As5 – Implement 2V2PL

組員：
謝易軒 103020007
王偉 1062037s

一、工作項目:

根據 2V2PL 協定中，一個 tx 要去 modify a record 必須循著一定程序，我會先簡介 2V2PL 的理論概念，然後以下我將依照此程序去設計的工作項目條列出來。 在 2V2PL，當一個 tx 要去修改一個 block 中的某個紀錄時，他只能先在 private workspace 修改，這部分是為了提高 write 時的 read concurrency。為了在當前 tx 執行修改時，能確保其他 tx_i 不會讀到 tx 所寫入的資料，並且 tx 在寫入後讀取同一條紀錄時要能讀到剛寫入的資料。我們的基本設計想法即是讓每個 tx 都保有一個獨立的 workspace，簡單來說，該 tx 要先獲取欲修改的 Record 的 shadow exclusive lock(shxlock)，然後將這塊資料寫入 private workspace(hashmap)中，等到最後 tx 要 commit 前，才去獲取 xlock，將資料 in-place write 回到 buffer 再到 disk 中。

1. Q:如何設計 private workspace?
   我的理解上，private workspace 基本上就是分配給每一個 tx 的快取，做為該 tx 去做 read 或是 write 時所存取資料塊的 buffer。而設計 private workspace 必須同時考慮 shadow write 與 in-place write 的需求，最主要的考量還是 in-place write，因為最後寫回 disk through buffer，必須知道所存資料，要寫回哪個位置(top down from file, blk, record to field)，要由哪一個階層來負責做這項工作，因此，我將在第三點詳述。
2. Q:如何設計 lock compatibilty?
   我實做的部分，主要考慮了以下幾點子問題:何時可以獲取 shxlock? shadow_modifyRecord()時，multi-granularity locking 邏輯是否正確? avoidDeadlock()如何設計? tx 提交時如何把鎖放掉?
3. Q:如何換鎖(from shxlock to xlock)?
   在 tx commit 時，從 recordFiles 中抓出的 recordfile 會負責做 in-place write。當 rf 做 setval 時，他會 top down 的去呼叫到 recordpage 做 setval，我在此即沿用原本程式獲取 xlock 的過程，即呼叫 tx.concurrencyMgr.modifyRecord()；相應地 shxlock 的獲取上，我是在 record page 中新設立一個方法 shadow_setval()來做 shadow write，其下 tx.concurrencyMgr.shadow_modifyRecord()會去獲取 shxlock。
   Shadow_setVal() @RecordPage (for shadow write)

```
public void shadow_setVal(String fldname, int offset, Constant val) {
    if (tx.isReadOnly() && !isTempTable())
        throw new UnsupportedOperationException();

    RecordId rid = new RecordId(blk, currentSlot);
     //how to deal w/ conflict in hashmap? <1.key, 2.val>
    String fieldname_rid = "fieldname:"+fldname+"rid:"+rid;
    Transaction.fld_rid_val value = tx.new fld_rid_val(fldname, rid, val);


    //Constant c = tx.hashmap_get(fieldname_rid).constant;
    if (tx.hashmap_get(fieldname_rid)==null) {
        //get shxlock at first
        if (!isTempTable())
            tx.concurrencyMgr().shadow_modifyRecord(new RecordId(blk, currentSlot));
        tx.hashmap_put(fieldname_rid, value);
    }
    else
        //update entry (w/ shxlock on hand)
        tx.hashmap_put(fieldname_rid, value);
}
```

setVal() @RecordPage (for in-place write)

```
private void setVal(int offset, Constant val) {
    if (tx.isReadOnly() && !isTempTable())
        throw new UnsupportedOperationException();
    //get lock
    if (!isTempTable())
        tx.concurrencyMgr().modifyRecord(new RecordId(blk, currentSlot));
    //set value
    LogSeqNum lsn = doLog ? tx.recoveryMgr().logSetVal(currentBuff, offset, val)
            : null;
    currentBuff.setVal(offset, val, tx.getTransactionNumber(), lsn);
}
```

4. Q:如何做 inplace write?
   Tx commit 時，由誰來真正將東西寫進去? Recordfile->setval() RecordPage，
   這部分該如何設計? 考慮原本的程式設計，是不是要從最上層寫下來呢?
   我的答案是:每一個更新，基本上就涉及一個資料塊，如果你沒有 record page 中的 currentbuffer 你根本寫不進去。所以你最好需要一個 recordfile，所以剩下的問題就是 recordfile 存取的基本規則是什麼?一個 tx 中所有涉及的資料塊都可以再一個 recordfile 中拿取嗎?還是需要多個 recordfile?
   根據以上種種子問題，做 in-place write 基本上必須考慮以下兩點:第一:要考慮寫入位置，可不可以看當前 tx 對哪一些資料塊拿有 shxlock，然後據此從 private workspace 提取資料，最後把他寫回去，同時，一個 recordfile 中只存放一個 table 的資料(homo- record file 的特質)，但是一個 tx 可能會用到多個 tables，這個又要怎麼處理呢?。第二:要考慮如何釋放 shxlock，然後去拿 xlock。Filename>rid>fldname>
   我最後在 tx 裡面，做了兩個 hashmap<key, value> 其一(hashmap)根據“fldname+rid” 來確立 fld_rid_val；另一者 (recordFiles) 利用“filename”確立 recordfile。第一個 hashmap 得做法，目的在於做

tx.commit()前，要先做 in-place write，此時我們必須遍歷 hashmap 所有的 entry，然後將 val，循著 file>rid>fldname 這個階層寫入，所以我設計了一個類(fld_rid_val)，其中包含所存資料塊的(value, fieldname, recordid)，除此之外，每一個 entry 還有另一個意義，即這一個 fld_rid_val 是被當前 tx 所 shxlocked。第二個 hashmap:recordFiles，主要的功能在於，記錄負責做 in-place write 工作的 recordFile，實際運作上只要提取 rid.block 即可知道 filename 然後找出要負責 in_place write 的 recordfile!

commit() @Transaction

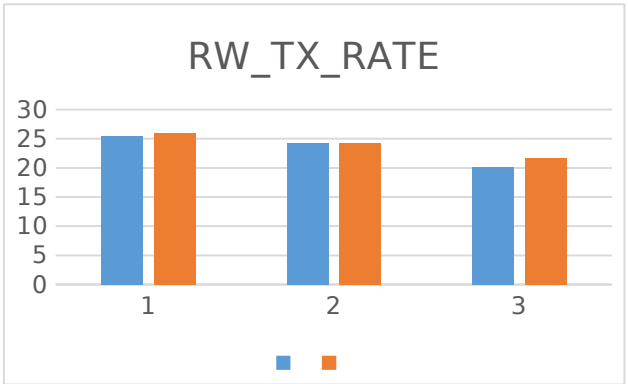
```
public void commit() {
    //in-place write
        if(!this.isReadOnly() || this.isReadOnly()) {
            //record file correspond to the rid should do the in-place write
            for (Map.Entry<String, fld_rid_val> entry : this.hashmap.entrySet()) {
                fld_rid_val val = entry.getValue();
                RecordFile rf = this.recordFiles_get(val.rid.block().fileName());
                rf.moveToRecordId(val.rid);
                rf.in_place_setVal(val.fldname, val.constant);
            }
        }
```

二、實驗數據:未關閉寫入快取，比較 throughput(單位:K)

0. 實驗環境: Intel Core i3-4030U CPU @ 1.9GHz, 4 GB RAM, 16 GB SSD, Windows

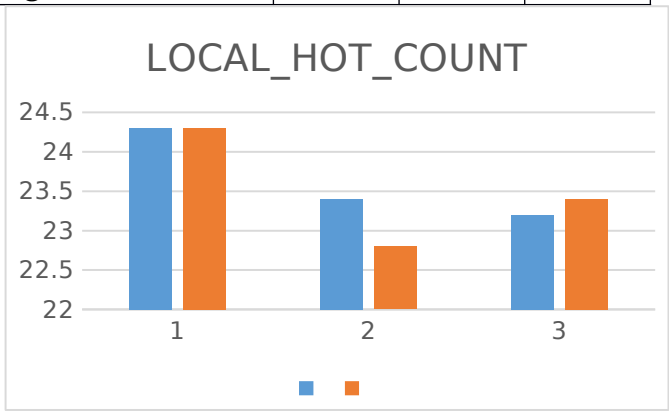
1.

RW_TX_RATE	0.25	0.5	0.75
2v2pl	25.4	24.3	20.1
origin	25.9	24.3	21.6



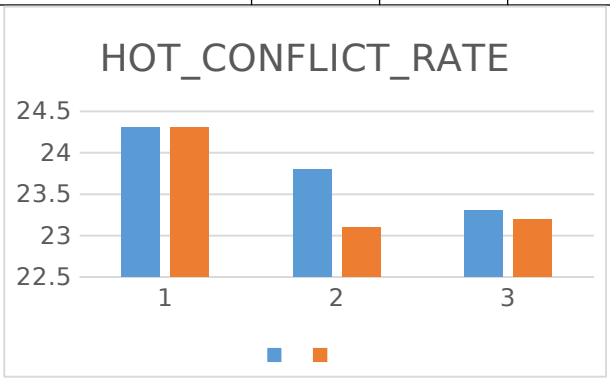
2.

LOCAL_HOT_COUNT	1	3	5
2v2pl	24.3	23.4	23.2
origin	24.3	22.8	23.4



3.

HOT_CONFLICT_RATE	0.001	0.005	0.01
2v2pl	24.3	23.8	23.3
origin	24.3	23.1	23.2



實驗分析

1. 為何在使用 2V2PL **RW_TX_RATE** 調高後，throughput 沒有顯著的提昇?

本來預期在 implementf 2V2PL，throughput 應該會隨著 write transaction 的比例提高，2v2pl 的 throughput 會相較於 2pl 有相當大的提昇，然而觀察實驗項目 1 的結果，兩者的 performance 竟然差不多的。
因此，我們認為應該是 private workspace 中使用的 HashMap 在 transaction 要 commit 時，需要用 iterate 完所有“fldname+rid”所至，因為複雜度會隨著要想入的檔案量鎖提高，另外就是 shx lock 轉成 x lock 鎖產

生的 ovrehead

2. LOCAL_HOT_COUNT 提高時，為何使用 2V2PL 會比 2PL 有更高的 throughput 呢?

這部份是符合我們的預期，因 hot record 是被讀寫的次數非常頻繁，因此 2V2PL 的特點"writing 不會 阻礙 read“的特點被充分的展現出來，由於 shadow wring 的 shx lock 是與 reading 的 s lock 可以相容，因此即使是在短的時間同一筆 record 被正在拿著 shx 的 transaction 寫，別的 record 也能夠讀，使 throughput 能夠提昇。

3. HOT_CONFLICT_RATE，為何使用 2V2PL 會比 2PL 有更高的 throughput 呢?

這部份也是符合我們的預期，因為每個 transaction 即使在 access 同一筆 record，都會先存在自己的 private workspace 因此在 commit 前，都不會產生相撞的問題。transaction 同時 commit 的比例相當的低，因此比起 2PL 只能把 recordfile 鎖起，只讓其他 transaction 等待。2V2PL 顯然可以有更大的 throughput。