Simple DB Lab4

Git 地址：https://github.com/Facwl/SimpleDB

（我用 chorme 能打开这个 github 链接，用 edge 就显示 404，不明原理）

一、 代码改动

因为 *getPage()* 中调用的系统函数 （如 Thread.sleep）以及一些功能实现的需要 在几乎所有的 Test 里面都添加了 *IOException* 和 *InterruptedException* ； 不添加，会导致大面积报错。除此之外没有对测试的功能进行任何改动，建议直接使用提交的 Test 测试。

二、 设计思路

1. ConcurrentHashMap 的使用

首先，理解锁的两种类型。因为允许一个页面同时被多个请求共享锁的事务占有，因此，一个页面 lock 的记录应该设置为数组。

```
ConcurrentHashMap<PageId, List<LockStru>> stateRecord; 储存占据每一页的事务
```


```
ConcurrentHashMap<TransactionId,PageId> waitList;
储存等待序列（用于基于依赖图的死锁判定）
```

每当一个事务请求失败，就把它和它请求的 pid 存进这里，等到以后它成功请求到

锁时，再移出去。

2. Release 和 HoldsLock

均通过查找 ***stateRecord*** 实现，检查“有”和 进一步的 remove

3. Commit 和 Abort

正好对应两种操作，一种是写入缓冲区，一种是从磁盘回滚版本到缓冲区，都需要 markDirty

全部 required 的测试应该都能通过

三、 重难点

1. 基于 dependency graphs 的死锁检测

思路：分为直接死锁和间接死锁；间接死锁明显是一层一层推进的，使用递归实现。

```
public synchronized boolean isDeadLock(PageId pid,TransactionId tid)
{
    //System.out.println("dd?");
    List<TransactionId> TestedTids=new ArrayList<>();
    TestedTids.add(tid);
    //找到现在占用的Pid的资源，看他是否需要tid现在占用的资源

    List<LockStru> lcLst=stateRecord.get(pid);  //获取现在霸占着pid的事务

    if(lcLst==null||lcLst.size()==0)     //没有最好
    {
        return false;
    }

    List<PageId> tidRs=new ArrayList<>();       //tid占有的page们

    for (Map.Entry<PageId, List<LockStru>> entry : stateRecord.entrySet()) {
        for (LockStru ls : entry.getValue()) {
            if (ls.tid==tid) {
                tidRs.add(entry.getKey());
            }
        }
    }

    /*******************************/
    for(LockStru ls:lcLst)       //看现在拿着pid的tid们有没有间接或者直接等待 tisRs 的
    {
        TransactionId curHolder=ls.tid;
        if(curHolder!=tid) {
            boolean isWaiting=waitSrc(curHolder,tidRs,TestedTids);
            if(isWaiting)
                return true;
        }
```

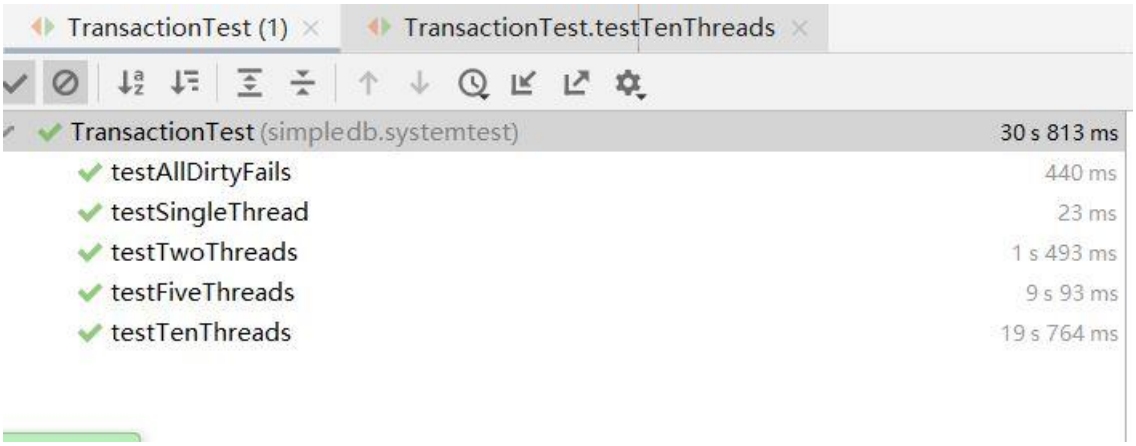
IsDeadLock **入口函数**

waitSrc 递归入口

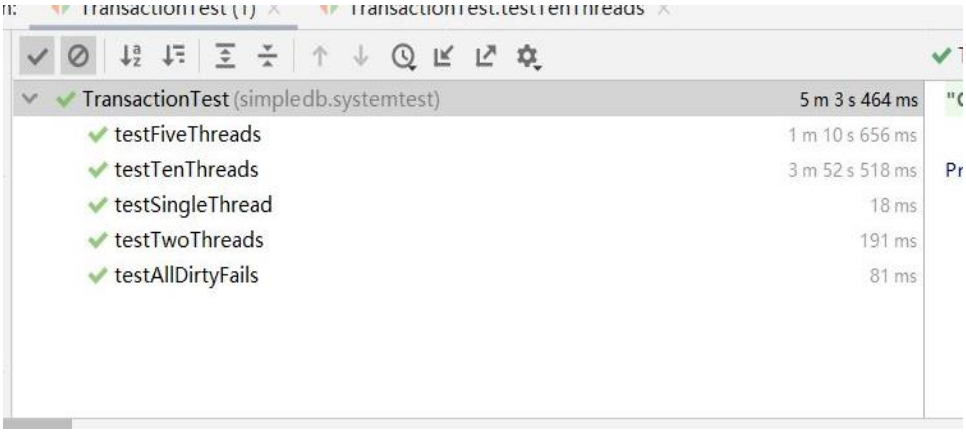
```java
private synchronized boolean waitSrc(TransactionId curHolder, List<PageId> curSrc, List<TransactionId> TestedTids) {
    //来判断等待
    PageId waitPg = waitList.get(curHolder);//这页事务正在等待的 资源pid
    if (waitPg == null) {   //如果这个tid什么资源都没有在等待 false
        return false;

    }
    for (PageId tmpPid : curSrc) {

        if (tmpPid == waitPg)
        {
            return true;    //直接等待
        }
    }
    //检测间接等待
    List<LockStru> holders=stateRecord.get(waitPg);   //拥有现在这个事务tid等待的pid的事务们

    if(holders==null||holders.size()==0)
    {
        return false;//没有 就没有
    }
    for (LockStru pls : holders) {      //否则，遍历这些事务
        TransactionId holder = pls.tid;
        if (!TestedTids.contains(holder)) {     //如果有新的没有被测试的事务tid,添加进已经在测试的事务（tested） 进入下一层
            TestedTids.add(holder);             //否则，不进行处理，防止无限递归
            boolean isWaiting = waitSrc(holder, curSrc, TestedTids);
                if (isWaiting)
                {
                    return true;
                }
            }
        }
    return false;
}
```

四、 思考题

1. Deadlock detection: timeouts versus dependency graphs



基于 timeouts 的运行时间：



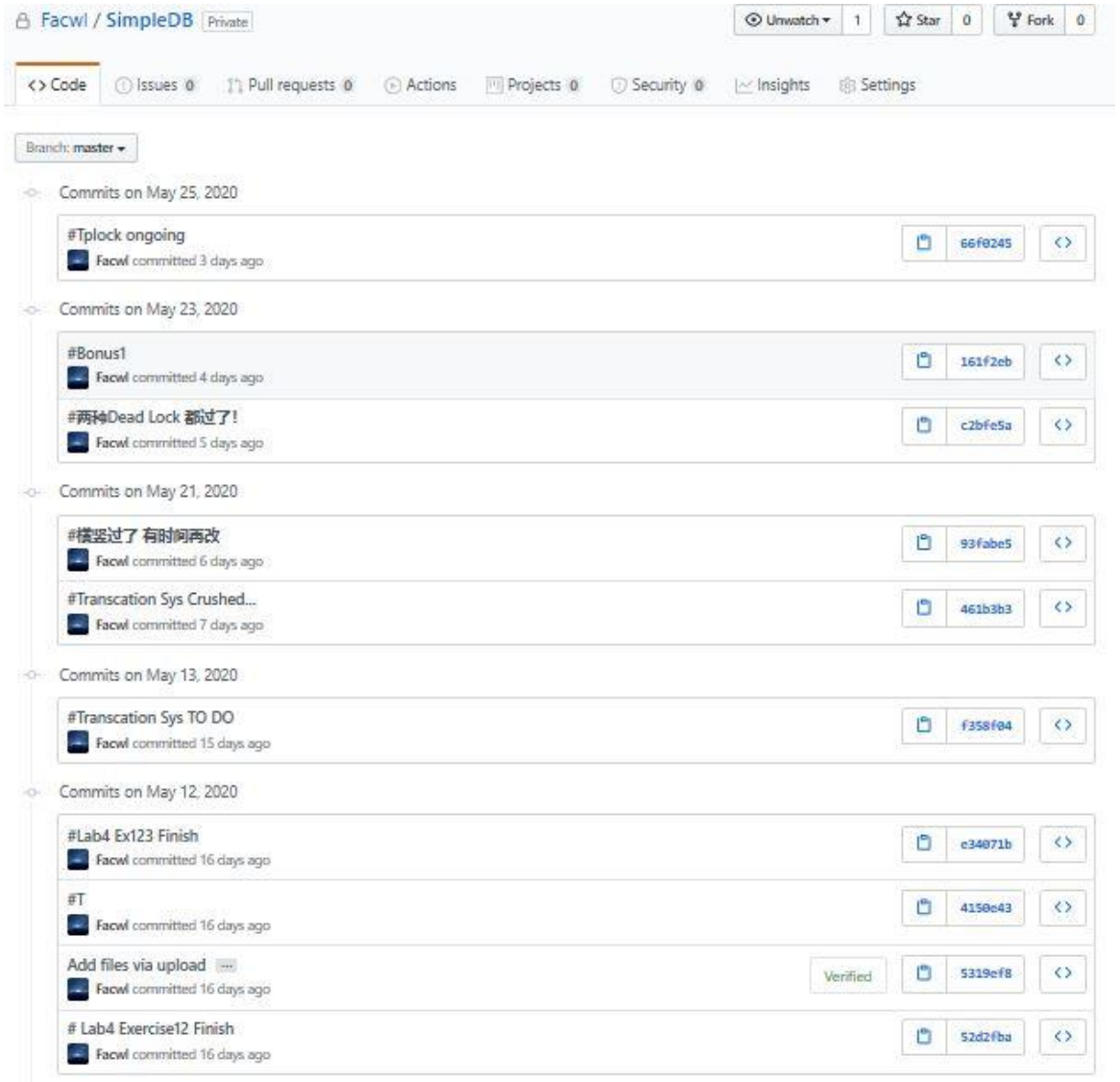
基于 graph 的运行时间

可以看出，timeout 策略明显好于依赖图。这里猜测原因是，递归不停的调用函数进行堆栈操作本身占用一部分时间，每次还要遍历 locklist，比较费时。

Graph 因为是依靠逻辑的，稳定性要高一些，比如 hashmap 在实际运行中变得特别大，使得正常查询就已经超过阈值，那就会抛出不必要的错误，还要调参。

五、 提交历史记录

特别说明：因为我有一次想版本回滚但不太会操作，早期提交中可能有显示所有文件都重新提交一遍的记录，那是因为我换了个本地储存仓库，以错误的方式强行提交了一次，请忽略。


Facwl / SimpleDB Private
Unwatch 1
Star 0
Fork 0
Code
Issues 0
Pull requests 0
Actions
Projects 0
Security 0
Insights
Settings
Branch: master
Commits on May 25, 2020
#Tplock ongoing
Facwl committed 3 days ago
66f0245
Commits on May 23, 2020
#Bonus1
Facwl committed 4 days ago
161f2eb
#两种Dead Lock 都过了!
Facwl committed 5 days ago
c2bfe5a
Commits on May 21, 2020
#模竖过了 有时间再改
Facwl committed 6 days ago
93fabe5
#Transcation Sys Crushed...
Facwl committed 7 days ago
461b3b3
Commits on May 13, 2020
#Transcation Sys TO DO
Facwl committed 15 days ago
f358f04
Commits on May 12, 2020
#Lab4 Ex123 Finish
Facwl committed 16 days ago
e34071b
#T
Facwl committed 16 days ago
4150e43
Add files via upload
Facwl committed 16 days ago
Verified
5319ef8
Lab4 Exercise12 Finish
Facwl committed 16 days ago
52d2fba