

Cloudera Runtime 7.2.11

Managing Apache Hive

Date published: 2019-08-21

Date modified: 2022-08-10

CLOUdera

<https://docs.cloudera.com/>

Legal Notice

© Cloudera Inc. 2024. All rights reserved.

The documentation is and contains Cloudera proprietary information protected by copyright and other intellectual property rights. No license under copyright or any other intellectual property right is granted herein.

Unless otherwise noted, scripts and sample code are licensed under the Apache License, Version 2.0.

Copyright information for Cloudera software may be found within the documentation accompanying each component in a particular release.

Cloudera software includes software from various open source or other third party projects, and may be released under the Apache Software License 2.0 (“ASLv2”), the Affero General Public License version 3 (AGPLv3), or other license terms. Other software included may be released under the terms of alternative open source licenses. Please review the license and notice files accompanying the software for additional licensing information.

Please visit the Cloudera software product page for more information on Cloudera software. For more information on Cloudera support services, please visit either the Support or Sales page. Feel free to contact us directly to discuss your specific needs.

Cloudera reserves the right to change any products at any time, and without notice. Cloudera assumes no responsibility nor liability arising from the use of products, except as expressly agreed to in writing by Cloudera.

Cloudera, Cloudera Altus, HUE, Impala, Cloudera Impala, and other Cloudera marks are registered or unregistered trademarks in the United States and other countries. All other trademarks are the property of their respective owners.

Disclaimer: EXCEPT AS EXPRESSLY PROVIDED IN A WRITTEN AGREEMENT WITH CLOUDERA, CLOUDERA DOES NOT MAKE NOR GIVE ANY REPRESENTATION, WARRANTY, NOR COVENANT OF ANY KIND, WHETHER EXPRESS OR IMPLIED, IN CONNECTION WITH CLOUDERA TECHNOLOGY OR RELATED SUPPORT PROVIDED IN CONNECTION THEREWITH. CLOUDERA DOES NOT WARRANT THAT CLOUDERA PRODUCTS NOR SOFTWARE WILL OPERATE UNINTERRUPTED NOR THAT IT WILL BE FREE FROM DEFECTS NOR ERRORS, THAT IT WILL PROTECT YOUR DATA FROM LOSS, CORRUPTION NOR UNAVAILABILITY, NOR THAT IT WILL MEET ALL OF CUSTOMER’S BUSINESS REQUIREMENTS. WITHOUT LIMITING THE FOREGOING, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, CLOUDERA EXPRESSLY DISCLAIMS ANY AND ALL IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, QUALITY, NON-INFRINGEMENT, TITLE, AND FITNESS FOR A PARTICULAR PURPOSE AND ANY REPRESENTATION, WARRANTY, OR COVENANT BASED ON COURSE OF DEALING OR USAGE IN TRADE.

Contents

ACID operations in Cloudera Data Warehouse.....	4
Options to monitor transactions.....	4
Options to monitor transaction locks.....	5
Data compaction.....	7
Compaction prerequisites.....	8
Compaction tasks.....	8
Starting compaction manually.....	9
Options to monitor compactions.....	9
Disabling automatic compaction.....	11
Configuring compaction using table properties.....	12
Configuring the compaction check interval.....	12
Query vectorization.....	12
Vectorization default.....	13
Query vectorization properties.....	13
Checking query execution.....	14
Tracking Hive on Tez query execution.....	15
Tracking an Apache Hive query in YARN.....	17
Application not running message.....	18

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.

Related Information

[Understanding CREATE TABLE behavior](#)

Options to monitor transactions

As a Hive administrator, you can view the list of all currently open and aborted transactions using the SHOW TRANSACTIONS statement or by querying the TRANSACTIONS view within the SYS database.

The query statements display the following details about transactions:

- TXN_ID: Unique internal transaction ID
- STATE: Transaction state
- STARTED: Timestamp when the transaction was started
- LAST_HEARTBEAT: Timestamp of the latest heartbeat
- USER: Hive user who initiated the transaction
- HOST: Host machine or virtual machine where the transaction was initiated
- HEARTBEAT_COUNT: Total number of heartbeats
- TYPE: Transaction type
 - DEFAULT
 - REPL_CREATED
 - READ_ONLY
 - COMPACTION
- TC_DATABASE: Hive database name
- TC_TABLE: Table name
- TC_PARTITION: Partition name (if the table is partitioned)
- TC_OPERATION_TYPE:
 - SELECT
 - INSERT
 - UPDATE
 - COMPACT
- TC_WRITEID: Unique internal write ID

The following sections describe the various options that you can use to monitor transactions.

SHOW TRANSACTIONS

You can run the SHOW TRANSACTIONS statement to view details about all open and aborted transactions.

SHOW TRANSACTIONS;

This statement lists all the transactions and you cannot filter or limit the results as required. Alternatively, you can use the SYS database to query and filter specific transactions.

Querying the SYS database

You can query the TRANSACTIONS view within the SYS database to filter and view specific transactions.

For example, you can run the following query to view transactions in a particular state:

```
SELECT *
FROM SYS.TRANSACTIONS
WHERE STATE='aborted';
```



Important: Currently, you cannot retrieve information about old or aborted transactions from the SYS database that is preventing the Cleanup phase. This is because the MIN_HISTORY_LEVEL table is not exposed in the SYS database. As a workaround, you can run the following query directly on the backend database:

```
SELECT * from TXNS where txn_id = (
  SELECT MIN(res.id) FROM (
    SELECT ntxn_next id from next_txn_id
    UNION ALL
    SELECT MIN(mhl_min_open_txn_id) id FROM min_history_level
    UNION ALL
    SELECT MIN(txn_id) id FROM txns WHERE txn_state = 'a'
  ) res);
```



Note: Similarly, you can also query the INFORMATION_SCHEMA database for details about transactions. You must use the Ranger service and set up access policies for Hive users on this database to make it accessible.

Options to monitor transaction locks

As a Hive administrator, you can view information about locks on a table, partition, or schema that are created as a result of transactions. You can either use the SHOW LOCKS statement or query the LOCKS view within the SYS database to view transaction locks.

Hive transactions, enabled by default, disable 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.

The query statements display the following details about transaction locks unless ZooKeeper or in-memory lock managers are used:

- LOCK_EXT_ID: Unique internal ID of a lock request that might be requesting multiple lock entries on several resources (tables or partitions).
- LOCK_INT_ID: Unique internal ID of a lock entry that has been requested by a LOCK_EXT_ID
- TXNID: Transaction ID associated with the lock, if one exists
- DB: Hive database name
- TABLE: Table name

- **PARTITION:** Partition name (if the table is partitioned)
- **LOCK_STATE:**
 - **acquired:** transaction initiator holds 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:** lock cannot be shared. No one else can hold the lock at the same time.
 - **shared_read:** any number of other shared_read locks can lock the same resource at the same time
 - **shared_write:** any number of shared_read locks can lock the same resource at the same time, but no other shared_write locks are allowed
- **LAST_HEARTBEAT:** Last time the holder of this lock sent a heartbeat
- **ACQUIRED_AT:** Time when the lock was acquired, if it has been acquired
- **USER:** Hive user who requested the lock
- **HOST:** Host machine or virtual machine on which the Hive user is running a Hive client
- **HEARTBEAT_COUNT:** Total number of heartbeats
- **BLOCKEDBY_EXT_ID:** ID of the lock (LOCK_EXT_ID) causing current lock to be in “waiting” mode, if the lock is in this mode
- **BLOCKEDBY_INT_ID:** ID of the lock (LOCK_INT_ID) causing current lock to be in “waiting” mode, if the lock is in this mode

The following sections describe the various options that you can use to monitor transaction locks.

SHOW LOCKS

You can run the **SHOW LOCKS** statement to view details about all transaction locks. Ensure that transactions are enabled

SHOW LOCKS;

The following examples illustrate some sample queries that you can run:

Query to check table locks

```
SHOW LOCKS mytable EXTENDED;
```

Query to check partition locks

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

Query to check schema locks

```
SHOW LOCKS SCHEMA mydatabase;
```

The **SHOW LOCKS SCHEMA** cannot be used with ZooKeeper or in-memory lock managers.

The **SHOW LOCKS** statement lists all the transaction locks and you cannot filter or limit the results as required. Alternatively, you can use the **SYS** database to query and filter specific locks.

Querying the SYS database

You can query the **LOCKS** view within the **SYS** database to filter and view specific locks.

The following examples illustrate how you can run queries on the **SYS.LOCKS** view to monitor transaction locks:

Query to view locks requested on a particular resource (table or partition)

```
SELECT *
FROM SYS.LOCKS
WHERE db='default'
AND table='tab_acid';
```

Query to view list of acquired locks

```
SELECT *
FROM SYS.LOCKS
WHERE lock_state='acquired';
```

Query to view blocking transactions that are preventing locks requested by a user- defined transaction, from being acquired

```
SELECT *
FROM SYS.TRANSACTIONS
WHERE txn_id IN (
  SELECT txnid
  FROM SYS.LOCKS
  INNER JOIN (
    SELECT blockedby_ext_id, blockedby_int_id
    FROM SYS.LOCKS
    WHERE txnid=4534) b
  ON lock_ext_id = b.blockedby_ext_id
  AND lock_int_id = b.blockedby_int_id
);
```



Note: Similarly, you can also query the INFORMATION_SCHEMA database for details about transaction locks. You must use the Ranger service and set up access policies for Hive users on this database to make it accessible.

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.

Related Information

[Apache Wiki transactions and compaction documentation](#)

Compaction prerequisites

To prevent data loss or an unsuccessful compaction, you must meet the prerequisites before compaction occurs.

Exclude compaction users from Ranger policies

Compaction causes data loss if Apache Ranger policies for masking or row filtering are enabled and the user hive or any other compaction user is included in the Ranger policies.

1. Set up Ranger masking or row filtering policies to exclude the user hive from the policies.

The user (named hive) appears in the Users list in the Ranger Admin UI.

Policy ID	Policy Name	Policy Labels	Status	Audit Logging	Roles	Groups	Users
7	all - hiveservice	--	Enabled	Enabled	--	--	hive rangerlookup impala

2. Identify any other compaction users from the masking or row filtering policies for tables as follows:
 - If the `hive.compaction.run.as.user` property is configured, the user runs compaction.
 - If a user is configured as owner of the directory on which the compaction will run, the user runs compaction.
 - If a user is configured as the table owner, the user runs compaction
3. Exclude compaction users from the masking or row filtering policies for tables.

Failure to perform these critical steps can cause data loss. For example, if a compaction user is included in an enabled Ranger masking policy, the user sees only the masked data, just like other users who are subject to the Ranger masking policy. The unmasked data is overwritten during compaction, which leads to data loss of unmasked content as only the underlying tables will contain masked data. Similarly, if Ranger row filtering is enabled, you do not see, or have access to, the filtered rows, and data is lost after compaction from the underlying tables.

The worker process executes queries to perform compaction, making Hive data subject to data loss ([HIVE-27643](#)). MapReduce-based compactations are not subject to the data loss described above as these compactations directly use the MapReduce framework.

Related Information

[Row-level filtering and column masking in Hive with Ranger policies](#)

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

- Tables or partitions you are compacting must be full ACID or insert-only ACID tables.
- Compaction must be enabled (initiator `hive.compactor.initiator.on=true`)

Procedure

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;
```

Options to monitor compactions

You can view the progress of compactions using the SHOW COMPACTIONS statement or by querying the COMP_ACTIONS view within the SYS database.

The query statement displays the following details about compactions:

- C_ID: Unique internal compaction ID
- C_DATABASE: Hive database name
- C_TABLE: Table name
- C_PARTITION: Partition name (if the table is partitioned)
- C_TYPE: Major or minor compaction

- **C_STATE:** Compaction state
 - initiated: waiting in queue to be compacted
 - working: currently being compacted
 - ready for cleaning: compaction completed and old files scheduled for removal
 - failed: compaction job failed. Details are printed to the metastore log.
 - succeeded: compaction job completed successfully
 - attempted: initiator attempted to schedule a compaction but failed or `hive.compactor.initiator.failed.compacts.threshold` is breached and compaction is skipped. Details are printed to the metastore log.
- **C_WORKER_HOST:** Thread ID of the worker thread performing the compaction
- **C_WORKER_ID:** Unique internal worker ID
- **C_WORKER_VERSION:** Version of distribution with running worker
- **C_ENQUEUE_TIME:** Time when compaction request was placed
- **C_START:** Start time of the compaction job
- **C_DURATION:** Duration (in ms) of the compaction job
- **C_HADOOP_JOB_ID:** ID of the submitted MapReduce job for MR compaction. 'None' for query-based compaction.
- **C_INITIATOR_HOST:** Host where initiator is configured
- **C_INITIATOR_ID:** Unique internal initiator ID
- **C_INITIATOR_VERSION:** Version of distribution with active initiator



Note: Details about "initiators" and "workers" are new additions that you can use to investigate the compaction setup in an environment. If you are unable to retrieve these details using the options listed here, you can run the following query on the backend database:

```
SELECT activity.pid,
       activity.username,
       activity.client_addr,
       activity.client_hostname,
       activity.query,
       blocking.pid AS blocking_pid,
       blocking.query AS blocking_query,
       blocking.client_addr AS blocking_client_addr,
       blocking.client_hostname AS blocking_client_hostname
FROM pg_stat_activity AS activity
JOIN pg_stat_activity AS blocking
  ON blocking.pid = ANY(pg_blocking_pids(activity.pid))
WHERE blocking.query like '%AUX_TABLE%';
```

The following sections describe the various options that you can use to monitor compactions.

SHOW COMPACTIONS

You can run the **SHOW COMPACTIONS** statement to view details about all the compaction jobs.

SHOW COMPACTIONS;

Authorization is not required to use the **SHOW COMPACTIONS** statement. Therefore, every user can view compactions and their current state.

Since the statement lists all the compaction jobs, you cannot filter or limit the results as required. Alternatively, you can use the **SYS** database to query and filter specific compactions.

Querying the SYS database

The **SYS** database in the Hive metastore contains the following sources that you can use to monitor compactions:

- **COMPACTION_QUEUE:** Stores information about compaction requests - both explicit user submitted and compactions discovered by the initiator and enqueued.

- **COMPLETED_COMPACTIONS**: Stores the same information present in **COMPACTION_QUEUE** along with the compaction's end time. Records are moved to the completed table when the cleaner finishes the job on the compaction request.
- **COMPACTIONS**: A view over the **COMPACTION_QUEUE** and **COMPLETED_COMPACTIONS** tables that resolves the column values to human readable format.

The **SYS** database tables and views are treated as normal Hive external tables or views, and therefore, standard access policies can be configured against these sources.

The following examples illustrate how you can run queries on the **SYS.COMPACTIONS** view to monitor compactions:

Query to display the last 10 failed compactions

```
SELECT *
FROM SYS.COMPACTIONS
WHERE C_STATE='failed'
ORDERBY C_ID DESC
LIMIT 10;
```

Query to check the status of a specific compaction using the compaction ID

```
SELECT *
FROM SYS.COMPACTIONS
WHERE C_ID='1234';
```

Query to view the total number of compactions in a particular state

```
SELECT COUNT(*)
FROM SYS.COMPACTIONS
WHERE C_STATE='ready for cleaning';
```



Note: Similarly, you can also query the **INFORMATION_SCHEMA** database for details about compactions. You must use the Ranger service and set up access policies for Hive users on this database to make it accessible.

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 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.

Query vectorization properties

You can manage query vectorization by setting properties in Cloudera Manager. The names of each property and its description helps set up vectorization.

Vectorization properties

hive.vectorized.groupby.checkinterval

In vectorized group-by, the number of row entries added to the hash table before re-checking average variable size for memory usage estimation.

hive.vectorized.groupby.flush.percent

Ratio between 0.0 and 1.0 of entries in the vectorized group-by aggregation hash that is flushed when the memory threshold is exceeded.

hive.vectorized.execution.enabled

Enable optimization that vectorizes query execution by streamlining operations by processing a block of 1024 rows at a time.

hive.vectorized.execution.reduce.enabled

Whether to vectorize the reduce side of query execution.

hive.vectorized.use.vectorized.input.format

If enabled, Hive uses the native vectorized input format for vectorized query execution when it is available.

hive.vectorized.use.checked.expressions

To enhance performance, vectorized expressions operate using wide data types like long and double. When wide data types are used, numeric overflows can occur during expression evaluation in a different manner for vectorized expressions than they do for non-vectorized expressions. Consequently, different query results can be returned for vectorized expressions compared to results returned for non-vectorized expressions. When this configuration is enabled, Hive uses vectorized expressions that handle numeric overflows in the same way as non-vectorized expressions are handled.

hive.vectorized.adaptor.usage.mode

Vectorized Adaptor Usage Mode specifies the extent to which the vectorization engine tries to vectorize UDFs that do not have native vectorized versions available. Selecting the "none" option specifies that only queries using native vectorized UDFs are vectorized. Selecting the "chosen" option specifies that Hive chooses to vectorize a subset of the UDFs based on performance benefits using the Vectorized Adaptor. Selecting the "all" option specifies that the Vectorized Adaptor be used for all UDFs even when native vectorized versions are not available.

hive.vectorized.use.vector.serde.deserialize

If enabled, Hive uses built-in vector SerDes to process text and sequencefile tables for vectorized query execution.

hive.vectorized.input.format.excludes

Specifies a list of file input format classnames to exclude from vectorized query execution using the vectorized input format. Note that vectorized execution can still occur for an excluded input format based on whether row SerDes or vector SerDes are enabled.

Checking 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,Ou |
| tput:["emp_no"] |
+-----+
23 rows selected (1.542 seconds)
```

Tracking Hive on Tez query execution

You need to know how to monitor Hive on Tez queries during execution. Several tools provide query details, such as execution time.

About this task

You can retrieve local fetch details about queries from HiveServer (HS2) logs, assuming you enable a fetch task. You configure the following properties:

hive.fetch.task.conversion

Value: minimal

Some select queries can be converted to a single FETCH task instead of a MapReduce task, minimizing latency. A value of none disables all conversion, minimal converts simple queries such as SELECT * and filter on partition columns, and more converts SELECT queries including FILTERS.

hive.fetch.task.conversion.threshold

Value: 1 GiB

Above this size, queries are converted to fetch tasks.

Increasing the static pool does not speed reads and there is not recommended.

Procedure

1. In Cloudera Manager, click Clusters Hive on Tez Configuration , and search for fetch.

2. Accept, or change, the default values of the fetch task properties.

Fetch Task Query Conversion hive.fetch.task.conversion	HiveServer2 Default Group ⓘ <input type="radio"/> none <input type="radio"/> minimal <input checked="" type="radio"/> more
Fetch Task Query Conversion Threshold hive.fetch.task.conversion.threshold	HiveServer2 Default Group ⓘ <input type="text" value="1"/> <input type="button" value="GiB"/> ▼

3. Navigate to the HiveServer log directory and look at the log files.

In Cloudera Manager, you can find the location of this directory as the value of HiveServer2 Log Directory.

HiveServer2 Log Directory	HiveServer2 Default Group ⓘ <input type="text" value="/var/log/hive"/> Directory where HiveServer2 will place its log files.
----------------------------------	---

4. In Cloudera Manager, click **Clusters** **Hive on Tez Configuration** , and click to the **HiveServer Web UI**.

HIVE_ON_TEZ-1 Actions ▾ Sep 13, 10:43 PM

[Status](#) [Instances](#) [Configuration](#) [Commands](#) [Charts Library](#) [Audits](#) [HiveServer2 Web UI](#) [Quick Links ▾](#)

HiveServer2

Active Sessions

User Name	IP Address	Operation Count	Active Time (s)	Idle Time (s)
admin	172.27.72.74	0	58523	58466

Total number of sessions: 1

Open Queries

User Name	Query	Execution Engine	State	Opened Timestamp	Opened (s)	Latency (s)	Drilldown Link
Total number of queries: 0							

Last Max 25 Closed Queries

User Name	Query
admin	INSERT INTO TABLE sample_07

5. Use Hue to track query progress.

Tracking an Apache Hive query in YARN

You need to know how to monitor Apache Hive queries in YARN. Using information from the output of query execution in Beeline, you can accomplish this task.

About this task

Procedure

1. Run a query in Beeline.

```
0: jdbc:hive2://ip-10-10-10-10.cloudera.site:> select * from depts;
INFO : Compiling command(queryId=hive_1599978188515_5723b397-c375-48c2-ab38-7be298603de9): select * from depts
...
```

2. Scroll down the output until you find the INFO containing the YARN App id.

```
...
INFO : Status: Running (Executing on YARN cluster with App id application_1599978188515_0010)
```

VERTICES			MODE	STATUS	TOTAL	COMPLETED	RUNNING	PE
NDING	FAILED	KILLED						
...								

3. In Cloudera Manager, click Clusters Yarn Instances Web UI .

YARN-1

Actions

Status

Instances

Configuration

Commands

Applications

Charts Library

Audits

Web UI

Quick Links

ResourceManager Web UI (nightly7x-unsecure-3)

HistoryServer Web UI (nightly7x-unsecure-3)

4. Click Resource Web UI Applications

Application ID	Application Type	Application Tag	Application Name	User	State	Queue
application_1599978188515_0010	TEZ	N/A	HIVE-dcb03c1e-...	hive	Running	default
application_1599978188515_0009	SPARK	N/A	PySparkShell	hdfs	Finished	default

5. Find the match for the App id and gather information you want.

Application not running message

Understanding the Application not running message from TEZ that appears in the YARN application log prevents confusion when inspecting Hive queries.

During startup, HiveServer starts sessions that are available until tez.session.am.dag.submit.timeout.secs expires, and then the Application Masters are killed. HiveServer transparently restarts dead AMs on demand when you run another query. HiveServer tries to clean up sessions from the Tez pool during shutdown. "Application not running" message in a stack trace logging is not an issue. This message is just a trace logged when the session is closed or restarted and the AM has timed out.