HiveReplicationDevelopment

Introduction

Replication in the context of databases and warehouses is the process of duplication of entities from one warehouse to another. This can be at the broader level of an entire database, or at a smaller level such as a table or partition. The goal of replication is to have a replica which changes whenever the base entity changes.

In Hive, replication (introduced in Hive 1.2.0) focuses on disaster recovery, using a lazy, primary-copy model. It uses notifications and export/import statements to implement an API for replication that can then be executed by other tools such as Falcon.

See Hive Replication for usage information.

Version 2 of Hive Replication

This document describes the initial version of Hive Replication. A second version is also available: see HiveReplicationv2Development for details.

Purposes of Replication

Replication is the process of making a duplicate copy of some object such as a database, table, or partition. There are two main use cases for replication: disaster recovery and load balancing.

Disaster Recovery

- Allows recovery of data after a disaster such as a server crashing unrecoverably or natural disasters such as fires and earthquakes.
- Prioritizes the safety of the data/metadata with no loss.
- Speed of availability is not as important.
- Hot-cold backup is often sufficient.

Load Balancing

- Allows the splitting of users across multiple systems to manage a heavier load than the system would normally be able to handle.
- Prioritizes speed and access.
- Tends to be read-only for most uses.
- Hot-warm backup is usually desirable.

Replication Taxonomy

Replication systems are frequently classified by their transaction source (“where”) and their synchronization strategy (“when”) [1].

Transaction Source

Primary-Copy

- Unilateral copy from primary to replica.
- **Pros:** Simple concurrency control (no locks introduced by the need for replication).

- **Cons:** Poor for non-read load balancing.

Update-Anywhere

- Allows bidirectional updates.
- **Pros:** Good for load balancing.
- **Cons:** Complex concurrency control.

Synchronization Strategy

Eager

- Creates a singular sequential stream of events.
- **Pros:** Guarantees strong consistency.
- **Cons:** Possible performance problems with long response times due to the requirement that no event may be processed until the replication of the current event has been staged.

Lazy

- Allows some reordering of events to optimize use of resources.
- **Pros:** Quick response time to user.
- **Cons:** Can have stale data on destination and temporary inconsistency.

Design

Taxonomy Design Choices

Replication in Hive uses lazy, primary-copy replication for a number of reasons as discussed below.

Primary-Copy vs Update-Anywhere

Since replication in Hive focuses on disaster recovery, the read-only load balancing offered by primary-copy replication is sufficient. The cost of complex concurrency control in update-anywhere replication is too high. Additionally, primary-copy replication can be isolated to be unidirectional per object, which allows for scenarios where two different objects can be replicated in different directions if necessary. This might be sufficient for some load-balancing requirements.

Eager vs Lazy

Eager replication requires a guaranteed delta log for every update on the primary. This poses a problem when external tables are used in Hive. External tables allow data to be managed completely outside of Hive, and therefore may not provide Hive with a complete and accurate delta log. With ACID tables, such a log does exist and will be made use of in future development, but currently replication in Hive works only with traditional tables.

Instead, Hive uses lazy replication. Unlike eager replication, lazy replication is asynchronous and non-blocking which allows for better resource utilization. This is prioritized in Hive with the acknowledgement that there is some complexity involved with events being processed out of order, including destructive events such as `DROP`.

Other Design Choices

In addition to the taxonomy choices listed above, a number of other factors influenced the design of replication in Hive:

- Hive’s primary need for replication is disaster recovery.
- Hive supports a notion of export/import. There are already some implementations of manual replication and time-automated replication with Perl /Python scripts using export/import and DistCp.
- Hive already has the ability to send notifications to trigger actions based on an event, such as an Oozie job triggered by a partition being loaded.
- UI is not intended as there are other data management tools (such as Falcon) that already cover data movement and include some form of feed replication. Hive should allow these tools to plug into the replication system. Hive handles the logic and provides APIs for external tools to handle the mechanics (execution, data management, user interaction).
- Timestamp/feed-based replication is solved by other tools and therefore Hive does not need to re-solve this problem.
- HDFS replication and backing metastore database replication are also possible using broad DFS copies and native replication tools of backing databases and therefore Hive does not need to re-solve these problems. Additionally, attempting to solve replication by these methods would lead to multiple sources of truth and cause synchronization problems as well as not easily allowing for a finer granularity of approach (i.e., they would not easily support replication of only hot tables or partitions).

Basic Approach

Hive already supports EXPORT and IMPORT commands which can be used to dump out tables, DistCp them to another cluster, and import/create from that. A mechanism which automates exports/imports would establish a base on which replication could be developed. With the aid of HiveMetaStoreEventHandler mechanisms, such automation can be developed to generate notifications when certain changes are committed to the metastore and then translate those notifications to export actions, DistCp actions, and import actions.

This already partially exists with the notification system that is part of the hcatalog-server-extensions jar. Initially, this was developed to be able to trigger a JMS notification, which an Oozie workflow could use to start off actions keyed on the finishing of a job that used HCatalog to write to a table. While this currently lives under HCatalog, the primary reason for its existence has a scope well past HCatalog alone and can be used as-is without the use of HCatalog IF/OF. This can be extended with the help of a library which does that aforementioned translation of notifications to actions.

Implementation

Hive’s implementation of replication combines the notification event subsystem with a mapping of an event to an actual task to be performed that helps automate the replication process. Hive provides an API for replication which other tools can call to handle the execution aspects of replication.

Events

Events are triggered when any change happens to the warehouse. Hive provides an API for tools to read these events. Each event has an associated event sequence ID (monotonically increasing). Once read, these events are then translated to replication tasks by means of a ReplicationTaskFactory.

Thus, the base replication mechanism in Hive is not tied exclusively to the export/import (EXIM) mechanism described previously, which instead serves only as a default event-to-task mapping mechanism. It is possible for an organization or a project integrating with Hive to plug in some other ReplicationTaskFactory implementation to achieve the same goal, which would allow modifications such as using additional bookkeeping or security aspects or using a different mechanism to move data from one warehouse to another. It is also theoretically possible to use a different implementation to achieve some other form of replication, even across database technologies, such as Hive --> MySQL, for instance. However, this document will focus on the details of the EXIM mechanism as implemented by EximReplicationTaskFactory.

Execution of events occurs after the mapping from event to task is completed. All aspects of execution, such as managing appropriate credentials, optimum cluster use, and scheduling, are outside of the scope of Hive and are therefore performed by replication tools such as HiveDR in Falcon.

Event IDs, State IDs, and Sequencing of Exports/Imports

As mentioned above, each event is tagged with an event sequence ID. In addition to event IDs, exports are tagged with the current state ID of the primary warehouse at the time when the export occurs (which will be a later time than the event itself) since replication is done asynchronously. This allows replication to determine if the state of the object at the destination is newer or older than that being exported from the source and make a decision accordingly at import time as to whether or not to accept the import. If the destination’s current state of the object is newer than the state of the object being exported from the source, it will not copy that change. This allows for robustness in the face of replication event repeats or restarts after failure.

Handling of Events

Each event is handled differently depending on its event type, which is a combination of the object (database, table, partition) and the operation (create, add, alter, insert, drop). Each event may include a source command, a copy, and a destination command. The following chart describes the ten event types and how they are handled with descriptions below.

Event	**Source Command**	**Needs Copy?**	**Destination Command**
CreateDatabase	No-op	No	No-op
DropDatabase	No-op	No	`DROP DATABASE CASCADE`
AlterDatabase	*(not implemented)*	*(not implemented)*	*(not implemented)*
CreateTable	`EXPORT … FOR REPLICATION`	Yes	`IMPORT`
DropTable	No-op	No	`DROP TABLE … FOR REPLICATION(‘id’)`
AlterTable	`EXPORT … FOR METADATA REPLICATION`	Yes (metadata only)	`IMPORT`
AddPartition	(multi) `EXPORT … FOR REPLICATION`	Yes	(multi) `IMPORT`
DropPartition	No-op	No	(multi) `ALTER TABLE … DROP PARTITION(…) FOR REPLICATION(‘id’)`
AlterPartition	`EXPORT … FOR METADATA REPLICATION`	Yes (metadata only)	`IMPORT`
Insert	`EXPORT … FOR REPLICATION`	Yes (dumb copy)	`IMPORT`

CreateDatabase

For security reasons, Hive does not currently allow the replication of a database before it exists since that may enable a replication of private data unintentionally. Accordingly, everything is set to no-op. However, this may be revisited in the future and therefore the implementation includes an API point.

DropDatabase

Drop does not require any action on the source side or any copying. On the destination side, it requires a drop database cascade. Although this is a dangerous command, the current requirement for a database to already exist before it can be replicated mitigates the risk by not allowing any replication definitions to be set up before the object exists and therefore this is considered an intended action.

AlterDatabase

This has not currently been implemented as Hive considers a database to be a container object. The only meaningful alteration would therefore be the HDFS location, which has no need for replication. Therefore there is no API point for this.

CreateTable

This requires an export on the source side, a copying of data, and an import on the destination side. This import will only execute if the import's state ID is newer than the replica object's state ID. Otherwise, it will be a no-op.

DropTable

Drop does not require any action on the source side or any copying. On the destination side, the table will be dropped if the state ID included in the drop command is newer than the state ID of the replica object. Otherwise, it will be a no-op.

AlterTable

This requires the export, copying, and import of metadata only. The import will only execute if its state ID is newer than the replica object's state ID. Otherwise, it will be a no-op.

AddPartition

Multiple partitions can be added atomically, resulting in multiple export commands bundled into one event and a copying of the data. The imports will only execute if their state ID is newer than the replica object's state ID. Otherwise, it will be a no-op.

DropPartition

Drop does not require any action on the source side or any copying. On the destination side, there can be multiple commands for one event, as with AddPartition. These are only executed if their state ID is newer than the replica object's state ID. Otherwise, it will be a no-op.

AlterPartition

This requires the export, copying, and import of metadata only. Unlike AddPartition and DropPartition, AlterPartition only modifies one partition at a time. The import will only execute if its state ID is newer than the replica object's state ID. Otherwise, it will be a no-op.

Insert

This requires the export, copying, and import of data (not metadata). Currently it is a dumb copy of the entire object (rather than applying only the changes made), although this is an area to improve and optimize in the future. The import will only execute if its state ID is newer than the replica object's state ID. Otherwise, it will be a no-op.

Future Features

In the future, additional work should be done in the following areas:

- Enable replication to work with ACID tables, especially with relation to load-balancing use cases. Load-balancing might also be needed only for specific hot tables, allowing for a more surgical use of replication.
- Limit unnecessary copies by allowing for event nullification/collapsing.
- Look at storing events generated on message queues like Kafka, or on-disk on HDFS, rather than in the metastore.
- Support replication of more Hive objects such as roles, users, etc.

HIVE-7973 tracks progress on developing replication in Hive.

References

[1] Kemme, B., et al., "Database Replication: A Tutorial," in *Replication: Theory and Practice*, B. Charron-Bost et al., Eds. Berlin, Germany: Springer, 2010, pp. 219-252.