Replication and House-keeping tasks

Objective

An HMS performs a number of housekeeping tasks. Assess whether

1. They are required to be performed in the replicated data
2. Performing those on replicated data causes any issues and how to fix those.

Even though we support table level replication, a database can not contain replicated and non-replicated tables together. Hence it's still the database which is replicated or not. We can rely on database level property to decide whether a table/partition in the database is replicated.

Incremental fail-over consideration

Consider a case where replication is setup between sites SRC and DST. When SRC fails over to DST, DST becomes source let's call it DST_SRC2 and will be used for all active transactions thenceforth. If a user wants to setup replication from DST_SRC2 to SRC, thenceforth called SRC_DST2, becomes the target of replication. Today, we need to bootstrap SRC_DST2 again, which is unnecessary in a controlled fail-over, where the data on SRC_DST2 is same as SRC. Replicating only the events on DST_SRC2 which are not available on SRC should suffice to bring SRC_DST2 in sync with DST_SRC2. Furthermore, if we want to perform a controlled fail-over from DST_SRC2 to SRC_DST2 and use the later as new replication target, so that they become DST_SRC2_DST3 and SRC_DST2_SRC3 resp., we should be applying only the events on SRC_DST2_SRC3 which are not available on DST_SRC2_DST3.

In this scenario the source and target of replication keep toggling between two sites. The incremental fail-over feature, which is not yet supported, proposes to use just incremental replication without any bootstrap to continue replication between the source and target to keep them in sync even after their roles are toggled.

When the target of replication becomes source i.e. DST becomes DST_SRC2, we need all the HMS tasks to perform their work on all the objects even if they were earlier the target of replication. If we rely on one or more table or database level properties to identify the objects the HMS tasks should work on, we need to make sure to wipe out/reset those properties during a controlled failover so that the task does not skip those objects.

Task-wise analysis

Following table lists the tasks with assessment of each.

Name of task	**Description**	**Requirement on replica**	**Issues caused and cure.**
statsUpdaterThread and statsUpdaterWorker threads	Update stats	The statistics is replicated from the source to target. If the source doesn't have statistics updated and this task is enabled on the replica, the stats will get updated on the replica. In a case where user doesn't want up-to-date stats on the source but want them up-to-date on the replica, having the stats updater thread run on the replica alone might be useful. But it causes issues.	StatsUpdatedThread runs Analyze commands, which take writeId on a transactional table. This means the writeIds on the target and the source can go out of sync if stats updater gets to update the statistics on the target. So,we shouldn't be running the thread on the replica. Given that the replica can have non-replicated databases which require this task, we should not disable the config in HMS. ~~Instead we should filter out the tables in the replicated database/s in getAllTableNamesForStats.~~In processOneTable() check for the repl related property to filter out any tables being the target of replication.
Repl change manager cleaner thread	Cleans the files in CM periodically.	If the target is source of further replication (cascaded repl case), we require ReplCM and hence CMCleaner. Since the CMCleaner removes files based on the modification time, which is dependent solely on the operations on the	

		target, this task can be configured and run independent of the source.	
CompactorInitiator, CompactorCleaner and CompactorWorker threads	Runs compactions periodically.	Since compactions are not replicated, we need it on the replica as well. Note: We are already disabling compaction for replicated target db till the first incremental is completed. Since the compaction on the target takes into consideration the transaction state on the target, running them independent of source doesn't harm consistency.	
Remote only tasks (specified by TASK_THREADS_REMOTE_ONLY configuration)			
AcidHouseKeeperService	Aborts any transactions without heartbeat for a long time.	On a replicated db, the only transactions that are allowed are compaction, read-only and repl transactions. The first two transactions need to be aborted if they run without heartbeat for long time. And this task does not include transactions created by repl in the list of transactions to be aborted. So, we are	

		safe (See org.apache.hadoop.hive.metastore.txn.TxnHandler#performTimeOuts())	
AcidOpenTxnCounterService	Counts number of open transactions periodically.	Since we allow opening new transactions on a replicated db, we should maintain this counter. Since we open all the transactions on the source on replica as well, and plus replica only transactions, the number of open transactions on replica could go higher than that on the source. Hence we should configure a higher number for (metastore.max.open.txns) on target compared to the source or not count repl transactions. Either way, this is out of scope of this work.	
AcidCompactionHistoryService	Purges obsolete items from compaction history data.	Since we are running compactions on the replica target, this service is needed anyway. This is independent of the source so no harm in having it running on replica.	
AcidWriteSetService	Periodically cleans WriteSet tracking information used in Transaction	Since the writes on replica are performed in a serial manner, there will not be any	

	management.	conflicting txns. Thus maintaining write_set is not required for a replicated db and hence the task is not required on a replicated db. But removing write_set only for repl would be a basic surgery out of scope of this work. Since write-sets are being created we need a task to purge them. If we didn't created them, it would be required for the non-replicated dbs.	
MaterializationsRebuildLockCleanerTask	Removes outdated materialization locks periodically.	I don't remember, our take on replicating materialized view. If we aren't replicating materialized views, we won't create materialization locks for the replicated db and thus running this task won't harm. It will be anyway needed for the materialized views in other non-replicated databases. If we decide to let the materialized views to be built on the replica, we will need this task to purge the materialization locks that they create. So either way, we need this task on the replica as well.	
PartitionManagementTask	Partition retention: drops any partition with age greater than	Since partition management is performed on the	If the task adds/removes partitions on target such that the set of partitions

	retention period. Partition discovery: adds partition for newly added locations, drops partition corresponding to the missing location.	source and replicated to the target, we do not need this task for the replicated db. However, for the dbs other than replicated db, we require this task. So, on the replica, we need to enable this task (if there is any non-replicated db there) but it should not work on the replicated dbs.	on the source and that on the target go out of sync, the data in the partitioned table may be inconsistent with the source and other tables on the replica. See Avoid running partition discovery and retention for replicated databases for details on the changes required.
HiveProtoEventsCleanerTask	Used by DAS to capture timings, events, configs etc.	Since the source and replica have different DAS	
DumpDirCleanerTask	Delete old dump directories.	In case of cascaded replication, we need this task on replica to clean old dump directories.	
EventCleanerTask	Cleans old partition events.	The only event supported seems to be load_done. I couldn't find out where do these events get added from. But it looks like it's something to do with writing into partitions, from outside Hive. That should not happen on a replicated db. Hence no events will be created on replica. We need to run this task if the target has non-replicated dbs. Since no partition events are expected	

		to be created on a replicated db, even if this task is running on the replica cluster it should not cause any harm. It will be good to disable this task when all the databases in a cluster are being replicated into. We should leave that to DLM/CM or user to configure.	
RuntimeStatsCleanerTask	Cleans run-time stats related to query-reexecution?	Since queries are expected to be run on the target, run time stats will be added and hence required to be cleaned up on target as well.	

Avoid running partition discovery and retention for replicated databases

Here are ways to avoid running partition discovery and retention for replicated databases

1. The metastore configurations “metastore.partition.management.catalog.name”, “metastore.partition.management.database.pattern” and “metastore.partition.management.table.pattern” decide which tables are considered by this task for partition management. If “metastore.partition.management.database.pattern” here does not include replicated databases, this task will not consider the tables in replicated db for partition management. DLM knows about the databases being replicated into, thus it can adjust this config such that those databases are not listed in the config.
2. If a cluster contains only replicated databases, the partition management task itself should be disabled. A user would know whether a cluster has any databases other than the ones being replicated into. So a user could disable the task itself in such a case.
3. As part of replication, the repl dump/load removes table properties “discover.partitions” and "partition.retention.period" from the replicated tables. The partition management task

wouldn't work on such tables. But this won't work with the controlled fail-over scenario described above, where the values of these properties set by user are required to be faithfully replicated to the target and used when the target becomes the new source.

4. When searching for candidate tables in PartitionManagementTask#run(), do not include the tables in the database being replicated into. We could check the dblevel property as well, but that would be an extra lookup since the Database object isn't handy in this code.

The fourth option looks a viable option and can also be used for stats updater thread.

Table level property to identify the target of replication

From the above analysis it looks like we will use a table level property to identify whether a table is target of replication and stop an HMS taks working on it. Here's analysis of various properties to decide which of those can be used.

1. **hive.repl.ckpt.key** : this property is used to check whether an object has been bootstrapped during replication. In case of incremental fail-over we will need this property set for the objects on the extant target so that incremental doesn't skip events corresponding those objects during replication. So, we can not wipe-out this property when the source and target switch roles for a controlled fail-over. **Hence we can't use this property for our purpose.**
2. **hive.repl.first.inc.pending** : this property is added during bootstrap and removed after first incremental has finished. So, **we can't use this property** as well when replication has progressed beyond the first incremental cycle.
3. **repl.last.id** : for a database or a table this property is used to skip the events already applied to the database or the table (although I didn't find the code for skipping events on a table). For this we check whether the given event has id lesser than the repl.last.id for the given object. In case of an incremental fail-over, we need to wipe out last.repl.id from all the tables of the new target. That should be fine, since the tables on the source are not expected to have this property. But when the source and target switch roles again, the last.repl.id value from the last cycle is gone. This should be safe, since that value should not have any relevance to the latest incremental cycle. We might have to add a new last.repl.id value in the would be target and set it so that all the events in the first incremental cycle after fail-back are applied. **We could use this property.**
4. **repl.is.target** : a new property to indicate whether this object is being replicated into. A new property should be safest. But we need to add it when a new target (as part of a fresh bootstrap or as part of fail-over) is being prepared and remove when target becomes the source. Which is similar to what we have to do for repl.last.id as well. So, instead I would use the existing property itself.