Data Access 3

Managing Apache Hive

Date of Publish: 2018-07-12

http://docs.hortonworks.com

Contents

ACID operations 3

Configure partitions for transactions 3
View transactions 3
View transaction locks 4

Data compaction 5

Initiate compaction 5
View compaction progress 5
Disable compaction 6

Query vectorization 6

Enable query vectorization 7
Check query execution 7

ACID operations

In Hive 3, you can perform ACID (atomicity, consistency, isolation, and durability) v2 transactions at the row level without any configuration. By default, managed tables are ACID tables. You cannot disable transactions.

HDP 3.0 and Hive 3 introduce mature versions of ACID transaction processing and low latency analytical processing (LLAP). ACID tables have been enhanced to the point of now serving as the default table type in HDP 3.0, without performance or operational overload. Application development and operations are simplified with stronger transactional guarantees and simpler semantics for SQL commands. You do not need to bucket ACID v2 tables, so maintenance is easier. With improvements in transactional semantics comes advanced optimizations, such as materialized view rewrites and automatic query cache. 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 despite a machine or system 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.

Configure partitions for transactions

To include INSERT, UPDATE, and DELETE statements in your transaction applications, you must set your dynamic partition mode to nonstrict.

Before you begin

Default limitations in the following parameters are changed to meet your needs:

- hive.exec.max.dynamic.partitions
- hive.exec.max.dynamic.partitions.pernode

Procedure

1. Navigate to hive-site.xml, and open it for editing.
2. Changehive.exec.dynamic.partition.mode to nonstrict.

Related Information

Hive Configuration Properties documentation on the Apache wiki

View transactions

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 where transaction was initiated

View transaction locks

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

About this task

Hive transactions, enabled by default, disables Zookeeper locking. DbLockManager stores and manages all transaction lock information in the Hive Metastore. Heartbeats are sent regularly from lock holders and transaction initiators to the Hive metastore to prevent stale locks and transactions. The lock or transaction is aborted if the metastore does not receive a heartbeat within the amount of time specified by the hive.txn.timeout configuration property.

Before you begin

You ensured that transactions are enabled (the default).

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;

   The following information appears in the output unless ZooKeeper or in-memory lock managers are used.

   - 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 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

To prevent NameNode capacity problems, as administrator, you need to manage compaction of delta files that accumulate during data ingestion.

Hive stores data in base files that cannot be updated by HDFS. Instead, Hive creates a set of delta files for each transaction that alters a table or partition and stores them in a separate delta directory. By default, Hive automatically compacts delta and base files at regular intervals. Compaction is a consolidation of files. You can configure automatic compactions, as well as perform manual compactions of base and delta files. Hive performs all compactions in the background without affecting concurrent reads and writes.

The compactor initiator should run on only one HMS instance.

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.

Transactional tables you created in an earlier version require a major compaction before upgrading to Hive 3.

Related Information

Apache Wiki transactions and compaction documentation

Initiate compaction

You manually start a compaction when automated compaction fails for some reason to perform housekeeping of files as needed.

About this task

Carefully consider the need for a major compaction as this process can consume significant system resources and take a long time. Start a major compaction during periods of low traffic. Starting a compaction queues requests that compact base and delta files for a table or partition. You use the following syntax to issue a query that starts compaction:

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

Procedure

Execute a query to start a major compaction of a table.
ALTER TABLE mytable COMPACT 'compaction_type'

ALTER TABLE compacts tables even if the NO_AUTO_COMPACTION table property is set.

View compaction progress

You view the progress of compactions by running a Hive query.

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
- Hadoop job ID - ID of the submitted Hadoop job

Disable compaction

You can disable automatic compaction of a Apache Hive table by setting a parameter.

About this task

Disabling automatic compaction does not prevent you from performing manual compaction.

Procedure

1. Navigate to the location of hive-site.xml and open the file in a text editor.
2. Set the NO_AUTO_COMPACTION to true.

Query vectorization

You can use vectorization to improve instruction pipelines and cache use. Vectorization enables certain data and queries to process batches of primitive types on 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

Enable query vectorization

To enable query vectorization, as Administrator you change hive-site.xml.

Procedure

1. Navigate to the location of hive-site.xml and open the file in a text editor.
2. Enable query vectorization by enabling vectorized execution in hive-site.xml.
   hive.vectorized.execution.enabled=true

   After you issue a query, Hive examines the query and the data to determine whether vectorization can occur. If not, Hive executes the query with vectorization disabled.

Check query execution

You can determine if query vectorization occurred during execution by running the EXPLAIN VECTORIZATION query statement.

Procedure

1. Start Hive from Beeline.

   ```
   $ hive
   ```

2. Set hive.explain.user to false to see vector values.

   ```
   SET hive.explain.user=false;
   ```

3. Run the EXPLAIN VECTORIZATION statement on the query you want CDP to process using vectorization.

   ```
   EXPLAIN VECTORIZATION SELECT COUNT(*) FROM employees where emp_no>10;
   ```

The following output confirms that vectorization occurs:

```
+----------------------------------------------------+
|                      Explain                       |
+----------------------------------------------------+
| Plan optimized by CBO.                             |
|                                                    |
| Vertex dependency in root stage                    |
| Reducer 2 <-Map 1 [CUSTOM_SIMPLE_EDGE] *vectorized*|
|       File Output Operator [FS_14]                 |
|         Group By Operator [GBY_13] (rows=1 width=12) |
|           Output:["_col0"],aggregations:["count(VALUE._col0)"] |
|         <-Map 1 [CUSTOM_SIMPLE_EDGE] vectorized    |
|           PARTITION_ONLY_SHUFFLE [RS_12]           |
|             Group By Operator [GBY_11] (rows=1 width=12) |
|               Output:["_col0"],aggregations:["count()"] |
|               Select Operator [SEL_10] (rows=1 width=4) |
|                 Filter Operator [FIL_9] (rows=1 width=4) |
|                   predicate:(emp_no > 10)          |
|                   TableScan [TS_0] (rows=1 width=4) |
|
 default@employees,employees,Tbl:COMPLETE,Col:NONE,Output:["emp_no"] |
|                                                    |
+----------------------------------------------------+
23 rows selected (1.542 seconds)
```