Hierarchical metastore event processing

Introduction

At present, metastore event processor is single threaded. Notification events are processed sequentially with a maximum limit of 1000 events fetched and processed in a single batch. Multiple locks are used to address the concurrency issues that may arise when catalog DDL operation processing and metastore event processing tries to access/update the catalog objects concurrently. Waiting for a lock or file metadata loading of a table can slow the event processing and can affect the processing of other events following it. Those events may not be dependent on the previous event. Altogether it takes a very long time to synchronize all the HMS events.

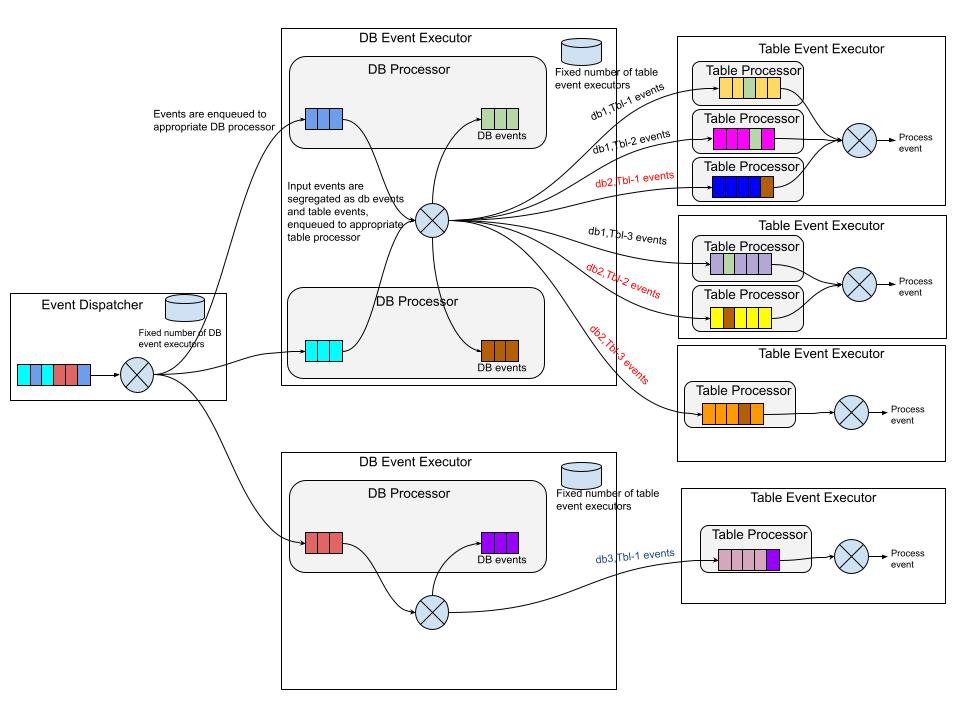
Hierarchical event processing

Existing metastore event processing can be turned into multi-level event processing. Idea is to segregate the events based on their dependency, maintain the order of events as they occur within the dependency and process them independently as much as possible.

1. All the events of a table are processed in the same order they have actually occurred.
2. Events of different tables are processed in parallel.
3. When a database is altered, all the events relating to the database(i.e., for all its tables) occurring after the alter db event are processed only after the alter database event is processed ensuring the order.

Three Layered event processing:

1. Event dispatcher
2. DB Event Executor
3. Table Event Executor



Metastore event processor is responsible for dispatching the metastore events. It maintains the fixed pool of DB event executors. Each DB event executor is responsible for processing of events belonging to a set of databases(with DB processor instances). Metastore event dispatcher fetches the events from HMS in batches, segregates them based on the database name and enqueues the events to appropriate DB processors that are bound to DB event executors.

Functions of Event dispatcher thread:

1. Management of DB event executors:
2. Association of DB processor to a DB event executor

During the event dispatch, if there is no DB processor to handle the event belonging to a database, a new DB processor instance is created and is associated with one of the available DB event executors that has less number of outstanding events to process at the moment(i.e., total number of outstanding events on all its table event executors).

1. Cleanup of DB event executors

After dispatching all the events received in the batch to respective DB processors, unwanted DB processors that do not have any events to process are removed from the respective DB event executors.

1. Certain events require preprocessing at metastore event dispatcher:
2. COMMIT\_TXN and ABORT\_TXN notification events from HMS do not have db name and table name in the events. Event dispatcher need to fetch the table writeids information from the transactionId present in the event and dispatch new events to the appropriate DB processors involved in the transaction.
3. ALTER\_TABLE event for rename can be within the database or across the databases. The dispatcher creates 2 events to the appropriate DB processors so that the old table is removed with one event and the new table is added with another event.

Note:

1. A DB processor is not permanently tied to a particular DB event executor. A DB processor instance is created and associated with one of the available DB event executors when required. And it is cleaned up when not required.

DB event executor is responsible for dispatching table events, processing database events and management of table event executors. Each DB event executor is responsible for multiple databases(i.e., DB processors). It maintains the fixed pool of table event executors. Each table event executor is responsible for processing of events belonging to a set of tables belonging to the databases(i.e., DB processors) of the DB event executor. Functions of DB event executor thread are:

It does the following for each of the DB processor it is responsible:

1. Segregates the events from the input queue as database events and table events.
   1. Table events are enqueued to the appropriate table processor for processing.
   2. Database events are enqueued to another queue on the same DB processor for later processing. They are also enqueued to all the available table processors for the database(i.e., in the DB processor). Database events in the table processor queue just act as a barrier to restrict the processing of table events that occurred after the database event, till the database event is processed on DB processor. It serves as synchronization of table processors for database events.
2. Checks and processes database events that are eligible for processing. A database event is processed only when all the table processors of the DB processor have finished processing events till that database event.
3. Manages the cleanup of table processors belonging to the DB processor. Table processors that do not have any events to process are removed from the DB processor and also from the respective table event executor to reduce the load on table event processing thread.

Note:

1. If there is no table processor to handle the event of a table at the time of table event dispatch, a new instance of a table processor is created and it is associated with one of the available table event executors that has less number of outstanding events to process at the moment.
2. All the tables of the database are mapped to any of available table event executors within the DB event executor. The DB processor will dispatch events of all tables in the database.
3. A Table processor is not permanently tied to a particular table event executor. Table processor instance is created and associated with one of the available table event executors within the DB event executor when required. And it is cleaned up when not required.

Table event executor is responsible for processing of table events. Each table event executor is responsible for multiple tables(i.e., table processors). Functions of table event executor thread are:

It does the following for each of the table processor it is responsible:

1. Applies the table events sequentially ensuring the concurrency and the consistency of the table catalog object.
2. When the event to process is a database event(acts as a barrier), it just marks the event(as reached till this event) and advances to the next table processor for event processing. Further event processing does not happen for the current table processor until that database event is processed on the DB processor, which happens when all the table processors reach till that barrier event.

Note:

1. Trying to acquire locks during a table event processing can affect the processing time of other tables sharing the same table event executor. But, table events processing on other table event executors remain unaffected.

DB processor and table processor apply the events on existing database and table catalog objects respectively.

As further optimization, when a drop database event is encountered at the DB processor, all the prior pending events at the DB processor and its table processors can be ignored as they are not required to be processed anymore. Similarly, when a drop table event is encountered, all the prior pending events at the table processor can be ignored. Thus avoiding processing of undesired events.

Reset Catalogd/Invalidate Metadata:

Upon reset, all the DB processor and table processor queues need to be cleared after pausing the metastore event processor.

Possible catalogd configurations:

1. Number of DB event executors on the metastore event processor.
2. Number of table event executors for a single DB event executor.
3. Maximum number of table events to be processed for a table(i.e., events on table processor) in a single scheduled run.
4. Maximum number of events to be dispatched for a DB processor in a single scheduled run.