallint
• int
• bigint

- date
- boolean
- float
- double
- timestamp
- stringchar
- varchar
- binary

Vectorization default

Vectorized query execution can affect performance. You need to be aware of the Boolean default value of hive.vectorized.execution.enabled.

Vectorized query execution is enabled by default (true). Vectorized query execution processes Hive data in batch, channeling a large number of rows of data into columns, foregoing intermediate results. This technique is more efficient than the MapReduce execution process that stores temporary files.