Concurrency Control

Remember

- A *schedule* is a sequence of actions (reads and writes) belonging to multiple transactions.
- A *serial schedule* is a schedule in which transactions are not executed concurrently.
- Two actions in a schedule are in conflict if
  1. They belong to the same transaction; or
  2. they act upon the same element and one of them is a write
- A schedule is *conflict-serializable* if we can obtain a serial schedule by repeatedly swapping *non-conflicting* actions.

Concurrency Control

Remember

- Transaction T_1 takes *precedence* over transaction T_2 if there are two actions $A_1 \in T_1$ and $A_2 \in T_2$ such that:
  1. A_1 is ahead of A_2
  2. Both actions are in conflict.
- The *precedence graph* of a schedule represents each transaction of the schedule as node, and adds edge from T_1 to T_2 if T_1 takes precedence over T_2.
- The precedence graph is acyclic if and only if the schedule is conflict-serializable.

Concurrency Control

Exercise 18.2.4 b

The schedule is:

$$r_1(A); w_1(B); r_2(B); w_2(C); r_3(C); w_3(A)$$

Task:

- Give the corresponding precedence graph.
- If possible, give an equivalent conflict-serializable schedule.
- Are there any serial schedules that must be equivalent, while not conflict equivalent?

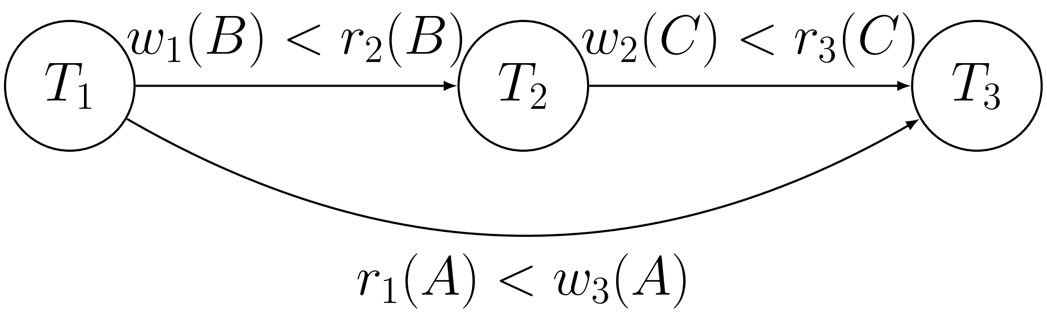
Concurrency Control

Exercise 18.2.4 b

The schedule is:

$$r_1(A); w_1(B); r_2(B); w_2(C); r_3(C); w_3(A)$$

Hence, the precedence graph is:



The schedule is already serial and conflict-free. No other serial schedule is equivalent (why?).

Concurrency Control

Exercise 18.2.4 d

The schedule is:

$$r_1(A); r_2(A); w_1(B); w_2(B); r_1(B); r_2(B); w_2(C); w_1(D);$$

Task:

- Give the corresponding precedence graph.
- If possible, give an equivalent conflict-serializable schedule.
- Are there any serial schedules that must be equivalent, while not conflict equivalent?

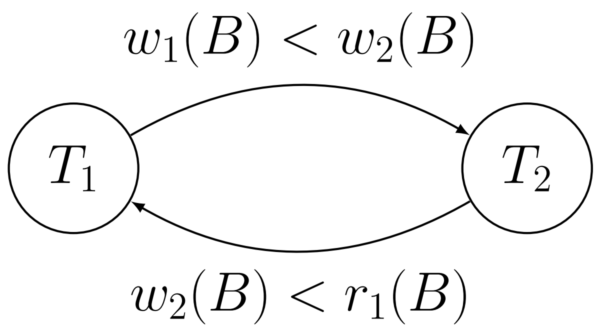
Concurrency Control

Exercise 18.2.4 d

The schedule is:

$$r_1(A); r_2(A); w_1(B); w_2(B); r_1(B); r_2(B); w_2(C); w_1(D);$$

Hence, the precedence graph is:



This precedence graph contains a loop; the corresponding schedule is therefore not conflict-serializable.

Concurrency Control

Exercise 18.2.4 d

The schedule is:

$$r_1(A); r_2(A); w_1(B); w_2(B); r_1(B); r_2(B); w_2(C); w_1(D);$$

Let's assume that the actions are as follows:

- T_1 write 1 in B
- T_2 write 42 in B
- T_2 write the value read from B in C
- T_1 write the value read from B in D

Let us further assume that there exist a serial schedule. In that schedule, independently of whether T_1 or T_2 executes first, C will contain 42 and D will contain 1.

Here, where both C and D will contain 42. Hence, no serial schedule is equivalent for all transaction.

Concurrency Control

Exercise 18.2.4 e

The schedule is:

$$r_1(A); r_2(A); r_1(B); r_2(B); r_3(A); r_4(B); w_1(A); w_2(B)$$

Task:

- Give the corresponding precedence graph.
- If possible, give an equivalent conflict-serializable schedule.
- Are there any serial schedules that must be equivalent, while not conflict equivalent?

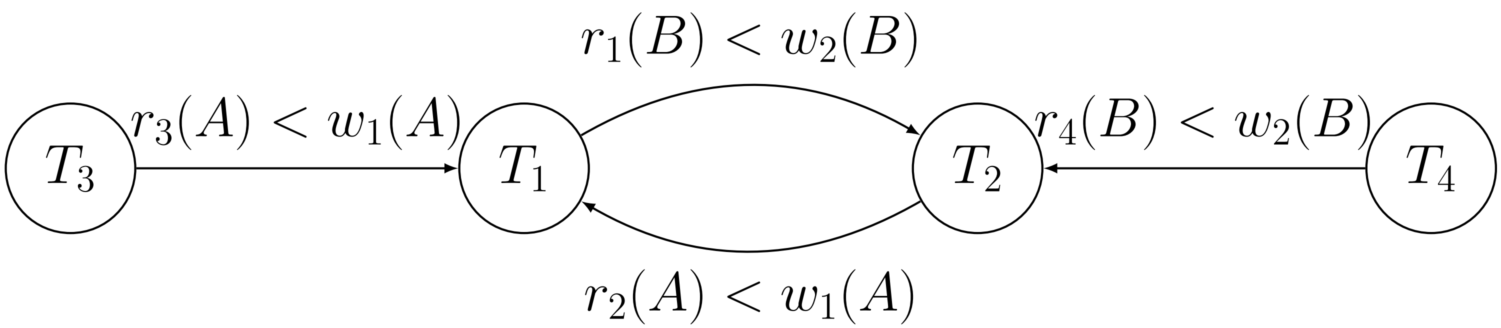
Concurrency Control

Exercise 18.2.4 e

The schedule is:

$$r_1(A); r_2(A); r_1(B); r_2(B); r_3(A); r_4(B); w_1(A); w_2(B)$$

Hence, the precedence graph is:



Concurrency Control

Exercise 18.2.4 e

This precedence graph contains a loop; the corresponding schedule is therefore not conflict-serializable.

An equivalent serial schedule could first contain T_3 and T_4, followed by T_1 and T_2.

Let's assume that both transactions T_1 and T_2 write $(A + B)$. With the given schedule, both A and B will take the same value.

If T_1 is executed first (resp. second), $B \leftarrow A + 2B$ (resp. $A \leftarrow 2A + B$). Hence, no serial schedule is equivalent for all transaction.

Concurrency Control

Remember

A timestamp-based scheduler stores for each transation T, a timestamp TS(T), and for For each database element X:

- RT(X): The highest timestamp of a transaction that read X
- WT(X): The highest timestamp of a transaction that wrote X
- C(X): A boolean value, true iff the most recent transaction to write X has commited.

Such scheduler can allow a read/write request to proceed, or *abort* and restart the transaction that made the request.

To abort a transaction, the the scheduler resets the value of X, and WT(X) to their last values.

Concurrency Control

Remember

The scheduler validates a read request $r_T(X)$ as follows:

- If $C(X)$ if false, wait for $C(X)$ to become true, or for the transaction that wrote X to aborts.
- If $C(X)$ is true and $\mathsf{TS}(T) > \mathsf{WT}(X)$, allow the read to proceed.
- Otherwise, abort and restart T.

Concurrency Control

Remember

The scheduler validates a write request $w_T(X)$ as follows:

- If $TS(T) \geq RT(X)$, and $TS(T) < WT(X)$, and $C(X)$ is false, wait for $C(X)$ to become true, or for the transaction that wrote X to aborts.
- If $TS(T) \geq RT(X)$, and $TS(T) < WT(X)$, and $C(X)$ is true, allow the write to proceed, but make no change to the database: X has already been overwritten.
- If $TS(T) \geq RT(X)$, and $TS(T) \geq WT(X)$, allow the write to proceed.
- Otherwise, abort and restart T.

Concurrency Control

Exercise 18.8.1 a

Given the following sequence of events:

$$st_1; st_2; r_1(A); r_2(B); w_2(A); w_1(B)$$

Task:

Tell what happens as each event occurs for a timestamp based scheduler.

Concurrency Control

Exercise 18.8.1 a

Given the following sequence of events:

$$st_1; st_2; r_1(A); r_2(B); w_2(A); w_1(B)$$

- T_1 starts first and hence gets a lower timestamp (e.g. $TS(1) = 1, TS(2) = 2$).
- The two first reads are allowed, and $RT(A) \leftarrow TS(1)$, $RT(B) \leftarrow TS(2)$.
- When $w_2(A)$ occurs it is allowed: $RT(A) \leq TS(2)$. Hence, $WT(A) \leftarrow TS(2)$, and $C(A) \leftarrow$ false.
- T_2 can commit, and set $C(A) \leftarrow$ true.
- However, when $w_1(B)$ occurs, $RT(B) \not\leq TS(1)$, and T_1 is aborted.

Concurrency Control

Exercise 18.8.1 c

Given the following sequence of events:

$$st_1; st_2; st_3; r_1(A); r_3(B); w_1(C); r_2(B); r_2(C); w_3(B); w_2(A)$$

Task:

Tell what happens as each event occurs for a timestamp based scheduler.

Concurrency Control

Exercise 18.8.1 c

Given the following sequence of events:

$$st_1; st_2; st_3; r_1(A); r_3(B); w_1(C); r_2(B); r_2(C); w_3(B); w_2(A)$$

- Each transaction gets a timestamp in order of their start point.
- The two first reads succeed, and $RT(A) \leftarrow TS(1)$, $RT(B) \leftarrow TS(3)$.
- $w_1(C)$ is allowed: $RT(C) \leq TS(1)$. Hence, $WT(C) \leftarrow TS(1)$, and $C(C) \leftarrow$ false. Then, T_1 can commit.
- $r_2(B)$ is allowed: $WT(B) \leq TS(2)$. However, $RT(B)$ needs not be updated. Why?
- $r_2(C)$ is allowed, but T_2 is paused until T_1 has committed: $WT(C) = TS(1) \leq TS(2)$.
- $w_3(B)$ is allowed: $RT(B) \leq TS(3)$, and $WT(B) \leq TS(3)$. Then, T_3 can commit.
- $w_2(A)$ is allowed: $RT(A) \leq TS(2)$, and $WT(A) \leq TS(3)$. Then T_2 can commit.

Concurrency Control

Exercise 18.8.2 a

Given the following sequence of events:

$$st_1; st_2; st_3; st_4; w_1(A); w_2(A); w_3(A); r_2(A); r_4(A);$$

Tell what happens as each event occurs for (a) a multiversion timestamp scheduler, and (b) a scheduler that does not maintain multiple versions.

- In a multiversion system, the three writes create three different versions of A. When T_2 reads A, it is given the value that it wrote itself. When T_4 reads A, it gets the value written by T_3, since it was the last to write a value.
- With a scheduler that only maintains one version, T_2 would be forced to abort.

Concurrency Control

Remember

A validation-based scheduler stores for each transaction T:

- RS(T): the set of element read by T
- WS(T): the set of element written by T
- For each database element X:

A transaction first reads element in RS(T), is then validated, and finally write new values for items in WS(T).

The scheduler may abort and restart T depending on its validation.

Concurrency Control

Remember

To validate a transaction T, we use the following rules:

Consider all transactions U that already passed validation, but were not finished when T started. T is valid if and only if:

$$\mathsf{RS}(T) \cap \mathsf{WS}(U) = \emptyset$$

Consider all transactions U that already passed validation, but were not finished when T started its validation. T is valid if and only if:

$$\mathsf{WS}(T) \cap \mathsf{WS}(U) = \emptyset$$

Concurrency Control

Exercise 18.9.1 c

Given the following sequence of events:

$$R_1(A, B); R_2(B, C); V_1; R_3(C, D); V_3; W_1(C); V_2; W_2(A); W_3(D)$$

Task:

Tell what happens when the sequence is processed by a validation-based scheduler.

Concurrency Control

Exercise 18.9.1 c

Given the following sequence of events:

$$R_1(A, B); R_2(B, C); V_1; R_3(C, D); V_3; W_1(C); V_2; W_2(A); W_3(D)$$

- Reads of T_1 and T_2 are processed
- Validation of T_1 is accepted (no other validated transaction)
- Reads of T_3 are processed
- Validation of T_3 is rejected $(RS(T_3) \cap WS(T_1) = \{C\})$;
- T_1 makes its writes and finishes.
- Validation of T_2 is rejected $(RS(T_2) \cap WS(T_1) = \{C\})$;

Concurrency Control

Exercise 18.9.1 f

Given the following sequence of events:

$$R_1(A, B); R_2(B, C); R_3(C); V_1; V_2; V_3; W_1(A); W_2(C); W_3(B)$$

Task:

Tell what happens when the sequence is processed by a validation-based scheduler.

Concurrency Control

Exercise 18.9.1 f

Given the following sequence of events:

$$R_1(A, B); R_2(B, C); R_3(C); V_1; V_2; V_3; W_1(A); W_2(C); W_3(B)$$

- All read requests are processed
- Validation of T_1 is accepted (no other validated transaction)
- Validation of T_2 is accepted (conditions satisfied)
- Validation of T_3 is accepted (conditions satisfied)
- T_1, T_2, and T_3 perform their writes