Uniqorn: Informal Consistency Proof

Paper # 222

1 Informal Proof: Uniqueness

We call the embedded version in a data record or index record as a lock in the proof (hope it is more understandable). Only one index record with a given alternate key (alternate keys serves as both partition keys and identifiers of index records) can persist in index store. If we can prove that for any persisted data record in data store, its derived index records are never absent in index store at any time, we can prove that no two persisted data records in data store can have the same alternate key at any time.

We will prove the consistency property "no derived index records of any persisted data record in data store are ever absent in index store" deductively. A read or delete operation never adds or deletes any index records into index store, therefore, we just need to consider create, update, and garbage cleanup operations. In Uniqorn, only garbage cleanup operations attempt to delete garbage index records. Also, every lock change will increase the priority of the lock. Moreover, no lock has higher priority than the lock embedded in a currently persisted data record.

We will first prove that create or update operations are consistent. A create or update operation of a data record creates or reads the latest (which is also the highest-priority) lock of the data record first, then persists all of its index records (each of them bears the lock) of the data record into index store, and in the end persists the data record into data store only if the create or update operation still holds the lock. For the create or update operation, the success of persisting the data record into data store has three meanings. First, no other concurrent create, update, delete operations have ever modified the data record since the start of the operation, so all additions or changes of alternate keys of the data record are captured and have been persisted into index store. Second, no garbage cleanup operations have changed the lock that has been initialized or updated by this operation at the start of the operation (a garbage cleanup operation needs to change the lock of the data record referred by a garbage-suspected index record before deleting this garbage-suspected index record). Therefore, none of its index records are missing at the moment that the operation successfully persists the data record into data store. Third, after the data record has been successfully persisted in data store, all of its index records become valid. Uniqorn never attempts to delete a valid index record, so no valid index records will ever be absent until the data record is deleted from data store by a delete operation. An aborted create or update operation does not cause any inconsistency since it never deletes any index record directly (though may indirectly call a garbage cleanup operation to delete a garbage-suspected index record for the purpose of reusing the alternate key) and it fails to modify any alternate key or attribute of the data record.

We will next prove that garbage cleanup operations are consistent. A garbage cleanup operation can be invoked by any other operations or manually by operators to clean up stale garbage index records. A garbage cleanup operation starts with reading the data record referred by a garbage-suspected index record. The index record is validated against the read data record in the first place. The garbage cleanup operation will abort if the index record is valid (i.e., the read data record does have the alternate key of the index record), therefore valid index records will stay valid and persisted. Otherwise, if the index record is validated as garbage, the garbage cleanup operation changes the lock of the read data record. Any ongoing create or update operation that had changed the lock before the start of the cleanup operation will be definitely aborted since they lost the lock to the cleanup operation. Their added or updated index records (bearing a lower-priority lock than the lock of the cleanup operation) become garbage (no possibility to turn into valid) and can be safely deleted by the cleanup operation. Therefore, the cleanup operation will never delete any valid index records.

Therefore, all CRUD and cleanup operations are consistent, therefore, we prove that "no derived index records of any persisted data record in data store are ever absent in index store" and hence the uniqueness property of Uniqorn.

2 Informal Proof: Linearizability

The linearility point is the commit point of every modification (insert, update, delete) of the data record, is the read point of every read that reads the data record. That is all because of the data store (i.e., all of the data records are the source-of-truth). The index store (i.e., all of the index records) are just references to their data records. Based on our assumption that the storage engines (powered both index store and data store) are read-consistency (that is, reading a record always returns the latest committed/persisted record in real-time/wallclock-time), which is linearizability). So CRUD by alternate keys are linearizable.