ECE428-HW4

Shengjian Chen (3180111657)

QUESTION 1

(a) conflict

1/7 conflict on A; 5/4 conflict on B; 6/3 conflict on C; 2/9 conflict on D

(b) T1/T2

Yes, it is interleaving serially equivalent(T1, T2).

(c) T1/T2/T3

Yes, it is possible for both. (T3, T1)||for B; (T2, T3)||for D.

(d) values

A=5; B=5; C=6; D=1; E=3;

(e) serial

(T1, T2, T3): A=5; B=5; C=3; D=1; E=3;

(T1, T3, T2): A=2; B=5; C=3; D=1; E=0;

(T2, T1, T3): A=5; B=9; C=7; D=5; E=3;

(T2, T3, T1): A=5; B=5; C=10; D=1; E=3;

(T3, T1, T2): A=2; B=5; C=6; D=1; E=0;

(T3, T2, T1): A=2; B=5; C=7; D=1; E=0;

(f) tell

When we form the serially equivalent, the transactions should not be cross or cycle. This means that if we have (T1,T2) and (T2,T1), this will forms a cross and this cannot be serially equivalent.

(g) tell

Where there is a cycle or a cross, the second side should have a lock to wait till the first side release the lock. For example, in this case we have (T1,T2),(T2,T3),(T3,T1). Then in the transaction of (T3,T1), there should be a lock on T1.

(h) timestamps

T1 was not available since it was writing a reading value.

(i) modification

Take write(A,z+1) in T2 above and move x=read(A) in T1 down to make it in serial T2->T1. Thus, we would have $T2- > T3- > T1$ in total serial order and not one end even all started.

QUESTION 2

(a) Rounds

As the recurrence shows, $N_{t+1} = |N_t + N_t(N - N_t)/(N - 1)|$ So we have the following series.

t=0, N = 1;

t=1, N = 2;

t=2, N = 3;

t=3. N = 3+2 = 5;

...

t=11, N = 100;

So the round we need is 11.

(b) Rounds

(c) MINING

Sorry, I have searched from 1-100,000; still did not find it. Here is the code.

Okay, I find it now it's 668091. SO, sc54 668091.

```
import os
for i in range(1,100000):
    retcode = os.popen("echo sc54 " + str(i) + " | sha256sum")
    result = retcode.readlines()
    if (result[0][0:5] == "00000")
    print(i)
    break
```

QUESTION 3

(a). Commit or Abort

T1 would be committed first; then T2 would be aborted since WITHDRAW C is violated; T3 would be committed and finally T4 would be Aborted

(b). Final Values

A:0; B:30; C:40

QUESTION 4

(a). non-serial

read(A)T1, write(B)T1, write(A)T1, read(C)T2, write(D)T2, read(C)T1, write(E)T1, read(A)T2, read(E)T2, write(B)T2;

(b). partial

First, T2 starts until it readlock(A) and read(A).

Then, T1 is available to readlock(A) and read(A). Then writelock(B) write(B).

Now the first lock is needing T2 to release readlock(A) for T1 to writelock(A) and write(A).

While in T2, it need T1 to release writelock(B) to write(B).

(c). 2-phase

read(A)T1, write(B)T1, write(A)T1, read(C)T2, write(D)T2, read(A)T2, read(C)T1, write(E)T1, read(E)T2, write(B)T2;

Here when in T2 read(A), it starting releasing locks from T1 of writelock(A). However, still in T1, later it requires locks for read(C) and write(E). Thus, it's still requiring the locks, which is not a 2-phase locking condition.

(d). t1-abort

We focus on A in this situation. We should let T1 to read(A) first then T2 read(A) then T1 write(A).

Thus, for A.RTS, it is [1,2]. But when T1 is trying to write A. Its id is 1 which is lower than the maximum id of A.RTS, which is 2. Thus, it would call T1's abort.

(e). commit

The order is same as the order in sub-question (a).

Details by timestamp:

A.RTS = [1]; then B.TW = 1; then A.TW = 1; then move to T2;

C.RTS = [2]; D.TW = 2; then move to T1;

C.RTS = [2,1]; E.TW = 1; then T1 all commited and go to T2;

now we can process the following 2 process since all needed in T1 is commited. Thus, A.RTS = [1,2], E.RTS = [2]; And since it was in T2, 2 is the largest number so the final process write(B) can be committed.