k2view

Alpinist

LUI Sync Components - Part 2

WELCOME

Agenda

- Schema change
- Encryption
- Pools & cache
- Parallel sync of the same instance

LUI Schema change

Upon modifying the LU Schema metadata, every existing LUI in Fabric must undergo a schema upgrade process:

- With each GET operation, Fabric verifies whether the LUI requires a schema upgrade by comparing the schema_hash property of the LUI with the latest deployed schema hash.
- If an upgrade is required, Fabric compares the LUI schema with the latest schema and proceeds with the upgrade process.

During the Schema upgrade process, populations will run automatically only for:

- New table added
- Table with a new column added
- Table with a new link connecting to a parent table
- Table that a link to a parent table got removed
- Table with newly added/removed index

SQLite doesn't support altering column type. Therefore, when a column type is changes, Fabric performs the below steps:

- Creates a temporary table within the LUI.
- Copies the data from the original table to the temporary table.
- Deletes the original table.
- Renames the temporary table to match the original table's name.

LUI Schema change

1. Changing population logic alone does not automatically activate it.
   To ensure the population runs after a schema upgrade deployment, you can:
   - Develop custom logic to activate the population
   - Deploy the change using Force Upgrade Post Deploy, after which the Sync mode is automatically set to FORCE during the first sync of each LUI.
     Note: even after unchecking this option, LUIs will keep syncing in FORCE on their first sync.
2. A schema change also impacts GET in Sync OFF. The schema change logic runs, but since the LUI is not saved to Cassandra, the changes do not take effect until the first Sync ON.
3. Populations are automatically activated for tables whose structure has changed, following the Sync policy. However, it's important to note that populations of child tables are not triggered automatically.
   1. Develop custom logic to activate the population, using the isStructureChanged() built-in function to identify the scenario.

LUI Encryption

Fabric offers a built-in functionality for encrypting either the entire LUI or specific data within the LUI (like PII data).

More details are provided in the Security course.

Best practice:

1. Do not encrypt the LUI if not needed. Encrypt and decrypt add time to the Sync duration. Instead, if needed, you can encrypt only the required data inside the LUI.

2. If the LUI encryption is done before the LUI is compressed, the LUI size increases

Pools & cache

Fabric pool

Connections to Fabric are managed using a pool. The pool's connections are established once Fabric is starting

- Pool size: config.ini.[fabricdb].MDB_CONTEXT_POOL_SIZE=200 (size is fixed, no minimum or maximum).
- Timeout to wait on connection: config.ini.[fabricdb].MDB_CONTEXT_POOL_GET_TIMEOUT_MILLIS after which exception will be thrown.

Note:

When setting MDB_CONTEXT_POOL_SIZE =0:

1. *There will be no limitation to the number of connections to Fabric*
2. *Connection to Fabric will be lazy - opened on request and discarded after each use*
3. *There might be performance impact. Tune carefully and test before*

Use case: Specific scenarios where numerous connections need to be temporarily open, without increasing the pool size in order to avoid excessive resource and memory usage.

Pools & cache

Broadway pool

Fabric keeps a pool of compiled Broadway flows per lu/flow.

The size of the pool is set in config.ini. [fabricdb]. BROADWAY_LU_POOL_SIZE. Default is 200 (per node). If more flows are used in parallel, they will be parsed and disposed after use.

Note:

Tune this parameter in case you are executing a Broadway flow more than 200 times in parallel and observe performance issues

Pools & cache

LUI prepare statements cache

Every Fabric session keeps a cache for the prepared statement.

The max size of the cache is set in config.ini. [fabricdb]. MDB_PREPARED_STATEMENT_CACHE_LIMIT. Default is 200

If cache reaches the limit, a warning message will appear in logs:
FabricDB Prepared Statement Cache limit reached 200

As a workaround, and until issue is found, you can set the parameter to 0 to avoid caching.

Pools & cache

Prepared statements best practice:

- Use Non-Bind variables in an SQL statement only when the values are constant values.
  For example:
  - Select * from customer where customer id = ? And timestamp = 181736329298

    If timestamp changes for every select, use one of the options below:
  - Select * from customer where customer id = ? And timestamp = ?
  - Select * from customer where customer id = 123 And timestamp = 181736329298
- In any other case, use Prepare and send the values as parameters to the SQL statement.

Note:

In Fabric 6.2 and above, binding is supported for all Fabric commands.

For example:
fabric().execute("get Customer.?", cust1);

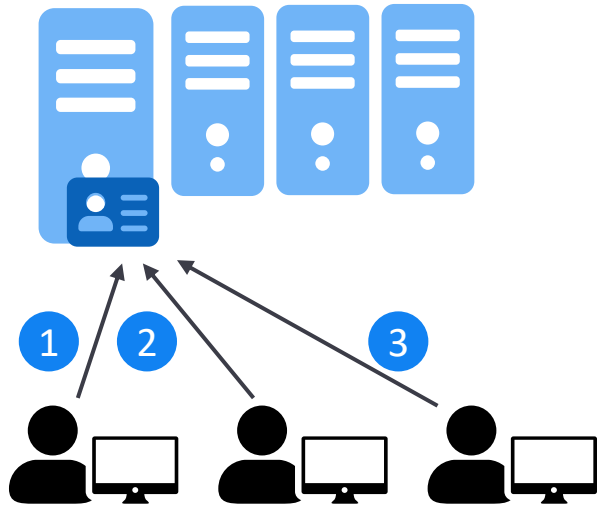
Parallel Sync of the same instance

Parallel GETs of the same instance on the same Fabric node

Running GET with Sync ON fetches the MDB file to the cache storage and performs a **write lock** of the SQLite database on attach (open a transaction).

Sequential GET on the same node for the same instance are trying to use the same MDB file. Therefore:

- If Running in Sync ON - will not be able to open a transaction as the file is already in an exclusive lock.
  Therefore, it will wait until the locking is released (even if no changes are required).
- If Running in Sync OFF - will wait on read if LUI is inside a transaction.



Parallel Sync of the same instance

Parallel GETs of the same instance on the same Fabric node

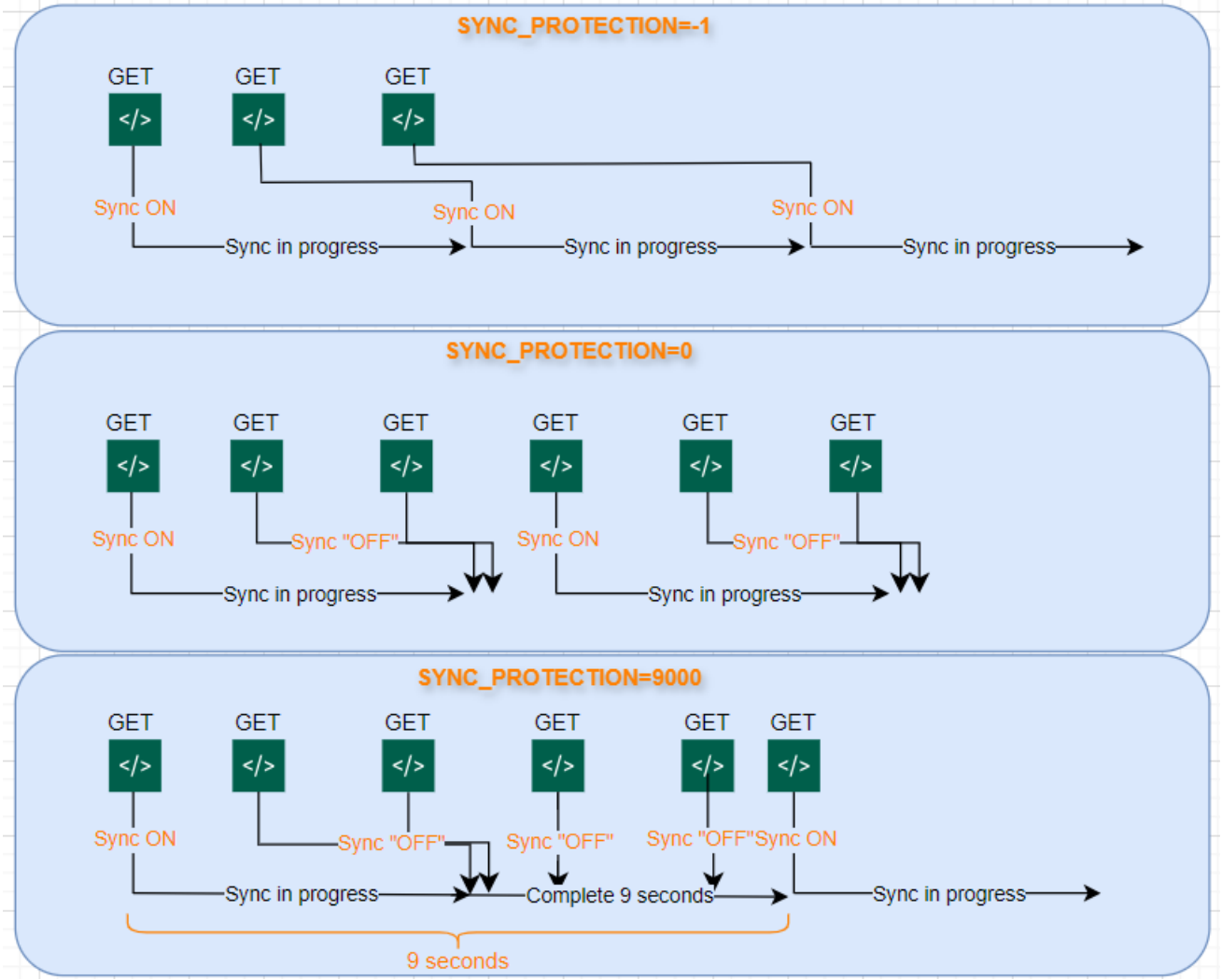
To improve the response time of multiple GETs for the same LUI on the same Fabric node, Fabric supports a time window on which the sequential GETs in Sync ON will behave like Sync OFF (populations will not run) and the cache validation, that the MDB is up-to-date will not be performed.

To activate the time window, set the **SYNC_PROTECTION** in config.ini as follows:

- The default value is **zero**. When Sync is set to ON, Fabric implements the Sync only on the first request. The following GETs will be treated as Sync OFF until the first Sync has been completed.
- If this parameter is set to **-1**, Sync ON Protection is disabled and Fabric implements the Sync on each request. All requests have Sync set to ON in this case.
- This parameter can be set in **milliseconds**. In such case, each SYNC activated from the time in which the first Sync ON has started, and until the defined milliseconds setting has passed, will be treated as Sync OFF.

Parallel Sync of the same instance

Parallel GETs of the same instance on the same Fabric node



Parallel Sync of the same instance

Parallel GETs of the same instance on the same Fabric node

- During the SYNC_PROTENCTION time window, the LUI version will **not** be validated against the storage, for every sequential GET
- Time to wait for the MDB to be released can be defined in config.ini MDB_ATTACH_TIMEOUT parameter. The default is 10000.
- SYNC_PROTECTION can be disabled on the session level using the SET SYNC_PROTECTION=off command.

Fabric 8 will introduce a new mechanism utilizing a write-ahead log (WAL) file on the database file to manage transactions. This will allow read operations to be performed while the file is in transaction.

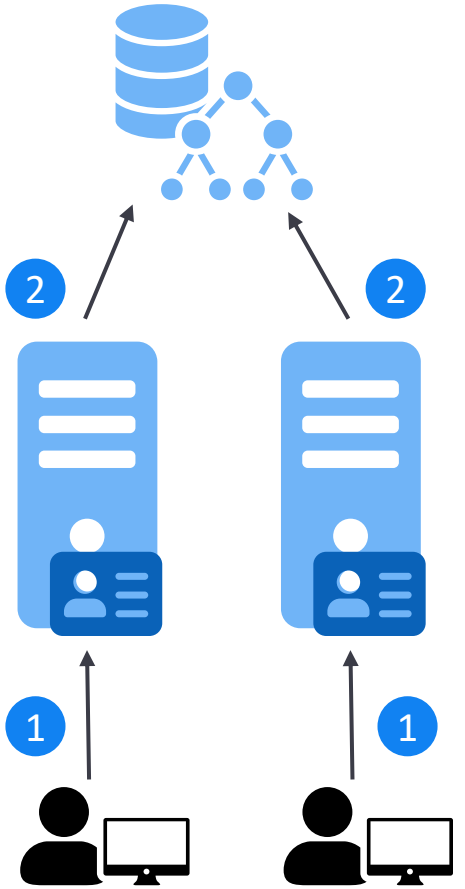
Parallel Sync of the same instance

Parallel GETs of the same instance on a different Fabric nodes

In case multiple GETs are running concurrently on the same LUI on 2 different nodes, the locking mechanism is not enforced because each node operates on its own cached file.

As a result, it is possible for both nodes to attempt to update the same instance at the same time.

Therefore, when storing the MDB file back to the storage, the changes made by the latest node may override those made by the first node.



Parallel Sync of the same instance

Parallel GETs of the same instance on a different Fabric nodes

To avoid overriding data, use the optimistic locking mechanism configured in config.ini.[system_db_entity_storage].**OPTIMISTIC_LOCKING** as follows:

- **NONE** (default). The latest transaction overrides the LUI (Instance ID).
- **QUORUM**. The latest transaction fails (the commit of the first sync requires a quorum).
- **LOCAL QUORUM**. The latest transaction fails (the commit of the first sync requires local quorum on the DC (Data Center)).

When employing optimistic locking, an error will be raised during the process of saving the LUI back to the storage if the LUI version has changed since it was first fetched.

Parallel Sync of the same instance

Parallel GETs design considerations

1. Processes that sync instances should ensure that the same instance is getting synced on the same node, to avoid syncing the same LUI on 2 different nodes at the same time.

   Other processes such as APIs, jobs, or flows should use the LUI in Sync OFF When possible.

2. To avoid data override, set OPTIMISTIC_LOCKING with either 'QUORUM' or 'LOCAL_QUORUM'.

   This approach is recommended when dealing with SOR data or when running simultaneous syncs of the same LUI on different nodes (for example, Sync FORCE on first node and Sync ON on the second).

3. When using gcp or azure as the entity storage, configure VALIDATE_REMOTE_VERSION=true in [gcs_storage] or in [azure_blob_storage] instead of the [system_db_entity_storage].OPTIMISTIC_LOCKING parameter.

4. OPTIMISTIC_LOCKING functionality is not supported when using S3.