Team5 Assignment5 Report2

107062313 黃寶萱　107062117 李采蓉　107062317 陳怡汝

◈ Prepare Keys

➔ StoredProcedure :

在拿到一個 stored procedure 時會先呼叫 prepare()，助教的做法在這邊會呼叫 prepareKeys()，紀錄該 Tx 需要哪些 lock，之後才呼叫 scheduleTransactionSerially()創建新的 Tx。

```
public void prepare(Object... pars) {
    // prepare parameters
    paramHelper.prepareParameters(pars);

    // Collect read/write sets
    prepareKeys();

    // create a transaction
    boolean isReadOnly = paramHelper.isReadOnly();
    tx = scheduleTransactionSerially(isReadOnly, readSet, writeSet);

}
```

➔ MicroTxnProc :

在 create 新的 Tx 之前先呼叫 prepareKeys()，紀錄該 Tx 執行時需要拿到那些 lock，分別存在 readSet 以及 writeSet 中。

```
@Override
protected void prepareKeys() {
    MicroTxnProcParamHelper paramHelper = getParamHelper();
    for (int i = 0; i < paramHelper.getReadCount(); i++) {
        Map<String, Constant> keyEntryMap = new HashMap<String, Constant>();
        keyEntryMap.put("i_id", new IntegerConstant(paramHelper.getReadItemId(i)));
        readSet.add(new PrimaryKey("item", keyEntryMap));
    }
    for (int i = 0; i < paramHelper.getWriteCount(); i++) {
        Map<String, Constant> keyEntryMap = new HashMap<String, Constant>();
        keyEntryMap.put("i_id", new IntegerConstant(paramHelper.getWriteItemId(i)));
        writeSet.add(new PrimaryKey("item", keyEntryMap));
    }
}
```

◈ Create Tx and ccMgr

➔ StoredProcedure :

呼叫 scheduleTransactionSerially()創建新的 Tx。

```
public void prepare(Object... pars) {
    // prepare parameters
    paramHelper.prepareParameters(pars);

    // Collect read/write sets
    prepareKeys();

    // create a transaction
    boolean isReadOnly = paramHelper.isReadOnly();
    tx = scheduleTransactionSerially(isReadOnly, readSet, writeSet);

}
```

➔ MicroTxnProc :

在 scheduleTransactionSerially()中利用一個 ReentrantLock object 控制一次只能創建一個 Tx 與 ConservativeConcurrencyMgr，並進行 bookReadKeys 與 bookWriteKeys 的動作，將該 Tx 需要的 read/write lock 分別存在 HashSet readObjs、writeObjs 中，bookedObjs 則記錄所有已經 lock 的 object (read+write)。

```
private Transaction scheduleTransactionSerially(boolean isReadOnly,
        Set<PrimaryKey> readSet, Set<PrimaryKey> writeSet) {
    SERIAL_CONTROL_LOCK.lock();
    try {
        Transaction tx = VanillaDb.txMgr().newTransaction(
                Connection.TRANSACTION_SERIALIZABLE, isReadOnly);

        ConservativeConcurrencyMgr ccMgr = (ConservativeConcurrencyMgr) tx.concurrencyMgr();

        // Reserve lock so that deterministic ordering is ensured
        ccMgr.bookReadKeys(readSet);
        ccMgr.bookWriteKeys(writeSet);

        return tx;
    } finally {
        SERIAL_CONTROL_LOCK.unlock();
    }
}
```

◈ Acquire locks before execution

➔ StoredProcedure :

在執行 executeSql()前呼叫 function acquireBookedLocks()拿取 xLock 與 sLock。

```
public SpResultSet execute() {
    boolean isCommitted = false;

    try {
        ConservativeConcurrencyMgr ccMgr = (ConservativeConcurrencyMgr) tx.concurrencyMgr();

        // Acquire locks before execution
        ccMgr.acquireBookedLocks();

        executeSql();

        // The transaction finishes normally
        tx.commit();
        isCommitted = true;
```

➔ ConservativeConcurrencyMgr

aquireBookedLocks()根據 readObjs 與 writeObjs 內容分別拿取 xLock 與 sLock。

```
public void acquireBookedLocks() {
    bookedObjs.clear();

    for (Object obj : writeObjs)
        lockTbl.xLock(obj, txNum);

    for (Object obj : readObjs)
        if (!writeObjs.contains(obj))
            lockTbl.sLock(obj, txNum);
}
```

◈ ConservativeConcurrencyMgr

■ bookReadKey()、bookWriteKeys()、bookWriteKey()、bookWriteKeys()

根據傳入的參數將該 Tx 需要的 read/write lock 加到 readObjs/writeObjs 中。

bookedObjs 記錄所有已經 lock 的 object。

```
/**
 * Book the read lock of the specified objects.
 *
 * @param keys
 *            the objects which the transaction intends to read
 */
public void bookReadKeys(Collection<PrimaryKey> keys) {
    if (keys != null) {
        for (PrimaryKey key : keys) {
            // The key needs to be booked only once.
            if (!bookedObjs.contains(key))
                lockTbl.requestLock(key, txNum);
        }

        bookedObjs.addAll(keys);
        readObjs.addAll(keys);
    }
}
```

- aquireBookedLocks()根據 readObjs 與 writeObjs 內容分別拿取 xLock 與 sLock。

```
public void acquireBookedLocks() {
    bookedObjs.clear();

    for (Object obj : writeObjs)
        lockTbl.xLock(obj, txNum);

    for (Object obj : readObjs)
        if (!writeObjs.contains(obj))
            lockTbl.sLock(obj, txNum);
}
```

- modifyLeafBlock(blk)、readLeafBlock(blk) : block-level

  在 B-Tree index 中會使用到這兩個 function，拿取該 blk 的 xLock/sLock，並用 writtenIndexBlks/readIndexBlks 紀錄拿了哪些 blk 的 lock。

```
public void modifyLeafBlock(BlockId blk) {
    lockTbl.xLock(blk, txNum);
    writtenIndexBlks.add(blk);
}
```

◈ 與我們做法的比較

- 我們的做法是在 executeSql()前呼叫 staticSql()確定該 Tx 是否可以拿到所有需要的 lock，確定拿到所有的 lock 就將需要的 lock 鎖起來。而助教的作法是先在 prepare()前呼叫 prepareKeys()紀錄該 Tx 需要哪些 lock 並在 executeSql()前呼叫 acquireBookedLocks()拿取 xLock 與 sLock。我們在 execute()中的 staticSql()一次將檢查 lock 跟鎖 lock 兩個步驟做完，而助教是分開處理。雖然兩個功能都有完成，但我們認為助教的做法比較正確，因為兩個步驟發生的時間點較為準確，應該要在 Tx 創建之前確認 lock 的數量，在執行的時候才真正將 lock 鎖上。

- 我們的做法是在 transactionMgr 利用 doCheckTxs 與 canDoTxs 紀錄哪些 Tx 可以做/不可以做，在執行 staticSql()時將確定拿到 lock 跟沒拿到 lock 的 Tx 並分開儲存在兩個 sets，其中確定拿到 lock 以後，才會記錄拿取哪些 lock。而助教是利用 LockTable 的特性，不能執行的 Tx 會因為拿不到全部所需的 lock 而不能 execute()。我們認為助教的作法較為正確，使用兩個 sets 來維護資訊不太容易，還需要考慮重新檢查 Tx 的時間，若能妥善利用 LockTable 的特性將能更有效的管理 Txs，這是我們沒有考慮到的。

- 我們的做法是利用 logical table 紀錄哪些 lock 可以拿，是另外開 HashMap 來管理用到的 Records，並記錄 Read/Write 的次數。由於我們在實作 LockTable 時沒有想像中的順利，因此無法藉由 DB 中的 LockTable 機制來管理 Txs，而助教的方式是利用 DB 中的 LockTable 對 block 做 sLock/xLock。我們的方法是用另外的方法將 record 鎖住以達到 DB 中 LockTable 的效果，雖然可以盡可能地完成作業的要求，但是我們的作法還有很多的不足，因此我們認為助教的做法比較好，我們也會藉此機會學習如何正確地使用 DB 原有的資源做修正和優化。

◈ PrimaryKey

- 助教使用 PrimaryKey 表示 record 並記錄在 read/writeSet 中。在 PrimaryKey 裡助教紀錄了 table name 以及 record 的 i_id，並且使用了一些質數對兩者的 hashCode 進行計算產生可以唯一表示該 record 的 PrimaryKey。

```
public PrimaryKey(String tableName, Map<String, Constant> keyEntryMap) {
    this.tableName = tableName;
    this.keyEntryMap = keyEntryMap;

    genHashCode();
}
```

```
private void genHashCode() {
    hashCode = 17;
    hashCode = 31 * hashCode + tableName.hashCode();
    hashCode = 31 * hashCode + keyEntryMap.hashCode();
}
```

- 我們紀錄 transaction read/writeSet 的方式是用 i_id 直接表示，與助教的差別在於沒有引入目前是哪一個 table 的資訊，在計算上因為不用另外計算 hashCode 所以有微小的加速，在本次作業中，因為只用使用到 item 這一個 table，所以這樣的寫法也是可以的，但是在應用上因為我們的寫法沒有引入 table 的資訊，在需要同時 access 到不同 table 的時候就不能這樣用，然而使用者常會需要 access 到一個以上的 table，因此我們認為助教的寫法在實際使用上更符合 database 的特性，是更好的寫法。
  - 我們在 Transaction 裡面記錄一個 RecordTypeMap，以 i_id 為 key、type (“r”或 “w”) 為 value 的形式紀錄 transaction 的 read/writeSet。

```
public HashMap<Long, String> RecordTypeMap = new HashMap<Long, String>();
```