Conor Reisman

5/13/2013

CSE444

Lab 3

Design Decisions

I used page-based locking for simplicity and acquisition speed.

I used timeouts to detect deadlocks.

I aborted the timed-out thread instead of the blocking thread(s).

I implemented my own lock in order to complete this lab. This was necessary since ReentrantReadWriteLocks are thread-based and you cannot upgrade from read to write. My implementation, UpgradeableReentrantReadWriteLock, uses TransactionId-based locking and allows transactions to upgrade from read to write.

I used a ReentrantLock with two Conditions to implement my lock. This allowed me to prevent concurrent modification of the lock, while also allowing threads to safely block while waiting for the lock to open.

It also optionally allows range timeouts. This was because if I broadcasted a signal to all waiting threads, they'd all wake up and then go back to sleep very close together. They'd then all timeout at the same time and all abort. With staggered timeouts, one would timeout, abort, release its locks, and potentially allow others to continue. I chose to set a minimum timeout at 100ms and a maximum at 500ms.

API Changes

The evictPage documentation said not to evict DIRTY pages. This allows for the possibility of an operator getting a page using getPage(), having the page evicted, then editing the page and trying to commit. The operator would still have access to the page, but it wouldn't be in the BufferPool, so when it tried to commit, the BufferPool wouldn't find it to flush it to disk. To prevent this, I changed it so that evictPage will not evict pages where some transaction holds a write lock. It's possible they hold the write lock without the intent to write, but it's better to be safe than sorry.

Incomplete/Missing Implementation

Nothing that was required for lab 3 was left unimplemented.

Other

- The locking was a HUGE problem. Apparently you assumed we could use ReentrantReadWriteLock. There are several reasons you should have seen it wouldn't work. For example, the javadoc EXPLICITLY states that it is not upgradeable. Additionally, it works on the THREAD level. Yet you wanted us to do locking on the TRANSACTION level. You all showed a lack of knowledge about how the tests worked and what the invariant for the relationship between threads and transactions was. This then required us to make our own locks, when students

aren’t required to have taken any classes that actually teach locks and you provided no instruction either. It was an incredibly unfair assignment. You should be playtesting all of our assignments before assigning them.

- See other problems in notes.txt.