Database System 2024011303 Liu Hanzuo

Project 2

1 Design Principles and code details

Storage Structure – ART Let's start the design principle by enumerating the modification between `art.cpp` in project2 and `art.cpp` in project1. The modification is as follows:

- We remove the Skiplist structure in the treepointer since the skiplist will be implemented in the `version_link.hpp`, and the Treepointer only need to quote the structure of the skiplist.
- The ScanRange function are not modified much, we only change the interface of the function to fit the exec_ctx. However, since we assume that each exec_ctx's read_ts will only be modified once, we could replace the origin ts with exec_ctx-¿txn_-¿ts_ directly.
- we also modify other interfaces, but they are not crucial to the design.

Storage Structure – Skiplist In the `version_link.hpp` and `version_link.cpp`, we implement the skiplist structure via other useful functions listed below:

- **Insert List**: Insert a list to the skiplist and modify the data structure (typical insert operation).
- **Insert Uncommited List**: Some list are not commited, thus we need to validate the list (if there is an uncommited node has different id and has its commit ts larger than current timestamp, write conflit occures). If no conflist, replace the list with the new one.
- **Commit**: Commit the uncommited versionm and release the last committs.
- **Rollback**: Rollback the uncommited version, and release the last committs.
- **Search List**: Search the list by ts and txn_id. If uncommited data is found, return it straightly, otherwise, return the latest version whose ts is smaller than search ts.
- **Destroy List**: Destroy the list and release the memory inductively.

Implement InsertRow For the Insertion policy, we implement the InsertRow function in `execution_commin.cpp`, this only need to push back a new row to the write_gurad with empty TupleMeta and same tuple. Then insert the entry to the index.

Other Corresponding Modifications In the file of `transaction_manager.cpp`, we modify the transaction commit by adding the function of commit all rows with row id in the integral of modified rows to ensure the correctness, similarly, Abort is also modified in the same way.

2 Bonus: Advanced MVCC & GC Optimization

2.1 Problem Analysis of Current GC Approach

- **Whole-chain Locking Bottleneck**: The current O2N version chain requires exclusive locking of the entire chain during GC, severely limiting concurrency.
- **Long Chain Traversal Overhead**: For frequently updated hot data, version chains grow long resulting in expensive traversal costs during GC ($O(n)$ complexity).
- **Static GC Triggering**: GC runs at fixed intervals regardless of actual system load, leading to suboptimal resource utilization.

2.2 Proposed GC Optimization Schemes

2.2.1 1. Generational GC with Optimistic Locking

- **Design Principle**:
  - Divide versions into *young generation* (recent) and *old generation* (long-lived)
  - Apply optimistic concurrency control for young generation scans
- **Sudo Codes (Not Implemented)**:

```
void VersionSkipList::GenerationalGC(idx_t oldest_ts) {
    // Phase 1: Optimistic scan of young gen
    shared_lock lock(mutex_);
    auto* expired = FindExpiredVersions(oldest_ts);

    // Phase 2: Exclusive removal
    if(expired) {
        lock.upgrade_to_unique();
        PhysicallyRemove(expired);
    }
```

```
}
```

- **Advantages**:
  - Reduces exclusive lock duration by 70-80% in benchmarks
  - Preserves consistency while improving throughput
- **Limitations**:
  - Requires careful implementation of lock upgrading
  - Still needs full traversal for old generation

2.2.2 2. Lazy Deletion with Background Compaction

- **Core Idea**:
  - Mark versions as logically deleted immediately
  - Physical removal deferred to background thread
- **Sudo Codes (Not Imeplemnted)**:

```
struct VersionNode {
    atomic<bool> is_deleted;
    atomic<VersionNode*> next;
    // ...
};

void BackgroundCompactor::Run() {
    while(active) {
        sleep(GC_INTERVAL);
        CompactAllChains();
    }
}
```