Cloudera Runtime 1.0.0

Managing Apache Impala

Date published: 2020-11-30
Date modified: 2024-05-30

https://docs.cloudera.com/

Legal Notice

Contents

READ Support for FULL ACID ORC Tables

FULL ACID v2 transactional tables are readable in Impala without modifying any configurations. You must have connection to Hive Metastore server in order to READ from FULL ACID tables.

There are two types of transactional tables available with Hive ACID.

- INSERT-ONLY
- FULL ACID

Until this release, Impala in CDP supported INSERT-ONLY transactional tables allowing both READ and WRITE operations. The latest version of Impala in CDP now also supports READ of FULL ACID ORC tables.

By default tables created in Impala are INSERT-ONLY managed tables whereas the default tables in Hive are managed tables that are FULL-ACID and INSERT-ONLY.

Limitations

- Impala cannot CREATE or WRITE to FULL ACID transactional tables yet. You can CREATE and WRITE FULL ACID transactional tables with transaction scope at the row level via HIVE and use Impala to READ these tables.
- Impala does not support ACID v1.

Concepts Used in FULL ACID v2 Tables

Before beginning to use FULL ACID v2 tables you must be aware of these new concepts like transactions, WriteIds, rowIDs, delta delete directories, locks, etc. that are added to FULL ACID tables to achieve ACID semantics.

Write IDs

For every transaction, both read and write, Hive will assign a globally unique ID. For transactional writes like INSERT and DELETE, it will also assign a table-wise unique ID, a write ID. The write ID range will be encoded in the delta and delete directory names. Results of a DML transactional query are allocated to a location under partition/table. This location is derived by Write ID allocated to the transaction. This provides Isolation of DML queries and such queries can run in parallel without interfering with each other.

New Sub-directories

New data files resulting from a DML query are written to a unique location derived from WriteId of the transaction. You can find the results of an INSERT query in delta directories under partition/table location. Depending on the operation type there can be two types of delta directories:

- Delta Directory: This type is created for the results of INSERT statements and is named delta_<writeId>_<writeId> under partition/table location.
- Delete Delta Directory: This delta directory is created for results of DELETE statements and is named delete_delta_<writeId>_<writeId> under partition/table location.

UPDATE operations create both delete and delta directories.

Row IDs

rowId is the auto-generated unique ID within the transaction and bucket. This is added to each row to identify each row in a table. RowID is used during a DELETE operation. When a record is deleted from a table, the rowId of the deleted row will be written to the delete_delta directory. So for all subsequent READ operations all rows will be read except these rows.

Schematic differences between INSERT-ONLY and FULL ACID tables

INSERT-ONLY tables do not have a special schema. They store the data just like plain original files from the non-ACID world. However, their files are organized differently. For every INSERT statement the created files are put into a transactional directory which has transactional information in its name.

Full ACID tables do have a special schema. They have row identifiers to support row-level DELETEs. So a row in Full ACID format looks like this:

```
{
   "operation": 0,
   "originalTransaction": 1,
   "bucket": 536870912,
   "rowId": 0,
   "currentTransaction": 1,
   "row": {"i": 1}
   }
```

- The green columns are the hidden/system ACID columns.
- Field “row” holds the user data.
- operation 0 means INSERT, 1 UPDATE, and 2 DELETE. UPDATE will not appear because of the split-update technique (INSERT + DELETE).
- originalTransaction is the write ID of the INSERT operation that created this row.
- bucket is a 32-bit integer defined by BucketCodec class.
- rowId is the auto-generated unique ID within the transaction and bucket.
- currentTransaction is the current write ID. For INSERT, it is the same as currentTransaction. For DELETE, it is the write ID when this record is first created.
- row contains the actual data. For DELETE, row will be null.

Key Differences between INSERT-ONLY and FULL ACID Tables

Before beginning to use FULL ACID v2 tables you must be aware of the key differences between the INSERT-ONLY and FULL-ACID tables.

This table highlights some of the differences between the INSERT-ONLY and FULL ACID tables.

	INSERT-ONLY	FULL ACID
Schema	There is no special data schema. They store the data just like plain original files from the non-ACID world.	Data is in special format, i.e. there are synthetic columns with transactional information in addition to actual data.
Transactional information	Transactional information is encoded in directory names.	Full ACID tables also use the same directory structure as INSERT-only tables. Transactional information is encoded in the directory names. Directory name and filename are the source of transactional information.
Table properties	'transactional'='true', 'transactional_properties'='insert_only'	'transactional'='true'
Supported operations	INSERT-ONLY tables only support insertion of data. UPDATES and DELETES are not supported. These tables also provide CREATE TABLE, DROP TABLE, TRUNCATE, INSERT, SELECT operations.	FULL ACID ORC tables can be READ using IMPALA. These tables also provide UPDATE and DELETE operations at the row level using HIVE. This is achieved using transactions like Insert-Only Tables along with changes in ORC Reader to support deletes.
WRITE operation	WRITE operations are atomic and the results of the insert operation are not visible to other query operations until the operation is committed.	WRITE operations are atomic - The operation either succeeds completely or fails; it does not result in partial data.

	INSERT-ONLY	FULL ACID
INSERT operation	For every INSERT statement the created files are added to a transactional directory which has transactional information in its name.	INSERT operation is done through HIVE and this statement is executed in a single transaction. This operation creates a delta directory containing information about this transaction and its data.
DELETE operation	N/A	DELETE operation is done through HIVE and this event creates a special “delete delta” directory.
UPDATE operation	N/A	UPDATE operation is done through HIVE. This operation is split into an INSERT and DELETE operation. This operation creates a delta dir followed by a delete dir.
READ operation	READ operations always read a consistent snapshot of the data.	READ operations always read a consistent snapshot of the data.
Supported file format	Supports any file formats.	Supports only ORC.
Compactions	Minor and major compactions are supported.	Minor compactions can be created, which means several delta and delete directories can be compacted into one delta and delete directory. Major compactions are also supported.

Note: Currently, ALTER TABLE statement is not supported on both insert-only and full acid transactional tables.

File structure of FULL ACID transational table

Hive 3 achieves atomicity and isolation of operations on transactional tables by using techniques in write, read, insert, create, delete, and update operations that involve delta files, which can provide query status information and help you troubleshoot query problems.

Compaction of Data in FULL ACID Transactional Table

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 and stores them in a separate delta directory. When the number of delta and delete directories in the table grow, the read performance will be impacted, since reading is a process of merging the results of valid transactions. To avoid any compromise on the read performance, occasionally Hive performs compaction, namely minor and major. This process merges these directories while preserving the transaction information.

To initiate automatic compaction, you must enable it using Cloudera Manager. For more information on managing the compaction process, see the link provided under Related Information.

Automatic Invalidation/Refresh of Metadata

In this release, you can invalidate or refresh metadata automatically after changes to databases, tables or partitions render metadata stale. You control the synching of tables or database metadata by basing the process on events. You learn how to access metrics and state information about the event processor.

When tools such as Hive and Spark are used to process the raw data ingested into Hive tables, new HMS metadata (database, tables, partitions) and filesystem metadata (new files in existing partitions/tables) are generated. In previous versions of Impala, in order to pick up this new information, Impala users needed to manually issue an INVALIDATE or REFRESH commands.

When automatic invalidate/refresh of metadata is enabled,, the Catalog Server polls Hive Metastore (HMS) notification events at a configurable interval and automatically applies the changes to Impala catalog.

Impala Catalog Server polls and processes the following changes.

- Refreshes the tables when it receives the ALTER TABLE event.
- Refreshes the partition when it receives the ALTER, ADD, or DROP partitions.
- Adds the tables or databases when it receives the CREATE TABLE or CREATE DATABASE events.
- Removes the tables from catalogd when it receives the DROP TABLE or DROP DATABASE events.
- Refreshes the table and partitions when it receives the INSERT events.

  If the table is not loaded at the time of processing the INSERT event, the event processor does not need to refresh the table and skips it.
- Changes the database and updates catalogd when it receives the ALTER DATABASE events. The following changes are supported. This event does not invalidate the tables in the database.
  - Change the database properties
  - Change the comment on the database
  - Change the owner of the database
  - Change the default location of the database

    Changing the default location of the database does not move the tables of that database to the new location. Only the new tables which are created subsequently use the default location of the database in case it is not provided in the create table statement.

This feature is controlled by the ##hms_event_polling_interval_s flag. Start the catalogd with the ##hms_event_polling_interval_s flag set to a positive integer to enable the feature and set the polling frequency in seconds. We recommend the value to be less than 5 seconds.

Limitations

The following use cases are not supported:

- When you bypass HMS and add or remove data into table by adding files directly on the filesystem, HMS does not generate the INSERT event, and the event processor will not invalidate the corresponding table or refresh the corresponding partition.

  It is recommended that you use the LOAD DATA command to do the data load in such cases, so that event processor can act on the events generated by the LOAD command.
- The Spark API that saves data to a specified location does not generate events in HMS, thus is not supported. For example:

```
Seq((1, 2)).toDF("i", "j").write.save("/user/hive/warehouse/spark_etl.db/
customers/date=01012019")
```

- Event processing could have delays due to the polling interval and auto-refresh on large tables also takes time. If you want the metadata to be synced up immediately, manual REFRESH/INVALIDATE is a better choice and has a better guarantee.

Disable Event Based Automatic Metadata Sync

When the ##hms_event_polling_interval_s flag is set to a non-zero value for your catalogd, the event-based automatic invalidation is enabled for all databases and tables. If you wish to have the fine-grained control on which tables or databases need to be synced using events, you can use the impala.disableHmsSync property to disable the event processing at the table or database level.

This feature can be turned off by setting the ##hms_event_polling_interval_s flag set to 0.

When you add the DBPROPERTIES or TBLPROPERTIES with the impala.disableHmsSync key, the HMS event based sync is turned on or off. The value of the impala.disableHmsSync property determines if the event processing needs to be disabled for a particular table or database.

- If 'impala.disableHmsSync'='true', the events for that table or database are ignored and not synced with HMS.
- If 'impala.disableHmsSync'='false' or if impala.disableHmsSync is not set, the automatic sync with HMS is enabled if the ##hms_event_polling_interval_s global flag is set to non-zero.

- To disable the event based HMS sync for a new database, set the impala.disableHmsSync database properties in Hive as currently, Impala does not support setting database properties:

```
CREATE DATABASE <name> WITH DBPROPERTIES ('impala.disableHmsSync'='true');
```

- To enable or disable the event based HMS sync for a table:

```
CREATE TABLE <name> ... TBLPROPERTIES ('impala.disableHmsSync'='true' |
'false');
```

- To change the event based HMS sync at the table level:

```
ALTER TABLE <name> SET TBLPROPERTIES ('impala.disableHmsSync'='true' | '
false');
```

When both table and database level properties are set, the table level property takes precedence. If the table level property is not set, then the database level property is used to evaluate if the event needs to be processed or not.

If the property is changed from true (meaning events are skipped) to false (meaning events are not skipped), you need to issue a manual INVALIDATE METADATA command to reset event processor because it doesn't know how many events have been skipped in the past and cannot know if the object in the event is the latest. In such a case, the status of the event processor changes to NEEDS_INVALIDATE.

Metrics for Event Based Automatic Metadata Sync

You can use the web UI of the catalogd to check the state of the automatic invalidate event processor.

By default, the debug web UI of catalogd is at http://*impala-server-hostname*:25020 (non-secure cluster) or https://*impala-server-hostname*:25020 (secure cluster).

Under the web UI, there are two pages that presents the metrics for HMS event processor that is responsible for the event based automatic metadata sync.

- /metrics#events
- /events

  This provides a detailed view of the metrics of the event processor, including min, max, mean, median, of the durations and rate metrics for all the counters listed on the /metrics#events page.

The /metrics#events page provides the following metrics about the HMS event processor.

Name	Description
events-processor.avg-events-fetch-duration	Average duration to fetch a batch of events and process it.
events-processor.avg-events-process-duration	Average time taken to process a batch of events received from the Metastore.
events-processor.events-received	Total number of the Metastore events received.
events-processor.events-received-15min-rate	Exponentially weighted moving average (EWMA) of number of events received in last 15 min. This rate of events can be used to determine if there are spikes in event processor activity during certain hours of the day.
events-processor.events-received-1min-rate	Exponentially weighted moving average (EWMA) of number of events received in last 1 min. This rate of events can be used to determine if there are spikes in event processor activity during certain hours of the day.
events-processor.events-received-5min-rate	Exponentially weighted moving average (EWMA) of number of events received in last 5 min. This rate of events can be used to determine if there are spikes in event processor activity during certain hours of the day.

Name	Description
events-processor.events-skipped	Total number of the Metastore events skipped. Events can be skipped based on certain flags are table and database level. You can use this metric to make decisions, such as: • If most of the events are being skipped, see if you might just turn off the event processing. • If most of the events are not skipped, see if you need to add flags on certain databases.
events-processor.status	Metastore event processor status to see if there are events being received or not. Possible states are: • PAUSED The event processor is paused because catalog is being reset concurrently. • ACTIVE The event processor is scheduled at a given frequency. • ERROR • The event processor is in error state and event processing has stopped. • NEEDS_INVALIDATE The event processor could not resolve certain events and needs a manual INVALIDATE command to reset the state. • STOPPED The event processing has been shutdown. No events will be processed. • DISABLED The event processor is not configured to run.