Falcon-Hive DR Replication

Preventing Sudden Bee Colony Collapse (aka the Birds & Bees talk?)

Sushanth Sowmyan <khorgath@gmail.com>

Need

- DR Scenarios
- Load balancing between warehouses

Event-based replication

- Replication implemented not as a hive feature, but as a functionality that hive supports, extending the current export-import functionality
- Actual implementation of repl is implemented in a tool that lives in Falcon-space that will effectively call a export-distcp-import loop. This is intended to be pluggable, and tools other than Falcon can perform this task. If need be, someone might also want to write a replication daemon for hive that processes these events.
- First/initial dump(bootstrap) will be done for any new tables/entities that we need to replicate, and then any changes to that table from then on captured by an event log

Why not ...?

- Log-based :
  - If separate redo log, sheer volume, requires constant clear of log to prevent source system from being bogged down. Just maintaining a metadata log is pretty expensive.
  - If in-place edit log (with compaction semantics) maintaining history of all changes, still expensive if user makes many changes and complexities from interop with ACID.

Why not … ?

- Feed-based :
  - This is what is currently in place in Falcon, not flexible enough, requires knowledge of BI details, can have restrictions like type of partitioning key, etc.
  - Can balloon in complexity of config easily.

Why not … ?

- HDFS replication + DB replication?
  - Multiple sources of truth + sync headaches
    - Consistent object availability?
  - No inherent DR-type replication in HDFS (yet)
  - User might want finer granularity of replication desired.

Lazy Delta Log with possibility for Event nullification

- The moment we say we allow for a lazy delta log, i.e. a delta log that's generate after a commit(), it is possible that the object change again before the delta is generated, and we should not block that change on the generation of this log.
- This implies that multiple events might affect the object in question before a delta can be logged, and thus, the delta has to be “smartly” generated and applied.

Lazy Delta Log with possibility for Event nullification

- This, however, also allows for further optimization in terms of batching events, so as to nullify some events depending on future or past events.
  - “ADD PARTITION” on an event can be nullified if a future event drops it.
  - Or, we can nullify a future alter partition after a add partition if we know we'll capture the altered state when we process the add ptn.

Lazy Delta Log with possibility for Event nullification

- Has potential to be smaller in size than redo log if there's churn
- Asynchronous from user-perspective, so will not negatively affect writers
- Does mean that readers can be out of sync with src, so depending on what is used to perform the replication, and what api that provides to the end user to gate application, consistency might be too lenient an “eventual consistency”

Initial dev : Objects Scope

- Will replicate : DB, TABLE, PARTITON
- The scope of this drop for replication is limited to the primary data vehicles of a warehouse - partitions, tables(partitioned and unpartitioned, managed and external) and databases. These will be the only objects being replicated.

Initial dev : Objects Scope

- Won’t replicate : It will not replicate virtual objects like views, and will not handle other metadata objects such as roles, etc, it will be up to the warehouse administrator to manage those aspects currently.
  - These will eventually be added, but as future work

Initial dev : Data Scope

- Will replicate : External table access patterns like writing data to hdfs, then registering partition addition, or table creation (CREATE scenarios)
- If you are generating a new partition that did not previously exist, though, for example, and you copy in data directly through hdfs, then do a "alter table add partition" action through hive, this will generate an event, and we will record and replicate this.

Initial dev : Data Scope

- Will replicate: Hive-based data modifications that currently do not change metadata (Alter, Update scenarios)
- Some operations, such as INSERT INTO/ INSERT OVERWRITE on partitions that already exist, or unpartitioned tables that already exist currently do not generate events - we will add notifications for these so that they are covered.

Initial dev : Data Scope

- Won’t replicate : Direct HDFS writes without Metadata registering (Alter, Update scenarios)
- If a user copies new files in hdfs to a destination location for a partition or table that already exists, and there is no metadata event associated with it, there is no way for us to know that the object has changed, although you will see the changes if you query that through hive using a select.

Initial dev : Additional challenges

- Dynamic partitioning, where some changes might be ptn appends, some might be ptn creates
- Append into partitions in general
- Rollback-safety, maintaining atomicity, or at least, linearizability.

Caveats after initial dev – need to be addressed

- ACID mutual incompatibility
- Table renames
- Only hive native tables
- Alter … SET LOCATION
- Appends through HCatalog

Events

- {CREATE,ALTER,DROP}{DB,TABLE,PTN}
- Except.. No AlterDatabase for now.
- And.. CreateDatabase will be handled as a NoOp
- EventLog stored in the NOTIFICATION_LOG table on metastore db. Another table called NOTIFICATION_SEQUENCE also created.

New Metastore Tables

- NOTIFICATION_LOG
  - EVENT_ID
  - EVENT_TYPE
  - DB_NAME
  - TBL_NAME
  - MESSAGE
- NOTIFICATION_SEQUENCE

Setup Conf properties

- hive.metastore.event.listeners
  - Default : empty . Set this to org.apache.hive.hcatalog.listener.DbNotificationListener if you want the metastore to log events
- hive.metastore.dml.events
  - Default : false . Users should set to true if they have a heavy dml-insert usecase that they need replicated. If you have a more traditional ADD_PTN kind of replication usecase, you can have this off for a lower load – you just won't replicate INSERTs then.

Setup Conf properties

- hive.metastore.event.db.listener.timetolive
  - Default : 86400s – set to higher if we want events to last longer in the db.
- hive.exim.strict.repl.tables
  - Default : true – set to false if you want regular export/import to work on tables which are the destination of replication – not recommended to change, possible that you may have a usecase in a single .q script to override this if you have a need to import into one.

Maintaining state

- DB, Table & Partition objects now have an entry in their parameters list: repl.last.id that denotes how “recent” that object is.
- “0” or “null” denotes no knowledge about the object.
- Bootstrap is required to set initial state on destination.
- Things discovered along the way :
  - There is no DDL to edit a partition property
  - AlterTableDesc is a pain in comparison to the Metastore api for the same

The Basic Idea

- All export dumps are tagged with the “current” event id at the time of the dump
- All imports implement a “import if export dump is newer then the destination” semantic
- Drops implement a “drop if destination is older than the event which triggered this” semantic.
- Alters (not add ptn) are treated as metadata-only exports.

Changes to Hive QL

- EXPORT … [FOR [METADATA] REPLICATION(“comment”)]
- IMPORT … (as normal) – but handles new semantics
- DROP TABLE … FOR REPLICATION('eventid')
- ALTER TABLE ... DROP PARTITION (…) FOR REPLICATION('eventid')

New kinds of export dumps

- Original normal export : Import behaves normally (except for one new error condition)
- Replication export
  - Uses replace semantics unlike normal import
  - Import will import if this dump is an update to the object being affected
- Replication metadata export
  - Import will update metadata only, if update.
- Replication noop export : Ignored by import.

Why export/import?

- Wouldn't it be better to have a MD log, but use the data in-place, rather than exporting and importing?
  - Approach considered as optimization for export, but the basic problem is that src side data can change during the process of replication, which makes it difficult to use in practice. Also throws off state accounting, as we'll see later.
  - Still worth considering for import-side optimization, but this is easily done using MoveTask or the like instead of CopyTask on destination side, worth optimizing.

ReplicationTask & Command

- ReplicationTask has 1:1 map to Events in NOTIFICATION_LOG
- ReplicationTask can spawn multiple Commands, a Command is expected to be a single undoable/redoable action that is expected to take place in a linear fashion.
- While Events are atomic on source warehouse, Commands are atomic on dest wh, and if Commands fail, then HiveDR is expected to rollback all commands for the existing ReplicationTask
- This way, linearizable behaviour for ReplicationTask/Event is expected to be maintained, even if not directly atomic.

ReplicationTask

- Iterable<Command> getSrcCommands()
- Iterable<Command> getDstCommands()
- boolean isCopyRequired()

Other notes

- ReplicationTaskFactory instantiates ReplicationTasks. (currently provides NoopReplicationFactory & EXIMReplicationFactory) => extensible
- Always use Iterable<Command> and Iterable<Event> - actual list of Commands for a given Event can be arbitrarily large. Guava is awesome.
- Process each Command in sequence and roll back if necessary.

Command

- Interface that extends Writable

- List<String> get()
- List<String> getUndo()
- boolean isUndoable()
- boolean isRetriable()
- long getEventId()

Evolution of Command

- Command currently returns a List<String> to return hive statements that Falcon sends to HS2, but the future goal of Command is for it to be executable by itself, so that the end user need not attempt to do things like failure handling, recovery, redo/undo, etc.

Tasks

	Src commands	Dest commands	Needs copy?
CreateDatabase	Noop	Noop*	No
DropDatabase	Noop	Drop Database Cascade	No
AlterDatabase	Not yet implemented	Not yet implemented	Not yet implemented
CreateTable	Export ... FOR REPL	Import	Yes
DropTable	Noop	Drop Table … FOR REPL('id')	No
AlterTable	Export … FOR METADATA REPL	Import	Yes (only MD)

Tasks

Event	Src commands	Dest commands	Needs copy?
AddPartition	(mul) Export … FOR REPL	(mul) Import	Yes
DropPartition	Noop	(mul) Alter Table … DROP PTN(...) FOR REPL('id')	No
AlterPartition	Export … FOR METADATA REPL	Import	Yes (MD only)
Insert	EXPORT … FOR REPL *	IMPORT	Yes (currently dumb, but will get optimized)

Future work

- Event nullification
- Views, Functions, Roles (security)
- ACID compatibility
- Single EXPORT/IMPORT DB command?
- HDFS-based eventlog rather than metastore-table-based.
- If we ever get down to it : Things like kafka integration? Export-to-queue & Import-from-queue ?
- Implementing other ReplicationTaskFactories so we can do Hive->MySQL/ Hive->Oracle/etc kind of replications?

The end?


HIVE