Introduction

Replication Metrics will aggregate metrics related to Ranger, Atlas and Hive Replication. This will provide a consolidated view of the complete replication and not just Hive. This will also provide the progress information which can be queried by the user during the replication. A single replication specially bootstrap replication might take longer. So it is good to give the user information about the progress of the replication. Currently Ranger and Atlas service don’t provide any progress information but the metrics schema is designed in such a way that once they do, it can be easily captured. For Hive the progress information will be populated.

Metrics To Capture

1. Dbname
2. Replication type : bootstrap/incremental
3. Current Staging Directory
4. Time Taken
5. Total Count of Tables Dumped/Loaded
6. Total Count of Functions Dumped/Loaded
7. Total Count of Events Dumped/Loaded
8. Total Count of Policies(Ranger)
9. Total Count of Tags(Atlas)
10. Progress Information
11. Last Repl Id
12. Status
13. Logs

High Level Design

A transactional table on the relational DB(mysql and postgres) will be used to store the metrics, metadata and progress information. The schema design is discussed below. During the replication process, the progress information will be added to the database at regular intervals. Once the replication completes or fails, the status information will be updated. The intervals will be time intervals like every 10 mins. This can be kept configurable. There will be a view on top of the transaction table on the sys db. Clients can query the sys db directly using JDBC handler. Sys db is an internal hive database that holds system level data, logs and other system level information.
Retrieve Logs for the replication : The information_schema.scheduled_executions table already contains the executor id information. That can be used to retrieve logs related to a particular replication from the executor.

Retrieve Exceptions for the replication : The information_schema.scheduled_executions table already contains the executor id information. That can be used to retrieve exceptions or errors related to a particular replication from the executor.
Retrieve the Metrics : The RM or user can query the table in the sys db directly to get the metrics. Access to sys db will be through the JDBC handler. They can use the policy Id column to get the list of all executions. They can also query the latest execution run using the same and get the status and other details. If we compress the metrics field in the column(to reduce the space in db), while returning the query results, we need to prettify the json and send across the results in a user readable way. However we won't be able to support synchronous queries with tighter SLAs.
Clean Up : The older metrics will be cleaned up at a regular interval. A cleaner thread will run at a predefined interval(can be configurable) and clean up the metrics older than a specified time.

Schema Design

Metrics/Metadata ACID Table.

Column Name	DataTypes	Constraints	Description
Scheduled Execution Id	Long	Primary Key	This is the execution id of the scheduler. All queries scheduled using a hive scheduler has a execution id. Eg: any long value 13. Can be obtained by querying the information_schema. **select scheduled_execution_id from information_schema.scheduled_executions where scheduled_name =<>;**
Policy Id	Long	Index I1 Primary Key	This is the scheduled query id returned by the scheduler while scheduling a policy. All queries scheduled using a hive scheduler has a policy id. Eg : any long value. Can be obtained by querying the information_schema. **select scheduled_query_id from information_schema.schedule**

			d_queries where scheduled_name = <>;
Dump Execution Id	Long	Index I2 Primary Key	Empty for dump and Actual value for load. This is needed to tie the dump and load together to get consolidated metrics by Replication Manager. This is the scheduled execution id of the corresponding Repl Dump.
Metadata	Json(>My Sql 5.7) Varchar		Compressed Json 1. Dbname/Policy Name : This is the database name on which repl dump is called. Repl dump <dbname> or repl dump <*> 2. Replication Type : Bootstrap/Incremental 3. Staging Directory : Current Staging Directory where the data is dumped or loaded from. This is derived from the hive.repl.dir config. The current dump directory is appended to this path. Both source and target clusters need to be able to access this directory. The FS based ack is also done in this dir. 4. Last Repl Id : Last Replication Id of this dump/load for the database

Progress	Json(>My Sql 5.7) Varchar		Status : Success /Failed/Running Stages : This will be a json array(compressed) 1. Name - It can be one of the following Ranger Export/Ranger Import/Atlas Export/Atlas Import/Dump/Load 2. Status - Success/Failed. This will signify whether this stage succeeded or failed or is currently in progress 3. Metrics a. Name : Progress (Sample eg is Table : 1/10, Function : 2/2, Events : 1/10) - This will be updated to show the progress info 4. Start time 5. End time

Storage

This will be stored in a Relational DB. It needs the metrics column to be updated as the replication progresses. However we mayn't be able to support synchronous queries or tighter SLA's. The Replication Manager can't make blocking calls to update the UI based on the response. Currently it makes multiple getProgress calls which need to be relooked at. On the Hive side we need to look at better partitioning of data and data modeling so as to make read queries faster.

Reason for choosing Relational DB over Hive ACID

1. Frequent progress updates will result in creating multiple small delta files and it will be an overhead.
2. Hive cannot meet the transactional properties required. One transaction in Hive will result in 5 internal transactions which will be a big overhead.
3. Cons : Storage cost of Relational DB will be higher compared to S3.

Partitioning

A time based partitioning based on the clean up schedule. If customers need 7 days old metrics data, then we can have a daily partitioning. However if some customers are looking at retaining 6 months old data, we need to have less granular partitions. Based on the Relational DB used, we need to check how we can optimize this. It would be good if we can provide a range based partitioning. We will focus on MySql first and then Postgres.

CleanUp

The partitioning will be such that for clean up we needn't do a full table scan. We should have a time based partitioning so that we can drop the entire partition instead of doing a scan.
A config will be exposed to determine how long metrics need to be stored in the database.
Another config will be exposed to determine the frequency of the cleanup

Config Name	Description	Default Value
hive.metastore.metrics.expiry.duration	How old metrics will be maintained in the db(Values in secs)	604800s(7 days)
hive.metastore.metrics.clean.frequency	How frequently the cleaner thread will run(Value in secs)	86400s(1 day)

Based on time and not objects
Schema change
Partitioning