Outline

- Introduction
- Background
- Distributed DBMS Architecture
- Distributed Database Design
- Distributed Query Processing
- Distributed Transaction Management
  - Concurrency Control Ideas
- Building Distributed Database Systems (RAID)
- Mobile Database Systems
- Privacy, Trust, and Authentication
- Peer to Peer Systems
Useful References

- B. Bhargava, *Concurrency Control in Database Systems*, IEEE Trans on Knowledge and Data Engineering, 11(1), Jan.-Feb. 1999
- Textbook *Principles of Distributed Database Systems*, Chapter 11.1, 11.2
Concurrency Control

Interleaved execution of a set of transactions that satisfies given consistency constraints.

Concurrency Control Mechanisms:
- Locking (two-phase locking)
- Conflict graphs
- Knowledge about incoming transactions or transaction typing
- Optimistic: requires validation (backout and starvation)

Some Examples:
- Centralized locking
- Distributed locking
- Majority voting
- Local and centralized validation
Basic Terms for Concurrency Control

- Database
- Database entity (item, object)
- Distributed database
- Program
- Transaction, read set, write set
- Actions
- Atomic
- Concurrent processing
- Conflict
- Consistency
- Mutual consistency
- History
- Serializability
- Serial history
Basic Terms for Concurrency Control

- Serializable history
- Concurrency control
- Centralized control
- Distributed control
- Scheduler
- Locking
- Read lock, write lock
- Two phase locking, lock point
- Crash
- Node failure
- Network partition
- Log

- Live lock
- Dead lock
- Conflict graph (Acyclic)
- Timestamp
- Version number
- Rollback
- Validation and optimistic
- Commit
- Redo log
- Undo log
- Recovery
- Abort
Concurrency Control once again

- The problem of synchronizing concurrent transactions such that the consistency of the database is maintained while, at the same time, maximum degree of concurrency is achieved.

- Anomalies:
  - Lost updates
    - The effects of some transactions are not reflected on the database.
  - Inconsistent retrievals
    - A transaction, if it reads the same data item more than once, should always read the same value.


### Execution Schedule (or History)

- An order in which the operations of a set of transactions are executed.

- A **schedule (history)** can be defined as a partial order over the operations of a set of transactions.

\[
H_1 = \{W_2(x), R_1(x), R_3(x), W_1(x), C_1, W_2(y), R_3(y), R_2(z), C_2, R_3(z), C_3\}
\]

\[
\begin{align*}
T_1: & \quad \text{Read}(x) & T_2: & \quad \text{Write}(x) & T_3: & \quad \text{Read}(x) \\
& \quad \text{Write}(x) & & \quad \text{Write}(y) & & \quad \text{Read}(y) \\
& \quad \text{Commit} & & \quad \text{Read}(z) & & \quad \text{Read}(z) \\
& & & \quad \text{Commit} & & \quad \text{Commit}
\end{align*}
\]
Formalization of Schedule

A complete schedule $SC(T)$ over a set of transactions $T=\{T_1, \ldots, T_n\}$ is a partial order $SC(T)=\{\Sigma_T, <_T\}$ where

- $\Sigma_T = \bigcup_i \Sigma_i$, for $i = 1, 2, \ldots, n$
- $<_T \supseteq \bigcup_i <_i$, for $i = 1, 2, \ldots, n$
- For any two conflicting operations $O_{ij}, O_{kl} \in \Sigma_T$, either $O_{ij} <_T O_{kl}$ or $O_{kl} <_T O_{ij}$
Given three transactions

\[ T_1: \text{Read}(x) \quad T_2: \text{Write}(x) \quad T_3: \text{Read}(x) \]
\[ \text{Write}(x) \quad \text{Write}(y) \quad \text{Read}(y) \]
\[ \text{Commit} \quad \text{Read}(z) \quad \text{Read}(z) \]
\[ \text{Commit} \]

A possible complete schedule is given as the DAG

\[ R_1(x) \quad W_2(x) \quad R_3(x) \]
\[ W_1(x) \quad W_2(y) \quad R_3(y) \]
\[ R_2(z) \quad R_3(z) \]
\[ C_1 \quad C_2 \quad C_3 \]
A **schedule** is a prefix of a complete schedule such that only some of the operations and only some of the ordering relationships are included.

\[ T_1: \text{Read}(x) \quad T_2: \text{Write}(x) \quad T_3: \text{Read}(x) \]

\[ \begin{align*}
\text{Write}(x) & \quad \text{Write}(y) \\
\text{Commit} & \quad \text{Read}(z) \\
\end{align*} \]

\[ \begin{align*}
\text{Commit} & \quad \text{Read}(z) \\
\end{align*} \]
Serial History

- All the actions of a transaction occur consecutively.
- No interleaving of transaction operations.
- If each transaction is consistent (obeys integrity rules), then the database is guaranteed to be consistent at the end of executing a serial history.

\[ T_1: \text{Read}(x) \quad T_2: \text{Write}(x) \quad T_3: \text{Read}(x) \]
\[ \text{Write}(x) \quad \text{Write}(y) \quad \text{Read}(y) \]
\[ \text{Commit} \quad \text{Read}(z) \quad \text{Read}(z) \]
\[ \text{Commit} \quad \text{Commit} \]

\[ H_s = \{W_2(x), W_2(y), R_2(z), C_2, R_1(x), W_1(x), C_1, R_3(x), R_3(y), R_3(z), C_3\} \]
Serializable History

- Transactions execute concurrently, but the net effect of the resulting history upon the database is *equivalent* to some *serial* history.

- Equivalent with respect to what?
  - **Conflict equivalence**: the relative order of execution of the conflicting operations belonging to unaborted transactions in two histories are the same.
  - **Conflicting operations**: two incompatible operations (e.g., Read and Write) conflict if they both access the same data item.
    - Incompatible operations of each transaction is assumed to conflict; do not change their execution orders.
    - If two operations from two different transactions conflict, the corresponding transactions are also said to conflict.
Serializable History

\[ T_1: \text{Read}(x) \quad T_2: \text{Write}(x) \quad T_3: \text{Read}(x) \]

\[ \text{Write}(x) \quad \text{Write}(y) \quad \text{Read}(y) \]

\[ \text{Commit} \quad \text{Read}(z) \quad \text{Read}(z) \]

\[ \text{Commit} \quad \text{Commit} \]

The following are not conflict equivalent

\[ H_s = \{ \text{W}_2(x), \text{W}_2(y), \text{R}_2(z), \text{C}_2, \text{R}_1(x), \text{W}_1(x), \text{C}_1, \text{R}_3(x), \text{R}_3(y), \text{R}_3(z), \text{C}_3 \} \]

\[ H_1 = \{ \text{W}_2(x), \text{R}_1(x), \text{R}_3(x), \text{W}_1(x), \text{C}_1, \text{W}_2(y), \text{R}_3(y), \text{R}_2(z), \text{C}_2, \text{R}_3(z), \text{C}_3 \} \]

The following are conflict equivalent; therefore

\( H_2 \) is *serializable*.

\[ H_s = \{ \text{W}_2(x), \text{W}_2(y), \text{R}_2(z), \text{C}_2, \text{R}_1(x), \text{W}_1(x), \text{C}_1, \text{R}_3(x), \text{R}_3(y), \text{R}_3(z), \text{C}_3 \} \]

\[ H_2 = \{ \text{W}_2(x), \text{R}_1(x), \text{W}_1(x), \text{C}_1, \text{R}_3(x), \text{W}_2(y), \text{R}_3(y), \text{R}_2(z), \text{C}_2, \text{R}_3(z), \text{C}_3 \} \]
Serializability in Distributed DBMS

Somewhat more involved. Two histories have to be considered:
- local histories
- global history

For global transactions (i.e., global history) to be serializable, two conditions are necessary:
- Each local history should be serializable.
- Two conflicting operations should be in the same relative order in all of the local histories where they appear together.
Global Non-serializability

\[ T_1: \quad \text{Read}(x) \quad \text{Read}(x) \]
\[ x \leftarrow x + 5 \quad x \leftarrow x \times 15 \]
\[ \text{Write}(x) \quad \text{Write}(x) \]
\[ \text{Commit} \quad \text{Commit} \]

The following two local histories are individually serializable (in fact serial), but the two transactions are not globally serializable.

\[ LH_1 = \{R_1(x), W_1(x), C_1, R_2(x), W_2(x), C_2\} \]
\[ LH_2 = \{R_2(x), W_2(x), C_2, R_1(x), W_1(x), C_1\} \]
Evaluation Criterion for Concurrency Control

1. Degree of Concurrency

\[ \text{Scheduler Recognizes or Reshuffles} \]

Less reshuffle $\Rightarrow$ High degree of concurrency

2. Resources used to recognize
   - Lock tables
   - Time stamps
   - Read/write sets
   - Complexity

3. Costs
   - Programming ease
General Comments

- Information needed by Concurrency Controllers
  - Locks on database objects
  - Time stamps on database objects
  - Time stamps on transactions

- Observations
  - Time stamps mechanisms more fundamental than locking
  - Time stamps carry more information
  - Checking locks costs less than checking time stamps
General Comments (cont.)

- When to synchronize
  - First access to an object (Locking, pessimistic validation)
  - At each access (question of granularity)
  - After all accesses and before commitment (optimistic validation)

- Fundamental notions
  - Rollback
  - Identification of useless transactions
  - Delaying commit point
  - Semantics of transactions