# Lab 3: Transactions Report

By Yuan Yuan (yy189) & Yuxi Yang (yy191)

In this project, we have 10 keys (numKeys = 10). Every transaction has exactly 2 operations, either read or write, depending on probability. Transaction is represented as list of operations. Every transaction has a transaction ID tranID: BigInt.

Class Operation

| tranID: BigInt | key: BigInt | r: RingCell<br>(null if read) | readOrWrite: Int<br>0 for read, 1 for write |
|---|---|---|---|

transaction: List[Operation]

| Operation 1 | Operation 2 |
|---|---|

We prestore the 10 <key, value> pairs in KVStore with RingCell(0, 1) in case we need to read a pair before write it.

We use prepareRead and prepareWrite (write RingCell(random value, value+1)) to assemble transaction. After that, we call KVClient's begin to do Prepare in 2PC.

In this project, we use KVClient as our coordinator and the participants are the KVStores involved in the transaction. Since every transaction has exactly 2 operations, at most 2 KVStores need to vote. So, numVotes = 2 at the beginning of 2PC: Phase 1.

We need to deal with every operation in the transaction list. If the operation is read, we first check cache. If cache contains the <key, value> and it's not dirty, print "Cache: RingCell(prev, next)" and numVotes -= 1 (We no longer need to contact a KVStore for that

value).

If the <key, value> is not in cache, we ask the KVStore containing that key to prepare. If get the vote (Yes) before timer expires, numVote -= 1. If the KVStore can't get the requested lock before timeout or deadlock occurs, it will give no response to the sender.

If the operation is write, we put the <key, value> to both cache and dirtyset and ask the KVStore to prepare.

At the KVStore side, if the message received is Prepare (Operation), it will first save the transaction to log. In order to make the log be of chronological order, we use a HashMap from tranID to index. Log(index) is the index$^{th}$ arrival of operation.

Class Holder

| tranID: BigInt<br>(which transaction takes this key) | readOrWrite: Int<br>0 for readlock, 1 for writelock |
|---|---|

lock: HashMap[BigInt, Holder] <key, holder>

Then, we check the lock map. If no transaction is occupying the lock or the lock being occupied is a read lock and the requested lock is also a read lock, get the lock and vote yes. Otherwise, there is a reading and writing conflict. We set a scheduler to wait for 10 ms. After 10 ms, if the lock becomes available, get the lock and vote yes; else, vote no.

Let's go back to KVClient side. If all KVStores involved voted yes (numVotes == 0), we do commit. For every operation in the

transaction except the read-cache ones, we ask the corresponding KVStore to commit.

If the operation is read and we get the value before expiration, we put the <key, value> into cache and print “KVStore: RingCell (prev, next)”. If the operation is write we remove the <key, value> from dirtyset.

Abort in KVClient is similar to commit. Just tell the KVStores to abort.

Class Log

| operation: Operation | commitOrAbort: Int<br>1 for commit, 0 for abort |
|---|---|

At the KVStore side, if the message is Commit (tranID), we hash the tranID to get the corresponding operation in the log list. If the operation is read, get the value from store; else, put the <key, value> into store and print “tranID: XXX put [key, RingCell (prev, next)]”. We set commitOrAbort to 1 under that operation in the log. Then we release the lock and send the requested data to sender.

If the message is Abort(tranID): if the KVStore has voted yes, release the lock and set commitOrAbort to 0 under that operation in the log.

Running result:

```
tranID: 213603474980295952515054342301652494929 put [883398409736703597245367459396791102l8, RingCell(56, 57)]
KVStore: RingCell (30, 31)
Cache: RingCell (30, 31)
Cache: RingCell (65, 66)
Cache: RingCell (30, 31)
KVStore: RingCell (0, 1)
tranID: 140433306340751780352864644416044012707 put [213603474980295952515054342301652494929, RingCell(83, 84)]
tranID: 883398409736703597245367459396791102l8 put [259315330308333885067857420279572716123, RingCell(61, 62)]
KVStore: RingCell (0, 1)
tranID: 140433306340751780352864644416044012707 put [883398409736703597245367459396791102l8, RingCell(56, 57)]
KVStore: RingCell (56, 57)
Cache: RingCell (0, 1)
Cache: RingCell (30, 31)
Cache: RingCell (23, 24)
tranID: 883398409736703597245367459396791102l8 put [883398409736703597245367459396791102l8, RingCell(25, 26)]
Cache: RingCell (23, 24)
KVStore: RingCell (56, 57)
tranID: 17939588594489115916697638763245438334 2 put [232815393282890072006820722084635286104, RingCell(99, 100)]
KVStore: RingCell (0, 1)
KVStore: RingCell (0, 1)
Cache: RingCell (0, 1)
Cache: RingCell (0, 1)
tranID: 17939588594489115916697638763245438334 2 put [176794910650439334945613785118339304277, RingCell(14, 15)]
Cache: RingCell (23, 24)
Cache: RingCell (18, 19)
KVStore: RingCell (14, 15)
tranID: 17939588594489115916697638763245438334 2 put [298485851460053443693657716239517591112, RingCell(56, 57)]
tranID: 27682144145127518529757756830641460431 put [883398409736703597245367459396791102l8, RingCell(56, 57)]
Cache: RingCell (69, 70)
KVStore: RingCell (99, 100)
tranID: 17939588594489115916697638763245438334 2 put [259315330308333885067857420279572716123, RingCell(61, 62)]
```