Hortonworks

The Birds and the Bees Talk

(aka Replication)

Sushanth Sowmyan
<sush@hortonworks.com>
@khorgath

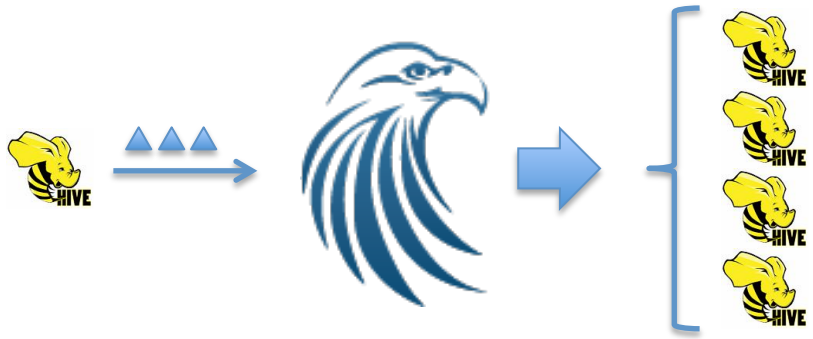
22nd April 2015, Hive Contributor's Meetup

Replication

- Need : Disaster recovery
- Solution : Replication, enabled through data management tools like Falcon
  - Nice to have : Load balancing (but not our initial thrust / reason we do this.)
- Idea : We already have export & import, and we already have notifications, so it should be a simple matter to extend them to implement replication, right?

Replication : Basic Idea

- Every transaction that makes a change writes out a redo log item that contains the change that happened.
- This delta is replayed on all the replicas to make them a faithful copy of the original.
- We add replication capability to hive, but leave the actual execution to data management tools like Falcon.



Before we go on..

- Approaches not considered:
  - Feed-definition-based
    - Other tools like Falcon already support this
    - Complex config, more in data management scope
  - HDFS+Metadata repl
    - Multiple sources of truth headaches
    - Partial replication issues

Taxonomy : Transaction source

- Transaction location : WHERE?
  - primary copy write : M-S models
    - simple concurrency control
    - not great for non-read load-balancing because primary is still a chokepoint
  - update anywhere : M-M models
    - good load balancing solution
    - complex concurrency control

Taxonomy : Synchronization Strategy

- Synchronization point : WHEN?
  - eager
    - strong consistency
    - can have performance impact, longer response times
  - lazy
    - fast, optimizable
    - can have stale data, temporary inconsistency

M-M models

- Great for load-balancing, but make concurrency control very difficult to achieve.
- "Good enough" if we achieve read-load-balancing for analytics loads.
  - If need be, we can allow replication isolation
    - T1 can replicate from A->B => T1 cannot be replicated from B->A, but does not prevent T2 being replicated from B->A
- We will ignore these for now until we solidify other aspects.

Synchronization

- Eager => Log is guaranteed written for every update that happens on the primary

- Lazy => Redo log is generated from the primary copy by a separate background daemon, done asynchronously

Eager Synchronization

- Log & Primary Copy Write approach
  - Write the log & the primary copy before returning successfully from a transaction
    - Costly in terms of disk & time
    - Synchronous, so blocking

Eager Synchronization

- Write out an in-place edit log first, and return successfully when that gets written, manage compaction semantics and versioning semantics for readers (similar to what's implemented for ACID on hive)
  - Good on performance, effectively asynchronous
  - Increased complexity
  - What happens if we return success to the user upon writing the delta, and then find that it fails on log-apply/compaction?
  - Not compatible with our notion of external tables - dependent on external writer to perform operations in a required manner, rather than traditionally used approaches of simply writing in data into the directory and registering new data with hive.

Lazy Synchronization

- Redo Logs generated asynchronously, typically by background daemons/tools
- Lazy-Redo Concurrent copy Approach:
  - Primary copy returns immediately, but needs to be able to provide snapshots or MVCC to make sure that this data does not change yet again during the time the redo log is being generated, keeps older copy around.
    - Complex to implement
    - What about DROP TABLE/etc ?
    - External tables and external management make this more difficult.

Lazy Synchronization

- Robustly-repeatable-replay of loose lazy redo logs
  (yes, I'm going for alliteration here : help me out here by suggesting more things I can add)
  - No guarantees that every single event translates to a unique delta => No need to keep versions at source.
  - Captures latest version at source for every event
  - Destination-apply needs to be robust and apply if the delta being applied is newer in "state" than the destination.
  - Can be temporarily out-of-synch, but will keep rubber-banding to latest state.
    - Bad for load balancing usecases
    - Great for DR
    - Stretches definition of "eventual" in eventual consistency.

Back to basics

- Event log for all operations on source warehouse
- {Create,Alter,Drop}{Database,Table,Partition}
- Export -> copy -> Import :
  - Generify : Commands on source wh, copy if needed, Commands on dest wh
- Translate each event to a ReplicationTask
  - (pluggable ReplicationTaskFactory allows third party implementation plugins)
- Third-party tool, like Falcon, consumes and executes ReplicationTasks.

Brass Tacks

- Creates => Export/Import if import is newer than dest object
- Alters => Export metadata/Import metadata if import is newer than dest object
- Drop = > Drop if event that spawned the drop is newer than dest object.

DDL Additions

- EXPORT [FOR [METADATA] REPLICATION(…)]
- DROP TABLE … [FOR REPLICATION(‘eventid’)]
- ALTER TABLE DROP PARTITION … [FOR REPLICATION(‘eventid’)]

Future Work

- Planned:
  - Event nullification/collapsing
  - Views, Functions, Roles replication
  - ACID interop
  - HDFS based eventlog rather than metastore-based one.
- Maybe?:
  - Single EXPORT/IMPORT command for database level
  - Export-to-queue & Import-from-queue semantics & kafka interop?
  - Other repl implementations like Hive->MySQL/Oracle/? & MySQL/Oracle/?->Hive
  - Improve load balancing usecase (Allow for barriers, maybe?)

References

- Umbrella jira : HIVE-7973
- Documentation tracking : HIVE-10246
- https://cwiki.apache.org/confluence/display/Hive/HiveReplicationDevelopment (work-in-progress)
- Kemme, Bettina, Ricardo Jiménez-Peris , Marta Patiño-Martínez, and Gustavo Alonso. "12.2 Basic Taxonomy for Replica Control Approaches." Replication Theory and Practice. By Bernadette Charron-Bost, Fernando Pedone, and André Schiper. Berlin: Springer-Verlag, 2010. N. pag. Print.