ECEN 757 | Homework 4

Muhammed U. Ersoy
M.S. ECEN Student
Texas A&M University
mue@tamu.edu

Problem 16.4

The operation create inserts a new bank account at a branch. The transactions T and U are defined as follows:

```
T: aBranch.create("Z");
U: z.deposit(10); z.deposit(20).
```

Assume that Z does not yet exist. Assume also that the deposit operation does nothing if the account given as the argument does not exist. Consider the following interleaving of transactions T and U:

T	U
	z.deposit(10);
aBranch.create(Z);	
	z.deposit(20);

State the balance of Z after their execution in this order. Are these consistent with serially equivalent executions of T and U?

Answer 16.4

The balance of Z would be $\$20$. When U first executes deposit on Z, the branch doesn't exist thus the operation has no effect. Then the branch is created by T, and U deposits $\$20$.
These are not consistent with a serially equivalent operation. For example if we look at $T \rightarrow U$;

T	U
aBranch.create(Z);	
	z.deposit(10);
	z.deposit(20);

T	U
	z.deposit(10);
	z.deposit(20);
aBranch.create(Z);	

In the first case the branch is created, and both deposits go through. Our final state would have $\$30$ in the branch. In the other case of $U \rightarrow T$ the branch would not exist when deposits go through, and the final amount would be $\$0$. Thusfort the transaction outlined is not serially equivalent.

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

Problem 16.9

The transactions T and U at the server in Exercise 16.8 are defined as follows:

```
T: x = read(i); write(j, 44);
U: write(i, 55); write(j, 66);
```

Initial values of a_i and a_j are 10 and 20, respectively. Which of the following interleavings are serially equivalent, and which could occur with two-phase locking?

(a)

T	U
x=read(i);	
	write(i,55);
write(j,44);	
	write(j,66);

(b)

T	U
x=read(i);	
write(j,44);	
	write(i,55);
	write(j,66);

(c)

T	U
	write(i,55);
	write(j,66);
x=read(i);	
write(j,44);	

(d)

T	U
	write(i,55);
x=read(i);	
	write(j,66);
write(j,44);	

Answer 16.9

(b) and (c) are serially equivalent since in both cases one of the transactions (either U or T) accesses all necessary resources before the other one.
(a) and (d) could occur with two-phase locking since all necessary conflicting lock requests are in the same order.

Problem 16.10

Consider a relaxation of two-phase locks in which read-only transactions can release read locks early. Would a read-only transaction have consistent retrievals? Would the objects become inconsistent? Illustrate your answer with the following transactions T and U at the server in Exercise 16.8:

```
T: x = read (i); y= read(j);
U: write(i, 55); write(j, 66);
```

in which initial values of a_i and a_j are 10 and 20.

Answer 16.10

The reads could become inconsistent. If the process reads one of the values and releases the lock, another process could write, and if the original process performs a read on another value again, the state will become inconsistent.

T	U
x=read(i);	
//T releases lock	
	write(i, 55);
	write(j, 66);
y=read(j);	

In this case T read x, then released the lock prematurely. U wrote to both i, and j because there was no lock on them. T was not aware of this. Thus the values of x and y ended up being inconsistent.

Problem 16.16

Consider optimistic concurrency control as applied to the transactions T and U defined in Exercise 16.9. Suppose that transactions T and U are active at the same time as one another. Describe the outcome in each of the following cases:

i) T's request to commit comes first and backward validation is used.

ii) U's request to commit comes first and backward validation is used.

iii) T's request to commit comes first and forward validation is used.

iv) U's request to commit comes first and forward validation is used.

In each case describe the sequence in which the operations of T and U are performed, remembering that writes are not carried out until after validation.

Answer 16.16

i) In backwards validation, T's readset is compared with U's write set. Since U has not committed by this point all options are valid.

ii) In this case U has committed thus the transaction is **n**ot valid.

iii) T's write set is compared with the reads of with the reads of U. Since U has not reads this is valid. Then U's write set would be compared with other active reads. Since T was already committed there are no active requests. Thus this is valid.

iv) U's write set is compared with T's reads. Since there is an overlap this **f**ails.

Parts of this answer were taken from Kontogiannis, I was comparing my original answer and it seemed incomplete so I amended based on this solution manual.

Problem 16.19

Consider the use of timestamp ordering with each of the example interleavings of transactions T and U in Exercise 16.9. Initial values of a_i and a_j are 10 and 20, respectively, and initial read and write timestamps are t_0. Assume that each transaction opens and obtains a timestamp just before its first operation; for example, in (a) T and U get timestamps t_1 and t_2, respectively, where $t_0 < t_1 < t_2$. Describe in order of increasing time the effects of each operation of T and U. For each operation, state the following:

i) whether the operation may proceed according to the write or read rule;

ii) when timestamps are assigned to transactions or objects;

iii) when tentative objects are created and when their values are set.

a)
$a_i = 10$, $T_{write} = \max$, $T_{read} = t_0$
$a_j = 10$, $T_{write} = \max$, $T_{read} = t_0$

T: x= read(i);//timestamp = t_1
read$_rule$:t$_1$ > T_{write} on committed version (t_0) and $D_{selected}$ is committed:
$x = 10$ is read, and $\max T_{read}(a_i) = t_1$

U: write(i,55); //timestamp = t_2
write rule: $t_2 \geq \max T_{read} = t_1$ and t_2 > write timestamp on committed version.
allows write on tentative version $a_i = 55$. $T_{write} = t_2$

T: write(j,44); write rule: $t_1 \geq \max T_{read} = t_0$ and t_1 > write timestamp on committed version = t_0
allows tentative write $a_j = 44$; update timestamp of a_j to t_1

U: write(j,66);
write rule: $t_1 \geq \max T_{read} = t_0$ and t_2 > write timestamp on committed version = t_0
tentative version $a_j = 66$; timestamp = t_2

T commits first $\rightarrow$ $a_j = 44$ $T_{write} = t_1$, $T_{read} = t_0$
U commits second $\rightarrow$ $a_i = 55$ $T_{write} = t_2$, $T_{read} = t_1$
U commits second $\rightarrow$ $a_j = 66$ $T_{write} = t_2$, $T_{read} = t_0$

b)
T: x= read(i);//timestamp = t_1
read$_rule$:t$_1$ > T_{write} on committed version (t_0) and $D_{selected}$ is committed:
$x = 10$ is read, and $\max T_{read}(a_i) = t_1$

T: write(j,44); write rule: $t_1 \geq \max T_{read} = t_0$ and t_1 > write timestamp on committed version = t_0
allows tentative write $a_j = 44$; update timestamp of a_j to t_1

U: write(i,55); //timestamp = t_2
write rule: $t_2 \geq \max T_{read} = t_1$ and t_2 > write timestamp on committed version = t_0
allows write on tentative version $a_i = 55$. $T_{write} = t_2$

U: write(j,66);
write rule: $t_1 \geq \max T_{read} = t_0$ and t_2 > write timestamp on committed version = t_0
tentative version $a_j = 66$; timestamp = t_2

T commits first $\rightarrow$ $a_j = 44$ $T_{write} = t_1$, $T_{read} = t_0$
U commits second $\rightarrow$ $a_i = 55$ $T_{write} = t_2$, $T_{read} = t_1$
U commits second $\rightarrow$ $a_j = 66$ $T_{write} = t_2$, $T_{read} = t_0$

c)
U: write(i,55); //timestamp = t_2

write rule: $t_2 \geq \max T_{read} = t_1$ and t_2 > write timestamp on committed version = t_0
allows write on tentative version $a_i = 55$. $T_{write} = t_1$

U: write(j,66);
write rule: $t_1 \geq \max T_{read} = t_0$ and t_2 > write timestamp on committed version = t_0
tentative version $a_j = 66$; timestamp = t_1

T: x= read(i);//timestamp = t_2
read$_rule$:$t_1 > T_{write}$ on committed version (t_0) and $D_{selected}$ is committed:
Since U is not committed yet, T has to wait

U commits $\rightarrow a_i = 55$ $T_{write} = t_1$, $T_{read} = t_0$
U commits $\rightarrow a_j = 66$ $T_{write} = t_1$, $T_{read} = t_0$
T continues the read and $x = 55$

T: write(j,44); write rule: $t_2 \geq \max T_{read} = t_0$ and t_1 > write timestamp on committed version = t_0
allows tentative write $a_j = 44$; update timestamp of a_j to t_2

T commits second $\rightarrow a_j = 44$ $T_{write} = t_2$, $\max T_{read} = t_0$
Final state of $a_i = 55$, $T_{write} = t_1$, $\max T_{read} = t_2$

d)
U: write(i,55); //timestamp = t_2
write rule: $t_2 \geq \max T_{read} = t_1$ and t_2 > write timestamp on committed version = t_0
allows write on tentative version $a_i = 55$. $T_{write} = t_1$

T: x= read(i);//timestamp = t_2
read$_rule$:$t_1 > T_{write}$ on committed version (t_0) and $D_{selected}$ is committed:
Since U is not committed yet, T has to wait

U: write(j,66);
write rule: $t_1 \geq \max T_{read} = t_0$ and t_2 > write timestamp on committed version = t_0
tentative version $a_j = 66$; timestamp = t_1

U commits $\rightarrow a_i = 55$ $T_{write} = t_1$, $T_{read} = t_0$
U commits $\rightarrow a_j = 66$ $T_{write} = t_1$, $T_{read} = t_0$
T continues the read and $x = 55$

T: write(j,44); write rule: $t_2 \geq \max T_{read} = t_0$ and t_1 > write timestamp on committed version = t_0
allows tentative write $a_j = 44$; update timestamp of a_j to t_2

T commits second $\rightarrow a_j = 44$ $T_{write} = t_2$, $\max T_{read} = t_0$
Final state of $a_i = 55$, $T_{write} = t_1$, $\max T_{read} = t_2$

Parts of 16.19 were taken from Kontogiannis, My original answer was incomplete compared to this, I amended it.

References

Kostas Kontogiannis. URL `http://www.softlab.ntua.gr/~kkontog/ECE454/answers6-12-selected.PDF`.