ACS Assignment 2

Svend Olaf V. Mikkelsen

Tuesday 1st December, 2015 23:53

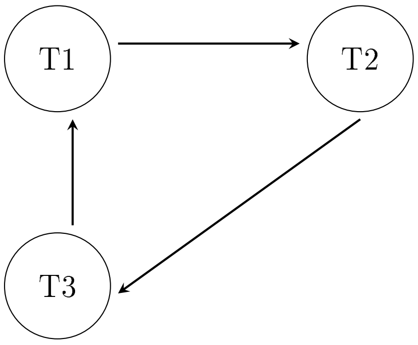
Exercises

Question 1: Serializability & Locking

Schedule 1

```
T1: R(X)                                                W(Y)    C
T2:         W(Z)    W(X)    C
T3:                                 R(Z)    R(Y)    C
```

The precedence graph which contains a node for each committed transaction in the schedule, and an arc from Ti to Tj if an action of Ti precedes and conflicts with one of Tj's actions is:



From the book page 76 we know that a schedule is conflict serializable if and only if its precedence graph is acyclic. Since the graph is cyclic, schedule 1 is not conflict serializable.

We also know that strict 2PL ensures that the precedence graph for any schedule that it allows is acyclic.

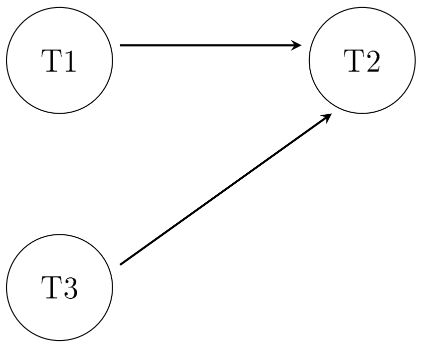
By modus tollens we then know that the schedule could not have been generated by strict 2PL.

Schedule 2

```
T1: R(X)                                W(Y)    C
T2:                             R(Z)                        W(X)    W(Y)    C
```

```
T3:         W(Z)    C
```

The precedence graph, which contains a node for each committed transaction in the schedule, and an arc from Ti to Tj if an action of Ti precedes and conflicts with one of Tj's actions:



We attempt to apply locks according to strict 2PL. Locks are shown with lower case letter for shared lock and upper case letter for exclusive lock:

```
T1: R(X)                            W(Y)    C
    xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                                    YYYYYYYYY
T2:                         R(Z)                     W(X)    W(Y)    C
                            zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz
                                                     XXXXXXXXXXXXXXXXX
                                                             YYYYYYYYY

T3:         W(Z)    C
            ZZZZZZZZZ
```

Since there are no conflicts the schedule could have been generated by a scheduler using strict 2PL, as shown.

Question 2: Optimistic Concurrency Control

For optimistic concurrency control one of the following validation conditions must hold to validate a transaction (book page 91):

1. Ti completes (all three phases) before Tj begins.
2. Ti completes before Tj starts its write phase, and Ti does not write any database object read by Tj.
3. Ti completes its read phase before Tj completes is read phase, and Ti does not write any database object that is either read or written by Tj.

In the tables unknown values are shown as x.

Scenario 1

```
T1: RS(T1) = {1, 2, 3}, WS(T1) = {3},
  T1 completes before T3 starts.
```

```
T2: RS(T2) = {2, 3, 4}, WS(T2) = {4, 5},
  T2 completes before T3 begins with its write phase.
T3: RS(T3) = {3, 4, 6}, WS(T3) = {3},
  allow commit or roll back?
```

			T1/T3		T2/T3
1	Ti completes before Tj begins.		1		x
2a	Ti completes before Tj writes	1		1	
2b	Ti does not write items read by Tj	0		0	
2	2a and 2b		0		0
3a	Ti completes reads before Tj complete reads	1		x	
3b	Ti does not write items used by Tj	0		0	
3	3a and 3b		0		0

Since we do not know if $T2$ completes before $T3$ begins, we will have to roll back $T3$.

Scenario 2

```
T1: RS(T1) = {1, 2, 3}, WS(T1) = {3},
  T1 completes before T3 begins with its write phase.
T2: RS(T2) = {5, 6, 7}, WS(T2) = {8},
  T2 completes read phase before T3 does.
T3: RS(T3) = {3, 4, 5, 6, 7}, WS(T3) = {3},
  allow commit or roll back?
```

			T1/T3		T2/T3
1	Ti completes before Tj begins.		x		x
2a	Ti completes before Tj writes	1		x	
2b	Ti does not write items read by Tj	0		1	
2	2a and 2b		0		x
3a	Ti completes reads before Tj complete reads	x		1	
3b	Ti does not write items used by Tj	0		1	
3	3a and 3b		0		1

Since we do not know if $T1$ completes before $T3$ begins, we must roll back.

Scenario 3

```
T1: RS(T1) = {2, 3, 4, 5}, WS(T1) = {4},
  T1 completes before T3 begins with its write phase.
T2: RS(T2) = {6, 7, 8}, WS(T2) = {6},
  T2 completes before T3 begins with its write phase.
T3: RS(T3) = {2, 3, 5, 7, 8}, WS(T3) = {7, 8},
  allow commit or roll back?
```

			T1/T3		T2/T3
1	Ti completes before Tj begins.		x		x
2a	Ti completes before Tj writes	1		1	
2b	Ti does not write items read by Tj	1		1	
2	2a and 2b		1		1
3a	Ti completes reads before Tj complete reads	x		x	
3b	Ti does not write items used by Tj	1		1	
3	3a and 3b		x		x

We can commit $T3$ since condition 2 is fulfilled for both $T1/T2$ and $T2/T3$.

Programming Task

A Concurrent Certain Bookstore

Implementation

Source code is in file src.zip.

Various methods are edited in the file:

```
acertainbookstore-2015.2.1/src/com/acertainbookstore/
     business/ConcurrentCertainBookStore.java
```

Methods test1, test2, test3, test4 are added in the file:

```
acertainbookstore-2015.2.1/src/com/acertainbookstore/
     client/tests/BookStoreTest.java
```

Questions for Discussion on the Concurrent Implementation of Bookstore

1

(a) One part of the strategy to achieve before-or-after atomicity is that the integrity of input data is verified before data structure updates begin. That way there is no need to roll back already written data if invalid input data is met. This is part of the handed out code and is not changed for this submission.

One lock is implemented for the entire data set. More processes can obtain read access to the data set, but only one process can obtain access to write to the data set at a time. When write access is given it is assured that no other processes read the data set at the same time.

The programming language used has a feature to implement these locks. The feature has an option to prevent that writers starve. That feature was used.

Concretely the bookstore application is implemented in Java and the ReadWriteLock interface was used as suggested in the assignment text. At the beginning of each method that will only read data a read lock is taken, and the lock is released at all exit points of the method, which can be at end or while handling exceptions.

For methods that will write, a write lock is taken.

(b) Four tests were made. Test 1 and test 2 were sketched in the assignment text. Test 3 and test 4 were made to test different concurrency aspects of the implementation. Test 3 attempts to show that processes that read data runs faster if they run concurrently. Test 4 attempt to show that readers do not starve writers.

1. One thread will buy a number of a specific book a number of times. Another thread will add the same number of books the same numbers of times. It it is verified that there is the same number of copies in stock afterwards. The thread that buys books count the number of successes since the book may be out of stock in some attempts.
2. One thread repeatedly buys a set a books and adds the same set of books. Another thread verifies that the numbers of copies in stock matches what is expected before or after each step in the other thread. The book set ought to contain more than one title, but currently contains only one title.
3. A large number of books are added to the bookstore. Then a number of threads retrieving the list of books a number of times is run sequentially and afterwards the same number of threads doing the same are run concurrently.

   There has been difficulties obtaining the expected result that concurrently would be faster. This could be due to programming errors, or that the threads for some other reasons are not divided between different processor cores, or due to some kind of overhead.
4. A number of books are added. A number of threads are started reading the book data to a list. A book is added from the main thread. The reading threads are expected to detect that a book was added and exit. The purpose of this test is to see that the reading threads would not starve the writing thread. This test fails for a currently unknown reason – no it succeeds.

2

The locking protocol is correct. It is equivalent with “Strict 2PL” with one data object. Since there is only one object, the predicate reads vulnerability is not present.

3

Deadlocks cannot occur since each thread only holds one lock.

4

We are asked if there is a bottleneck with respect to the number of clients in the bookstore after the implementation.

There is a bottleneck since only one client can write at a time.

The scalability if the concurrent bookstore is integrated with RPC's was not considered.

5

We should discuss the overhead being paid in the locking protocol in the implementation versus the degree of concurrency achieved.

Considering that the overhead in the solution chosen is close to minimal, and that some concurrency is expected since a fair amount of usage may be reading services, there should be a reasonable ratio between overhead and concurrency.