Project Proposal: Replicated Concurrency Control and Recovery

CSCI-GA.2434-001 Advanced Database Systems

Zien Yang(zy2236), Wenbo Song(ws1542)

October 2021

1 Introduction

This project implements a distributed database, complete with multi-version concurrency control, deadlock detection, replication, and failure recovery.

In this project, there are twenty variables and ten sites in total. For each transaction, the multi-version read consistency is used for the read-only transactions, otherwise the strict two-phase locking protocol is used to avoid non-serialization.

A graph data structure is used inside the transaction manager to keep track of the dependencies of current transactions. At the beginning of each tick, a graph traversal is needed for cycle detection to find potential deadlocks.

Available copy algorithm is used for data replication to enhance fault tolerance within the transaction manager.

2 Design Detail

2.1 Transaction Manager

- Data Structure
  - A graph for the dependencies of current transactions.
  - A list of sites (Data Manager) with the length of ten.
  - A queue to memorize the sequence of the transactions.
  - A class of transaction.
  - A queue to store the executing sequence of transactions.
  - A hash-map to store the commands for different transactions.
- Functions
  - **begin(transaction_id):** @Zien
    The function will create an instance of a transaction.
  - **beginRO(transaction_id):** @Wenbo
    The function will create an instance of a read-only transaction.
  - **read(transaction_id, variable_id): boolean** @Zien
    The function will go through all the available sites, and try to do a read operation base on the transaction_id and the variable_id. It returns True when the read operation is success; otherwise it returns False.

- **write(transaction_id, variable_id, value): boolean**@Wenbo
  The function will go through all the available sites, and try to do a write operation base on the transaction_id and the variable_id. It returns True when successfully write to all sites; otherwise it returns False.
- **dump()** @Zien
  The function will print out the committed values of all copies of all variables at all sites.
- **end(transaction_id)** @Wenbo
  The function will end a transaction and print whether this transaction is committed or aborted.
- **fail(site_id):** @Zien
  The function will turn down a site and change the status and erase the lock table.
- **recover(site_id):** @Wenbo
  The function will recover a site and change the status.
- **deadlock_detection():** @Zien
  The function will traverse the waiting-for graph to check if there is a deadlock situation. If there's a deadlock, it will abort the youngest transaction.

2.2 Data Manager

- Data Structure
  - Site status.
  - An hash-map to memorize the holding variables.
  - An hash-map to memorize the locking status of each variable with the corresponding transaction_id. If the site is down, the lock table will be erased.
  - A list of uncommitted variables.
- Function
  - **add_lock(variable_id, lock_type): boolean** @Wenbo
    The interface which is called by the Transaction Manager(TM) when the TM needs to add locks on a variable of this site. It returns True when the specified lock can be applied on the variable; otherwise it returns False.
  - **release_lock(transaction_id): boolean** @Zien
    The interface which is called by the Transaction Manager(TM) when the TM needs to release locks of a transaction on this site.
  - **update(variable_id, value): boolean** @Wenbo
    The interface which is called by the Transaction Manager(TM) when the TM needs to write to the variable. It returns True when the variable can be written to the specified value; otherwise, it return False.
  - **get_value(variable_id): current value of a variable** @Zien
    The interface which is called by the Transaction Manager(TM) when the TM need to read the variable. It returns the value of the variable on this site.
  - **commit(variable_id, value): boolean** @Wenbo
    Commit the changes made to a variable.
  - **abort(transaction_id): boolean** @Zien
    Aborts the transaction on this site.

- **fail(): boolean** @Wenbo
  The lock table should be erased.
- **recover(): boolean** @Zien
  All non-replicated variables are available for reads and writes. All replicated variables are available for writing, but not reading.
- **snapshot(): a snapshot of the variables and their values** @Wenbo
  Returns a snapshot of the site for read request from a read-only transaction.

2.3 Component Relationship Figure

The design detail is fully explained in the above section. Therefore, the design figure shown below focuses on the relationship between different design components.