Exercises on Concurrency Control (part 2)

Maurizio Lenzerini
Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
Università di Roma "La Sapienza"

Anno Accademico 2018/2019

http://www.dis.uniroma1.it/~lenzerin/index.html/?q=node/53

Esercise 1

Let C be the class of all the schedules S that, when given as input to a timestamp-based scheduler, is such that the scheduler accepts the schedule S and, when processing S, uses the Thomas rule at least once. Prove or disprove that every schedule in the class C is view-serializable.

Solution to exercise 1

The schedule

r1(X) r2(Y) w2(X) c2 w1(X) c1

is accepted by the timestamp-based method and is processed by using the Thomas rule. However, it is not view-serializable.

Esercise 2

Prove or disprove that every rigorous schedule is view-serializable

Solution to exercise 2

We recall the definition of rigorous schedule: a (complete) schedule is rigorous if for every pair of conflicting actions <ai(x),bj(x)>, where ai(X) appears before bj(X), the commit operation of Ti appears between them. Let S be a rigorous schedule on transactions T1,…,Tn, and let R be the serial schedule on T1,…,Tn reflecting the order of the commit operations in S. We show that S is conflict-equivalent to R. It is sufficient to show that, if ai(X),bj(X) is a pair of conflicting actions in S, with ai(X) appearing before bj(X) in S, then they appear in the same order in R. The proof of this claim is easy: since S is rigorous, the commit action of transaction Ti appears between ai(X) and bj(X), which means that Ti commits before Tj in S. Now, since R reflects the order of the commit actions, Ti comes before Tj in R, and therefore the claim is proven. Since we have shown that S is conflict-equivalent to a serial schedule on the same set of transactions, we have proved that S is conflict-serializable, and therefore is view-serializable too. This concludes the proof of the theorem.

Exercise 3

Let A and B be two elements of the database such that rts(A)=rts(B)=wts(A)=wts(B)=wts-c(A)=wts-c(B)=0 and cb(A)=cb(B)=true. Suppose that the system clock is 0, and that the system uses the clock values as the timestamps to be assigned to the various transactions (at their first action). Illustrate the actions of the timestamp-based scheduler when receiving the following complete schedule

r1(A) r2(B) r3(A) r2(A) w1(A) w3(A)

and tell whether the resulting sequence of actions (obtained by ignoring aborted transactions) is a strong strict 2PL schedule (with shared and exclusive locks).

Solution to exercise 3 (1)

- The initial situation is:

  wts(A) = wts-c(A) = wts(B) = wts-c(B) = rts(A) = rts(B) = 0 and cb(A)=cb(B)=true

- The system responds as follows:
  - r1(A) → ok → ts(T1)=1, rts(A)=1 -- because ts(T1)>= wts(A) and cb(A)=true
  - r2(B) → ok → ts(T2)=2, rts(B)=2 -- because ts(T2)>= wts(B) and cb(B)=true
  - r3(A) → ok → ts(T3)=3, rts(A)=3 -- because ts(T3)>= wts(A) and cb(A)=true
  - r2(A) → ok → rts(A)=3 -- because ts(T2)>= wts(A), cb(A)=true, and max(ts(T2),rts(A))=3
  - w1(A) → no: T1 rollbacks -- because ts(T1)<rts(A)=3 (write too late)
  - w3(A) → ok → wts(A)=3, cb(A)=false -- because ts(T3)= rts(A) and ts(T3)>=wts(A)

Solution to exercise 3 (2)

Since T1 rollbacks, the accepted complete schedule (the one obtained from the original one by ignoring the actions of the aborted transactions) is the following:

r2(B) r3(A) r2(A) w3(A)

that is a strong strict 2PL schedule (with shared and exclusive locks).

Exercise 4

Let A and B be two elements of the database such that rts(A)=rts(B)=wts(A)=wts(B)=wts-c(A)=wts-c(B)=0 and cb(A)=cb(B)=true. Suppose that the system clock is 0, and that the system uses the clock values as the timestamps to be assigned to the various transactions (at their first action). Illustrate the actions of the timestamp-based scheduler when receiving the following complete schedule

r1(B) w1(A) w2(B) w1(B) r2(A)

and tell whether such a schedule is a strong strict 2PL schedule (with shared and exclusive locks)

Solution to exercise 4 (1)

- The initial situation is:

  wts(A) = wts-c(A) = wts(B) = wts-c(B) = rts(A) = rts(B) = 0 and cb(A)=cb(B)=true

- The system responds as follows:
  - r1(B) → ok → ts(T1)=1, rts(B)=1 -- because ts(T1)>= wts(B), cb(B)=true and rts(B)=max(ts(T1),rts(B))
  - w1(A) → ok → wts(A)=1 and cb(A)=false -- because ts(T1)>= wts(A), cb(A)=true and ts(T1)>= rts(A)
  - w2(B) → ok → ts(T2)=3, wts(B)=3 and cb(B)=false -- because ts(T2)>= wts(B), cb(B)=true and ts(T2)>= rts(B)
  - w1(B) → T1 waiting for the commit or rollback of T2 -- because cb(B)=false, ts(T1) = rts(B) and ts(T1)<wts(B)
  - r2(A) → T2 waiting for the commit or rollback of T1 -- because cb(A)=false, and ts(T2)>wts(A) ===> **DEADLOCK!**

Solution to exercise 4 (2)

It is easy to see that the schedule is not in the class of 2PL schedules (with shared and exclusive locks).

Indeed, in order to release the lock on B, T2 should acquire the shared lock on A, but this is impossible, because T1 has such lock, and it cannot release the exclusive lock it has on A without breaking the 2PL rule, because it needs it for reading A.

Exercise 5

Let A and B be two elements of the database such that rts(A)=rts(B)=wts(A)=wts(B)=wts-c(A)=wts-c(B)=0 and cb(A)=cb(B)=true. Suppose that the system clock is 0, and that the system uses the clock values as the timestamps to be assigned to the various transactions (at their first action). Illustrate the actions of the timestamp-based scheduler when receiving the following complete schedule ("c" stands for commit and "a" for rollback)

r1(A) w2(A) c2 r3(B) w3(A) w1(A) a3 r1(B)

Solution to exercise 5

- The initial situation is:

  wts(A) = wts-c(A) = wts(B) = wts-c(B) = rts(A) = rts(B) = 0 and cb(A)=cb(B)=true

- The system responds as follows:
  - r1(A) → ok → ts(T1)=1, rts(A)=1, because ts(T1)>= wts(A), cb(A)=true and rts(A)=max(ts(T1),rts(A))
  - w2(A) → ok → ts(T2)=2, wts(A)=2 and cb(A)=false, because ts(T2)>= wts(A), ts(T2)>= rts(A) and cb(A)=true
  - c2 → ok → wts-c(A)=wts(A)=2, cb(A)=true
  - r3(B) → ok → ts(T3)=4, rts(B)=4, because ts(T3)>= wts(B), cb(B)=true and rts(B)=max(ts(T3),rts(B))
  - w3(A) → ok → wts(A)=4 and cb(A)=false, because ts(T3)>= wts(A), cb(B)=true and ts(T3)>= rts(A)
  - w1(A) → T1 waiting for the commit or rollback of T3, because cb(A)=false, ts(T1) >= rts(A) and ts(T1)<wts(A)
  - a3 → cb(A)=true, wts(A)=wts-c(A)=2
  - w1(A) → ignored by Thomas rule
  - r1(B) → ok → rts(B)=4, because ts(T1)>= wts(B), and rts(B)=max(ts(T1),rts(B))

Exercise 6

In the DBMS called Misty, a total order D is defined on the set of elements in the database, and the concurrency control strategy adopted by Misty is the following:

a) if a transaction T_1 reads an element X written by transaction T_2 (i.e., T_2 was the last transaction that wrote the element when T_1 reads it), and the first element used (i.e., read or written) by T_2 does not precede the first element used by T_1 according to D, then T_1 is aborted, otherwise T_1 continues;

b) analogously, if a transaction T_1 writes on an element X written by transaction T_2, or read by transaction T_2 (i.e., T_1 was the first transaction that wrote X after such reading), and the first element used by T_2 does not precede the first element used by T_1 according to D, then T_1 is aborted, otherwise T_1 continues.

Prove or disprove the following claim: every schedule accepted by Misty is serializable.

Solution to exercise 6 (1)

Consider a schedule S accepted by Misty, and let G be the precedence graph associated to S. We will prove that if the precedence graph G associated to S is cyclic, then we have a contradiction. This is obviously equivalent to say that if a schedule S is accepted by Misty, then the precedence graph G associated to S is acyclic, and therefore S is conflict-serializable, and hence serializable.

We proceed by proving three claims. In all of them, we assume that S is a schedule accepted by Misty, and G is the associated precedence graph.

Solution to exercise 6 (2)

First claim: if there is an edge from T_i to T_j in G, then the first element used by T_i in S precedes the first element used by T_j in S according to D.

Proof: the edge from T_i to T_j in G comes from alpha(x) in T_i and beta(x) in T_j, where at least one of alpha and beta is “write”. Therefore, S has the form:

$$\ldots,\text{alpha}(x),\ldots,\text{gamma}_1(x),\ldots.,\text{gamma}_k(x),\ldots,\text{beta}(x),\ldots$$

Where alpha(x),gamma$_1$(x),…,gamma$_k$(x),beta(x) are all the actions on x in S. We proceed by induction on k.

If k = 0, then there are two cases:

- alpha is “read” (therefore beta is “write”): T_j writes on an element read by T_i, and therefore the first element used by T_i precedes the first element used by T_j according to D.
- alpha is “write” (therefore beta is “read”,or “write”): T_j reads an element or writes on an element written by T_i and therefore the first element used by T_i precedes the first element used by T_j according to D.

Solution to exercise 6 (3)

If k >0, then there are two cases:

- $gamma_1(x),\dots,gamma_k(x)$ are all "read".
  - If beta is "read", then alpha is "write", which means that T_j reads an element written by T_i; so, the first element used by T_i precedes the first element used by T_j according to D.
  - If beta is "write" and alpha is "read", then T_j writes on an element read by T_i; so, the first element used by T_i precedes the first element used by T_j according to D.
  - If beta is "write" and alpha is "write", then T_j writes on an element written by T_i ; so, the first element used by T_i precedes the first element used by T_j according to D.
- at least one (say $gamma_h$) of $gamma_1(x),\dots,gamma_k(x)$ is "write" (and therefore h is different from i and j). Thus, S is constituted by

  $$S1 = alpha(x),\dots,gamma_h(x) \quad \text{and} \quad S2 = gamma_h(x),\dots,beta(x)$$

  By induction hypothesis on S1 and S2, the first element used by T_j precedes the first element used by T_h according to D, and the first element used by T_h precedes the first element used by T_j according to D. By transitivity, we can conclude that the first element used by T_i precedes the first element used by T_j according to D.

Solution to exercise 6 (4)

Second claim: if there is a path from T_i to T_j in G, then the first element used by T_i precedes the first element used by T_j according to D.

Proof: easy, by induction on the length of the path, where the base step of the induction is provided by the first claim.

Third claim: a cycle is a path of length greater than 1 from T_i to T_j (for some T_i) in G. Therefore, by the second claim, if there is a cycle in G, then the first element used by T_i precedes the first element used by T_j according to D. Since this is impossible, we conclude that G is acyclic, and therefore S is conflict-serializable and hence serializable.