
DATABASE

CSE3103 : Database
FALL 2020

Nazmus Sakib
Assistant Professor
Department of Computer Science and Engineering
Ahsanullah University of Science and Technology

Serializability

- **Basic Assumption** – Each transaction preserves database consistency.
- Thus, serial execution of a set of transactions preserves database consistency.
- A (possibly concurrent) schedule is serializable if it is equivalent to a serial schedule. Different forms of schedule equivalence give rise to the notions of:

1. **conflict serializability**
2. **view serializability**

Conflict Serializability

- To improve it, two or more transactions are run concurrently. But concurrency of transactions may lead to inconsistency in database. To avoid this, we need to check whether these concurrent schedules are serializable or not.
- **Conflict Serializable:** A schedule is called conflict serializable if it can be transformed into a serial schedule by swapping non-conflicting operations.
- **Conflicting operations:** Two operations are said to be conflicting if all conditions satisfy:
  - They belong to different transactions
  - They operate on the same data item
  - At Least one of them is a write operation

Conflict Serializability

- **Conflicting** operations pair ($R_1(A)$, $W_2(A)$) because they belong to two different transactions on same data item A and one of them is write operation.
- Similarly, ($W_1(A)$, $W_2(A)$) and ($W_1(A)$, $R_2(A)$) pairs are also **conflicting**.
- On the other hand, ($R_1(A)$, $W_2(B)$) pair is **non-conflicting** because they operate on different data item.
- Similarly, (($W_1(A)$, $W_2(B)$) pair is **non-conflicting.**

Conflict Serializability

S1: $R_1(A)$, $W_1(A)$, $R_2(A)$, $W_2(A)$, $R_1(B)$, $W_1(B)$, $R_2(B)$, $W_2(B)$

If O_i and O_j are two operations in a transaction and $O_i < O_j$ (O_i is executed before O_j), same order will follow in the schedule as well. Using this property, we can get two transactions of schedule S1 as:

T1: $R_1(A)$, $W_1(A)$, $R_1(B)$, $W_1(B)$
T2: $R_2(A)$, $W_2(A)$, $R_2(B)$, $W_2(B)$

Conflict Serializability

S1: $R_1(A)$, $W_1(A)$, $R_2(A)$, $W_2(A)$, $R_1(B)$, $W_1(B)$, $R_2(B)$, $W_2(B)$

Possible Serial Schedules are: T1->T2 or T2->T1

-> **Swapping non-conflicting operation**s $R_2(A)$ and $R_1(B)$ in S1, the schedule becomes,

S11: $R_1(A)$, $W_1(A)$, $R_1(B)$, $\mathbf{W_2(A)}$, $R_2(A)$, $\mathbf{W_1(B)}$, $R_2(B)$, $W_2(B)$

Similarly, s**wapping non-conflicting operations** $W_2(A)$ and $W_1(B)$ in S11, the schedule becomes,

S12: $R_1(A)$, $W_1(A)$, $R_1(B)$, $W_1(B)$, $R_2(A)$, $W_2(A)$, $R_2(B)$, $W_2(B)$

S12 is a serial schedule in which all operations of T1 are performed before starting any operation of T2. Since S has been transformed into a serial schedule S12 by swapping non-conflicting operations of S1, S1 is conflict serializable.

Conflict Serializability

- **Question: Consider the following schedules involving two transactions. Which one of the following statement is true?**
- S1: $R_1(X)\ R_1(Y)\ R_2(X)\ R_2(Y)\ W_2(Y)\ W_1(X)$
  S2: $R_1(X)\ R_2(X)\ R_2(Y)\ W_2(Y)\ R_1(Y)\ W_1(X)$
- Both S1 and S2 are conflict serializable
- Only S1 is conflict serializable
- Only S2 is conflict serializable
- None

Two transactions of given schedules are:

T1: $R_1(X)\ R_1(Y)\ W_1(X)$
T2: $R_2(X)\ R_2(Y)\ W_2(Y)$

Conflicting Instructions

- Let l_i and l_j be two Instructions of transactions T_i and T_j respectively. Instructions l_i and l_j **conflict** if and only if there exists some item Q accessed by both l_i and l_j, and at least one of these instructions wrote Q.

  1. **l_i = read(Q), l_j = read(Q). l_i and l_j don't conflict.**
  2. **l_i = read(Q), l_j = write(Q). They conflict.**
  3. **l_i = write(Q), l_j = read(Q). They conflict**
  4. **l_i = write(Q), l_j = write(Q). They conflict**

- Intuitively, a conflict between l_i and l_j forces a (logical) temporal order between them.
  - If l_i and l_j are consecutive in a schedule and they do not conflict, their results would remain the same even if they had been interchanged in the schedule.

Conflict Serializability

- Schedule 3 can be transformed into Schedule 6 a serial schedule where T_2 follows T_1, by a series of swaps of non-conflicting instructions. Therefore, Schedule 3 is conflict serializable.

T_1	T_2
read (A)	
write (A)	
	read (A)
	write (A)
read (B)	
write (B)	
	read (B)
	write (B)

Schedule 3

T_1	T_2
read (A)	
write (A)	
read (B)	
write (B)	
	read (A)
	write (A)
	read (B)
	write (B)

Schedule 6

Conflict Serializability (Cont.)

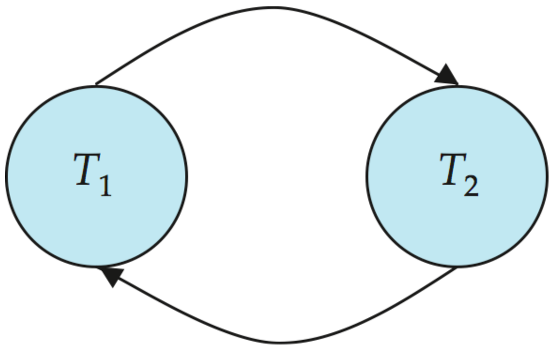
- Example of a schedule that is not conflict serializable:

T_3	T_4
read (Q)	
	write (Q)
write (Q)	

- We are unable to swap instructions in the above schedule to obtain either the serial schedule < T_3, T_4 >, or the serial schedule < T_4, T_3 >.

Precedence Graph

- Consider some schedule of a set of transactions T_1, T_2, ..., T_n
- **Precedence graph** — a direct graph where the vertices are the transactions (names).
- We draw an arc from T_i to T_j if the two transaction conflict, and T_i accessed the data item on which the conflict arose earlier.
- We may label the arc by the item that was accessed.
- **Example**



Precedence Graph

- **Precedence Graph** or **Serialization Graph** is used commonly to test Conflict Serializability of a schedule.
- It is a directed Graph (V, E) consisting of a set of nodes
- $V = \{T_1, T_2, T_3 \ldots\ldots\ldots T_n\}$ and a set of directed edges
- $E = \{e_1, e_2, e_3 \ldots\ldots\ldots\ldots\ldots\ldots e_m\}$.
- The graph contains one node for each Transaction T_i. An edge e_i is of the form $T_j \rightarrow T_k$ where T_j is the starting node of e_i and T_k is the ending node of e_i.
- An edge e_i is constructed between nodes T_j to T_k if one of the operations in T_j appears in the schedule before some conflicting operation in T_k .

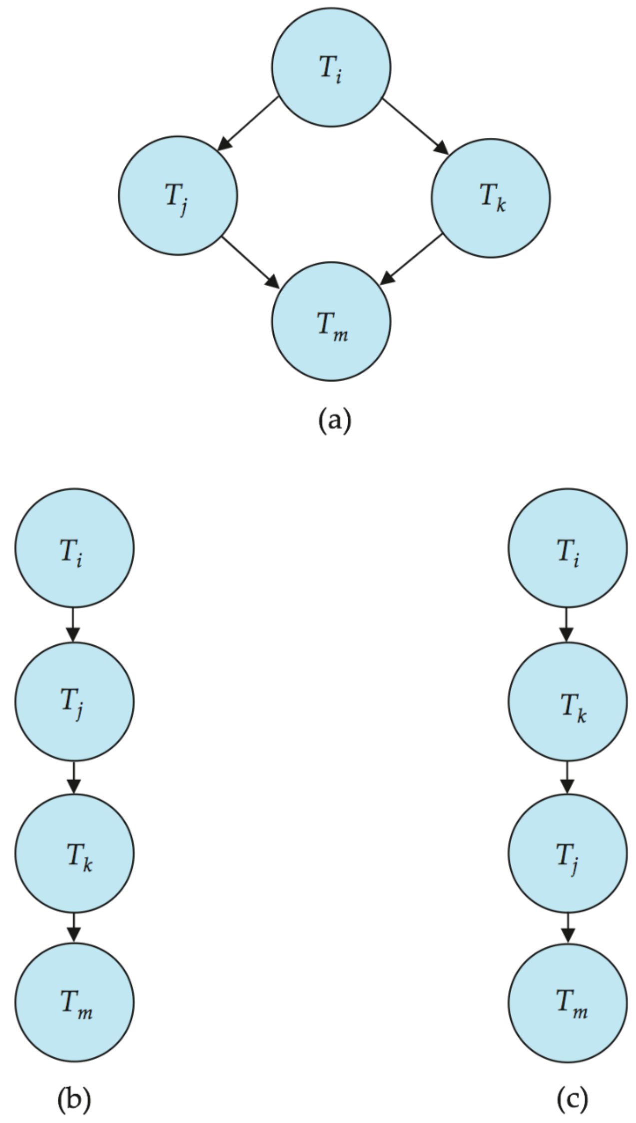
Precedence Graph

The Algorithm can be written as:

1. Create a node T in the graph for each participating transaction in the schedule.
2. For the conflicting operation read_item(X) and write_item(X) – If a Transaction T_j executes a read_item (X) after T_i executes a write_item (X), draw an edge from T_i to T_j in the graph.
3. For the conflicting operation write_item(X) and read_item(X) – If a Transaction T_j executes a write_item (X) after T_i executes a read_item (X), draw an edge from T_i to T_j in the graph.
4. For the conflicting operation write_item(X) and write_item(X) – If a Transaction T_j executes a write_item (X) after T_i executes a write_item (X), draw an edge from T_i to T_j in the graph.
5. **The Schedule S is serializable if there is no cycle in the precedence graph**.

Testing for Conflict Serializability

- A schedule is conflict serializable if and only if its precedence graph is acyclic.
- Cycle-detection algorithms exist which take order n^2 time, where n is the number of vertices in the graph.
  - (Better algorithms take order $n + e$ where e is the number of edges.)
- If precedence graph is acyclic, the serializability order can be obtained by a *topological sorting* of the graph.
  - That is, a linear order consistent with the partial order of the graph.
  - For example, a serializability order for the schedule (a) would be one of either (b) or (c)




data
distributed
database
DBMS
shared
parallel
architecture
systems
network
processing
performance
system
access
centralized
DBMSs
Distributed
DDBMS
processors
multiple
memory
sites
processor
Parallel
computer
single
technology
applications
computers
nothing
site
disk
stored
number
provide
decentralized
sometimes
local
large
may
fragments
typically
require
physically
users
Shared
one
logically
application
bottleneck
interconnection
Figure
management
transparent
organizational
administered
logical
control
scalability
approach
improve
distinction
definition
development
nodes
capable
global
makes
Management
replicated
also
many
sites
make
often
directly
disks
islands
remote
defined
controlled
based
structure
collection
provides
associated
partitioned
used
least
must
System
consists
way
incompatible
per
user
dis
within
machines
like
Database
major
fact
across
reliability
eliminates
communication
distribution
computing
split
order
scalable
architectures
referred
operational
resources
com
Date
discuss
mode
share
databases