Distributed Database Systems
Serializability Theory

Instructor: Randal Burns
Lecture for March 4, 2000
Computer Science 600.416
Johns Hopkins University

Serializability Theory
Formalizing the Transaction Concept

• Necessary for correctness proofs and establishing invariants

• Notation we use is:
  – The $O_j^{th}$ operation of transaction $T_j$ that operates on data $x$ is called $O_{ij}(x)$
  – $O_{ij}(x) \in \{read, write\}$
  – for which each read or write operation is assumed to be atomic

• Is this assumption reasonable? For what types of systems?
Transaction Notation

• The set of all operations executed by transaction $T_i$ is
  \[ O_S_i = \bigcup_j O_{ij} \]
• The termination condition of a transaction $T_i$ is
  \[ N_i \in \{\text{abort, commit}\} \]
• Transactions are assumed to start implicitly in this model
Partial Orderings

• A transaction is a partial ordering of the operations in $OS_i$
• A partial ordering $P = \{\Sigma, \prec\}$ defines an order within the domain $\Sigma$ according to the transitive and irreflexive binary operator $\prec$
  – binary 2 arguments
  – irreflexive means $\neg(x \prec x)$
  – Transitive $(x \prec y) \land (y \prec z) \rightarrow (x \prec z)$
• The operator $\prec$ reads “precedes in execution” and implies a program order on reads and writes.

Serializability Theory
Transaction Notation

• A transaction is an ordering \( T_i = \{\Sigma_i, \prec_i\} \) where
  
  - \( \Sigma_i = OS_i \cup \{N_i\} \)
  
  - For any two operations \( O_{ij}, O_{ik} \) if \( O_{ij} = \{R(x) \text{ or } W(x)\} \) and \( O_{ik} = \{W(x)\} \) for any data item \( x \), then either \( O_{ij} \prec O_{ik} \) or \( O_{ik} \prec O_{ij} \)
  
  - \( \forall O_{ij} \in OS_i, O_{ij} \prec N \)

• Or in English
  
  - A transaction is an ordering of reads, writes, and resolutions
  
  - Any operations against the same data item, must occur in a specified order
  
  - All reads and writes must occur before the transaction resolution

Serializability Theory
Notation Explained

• What about this definition
  – the order is specified by the program
  – this ordering can be represented as a directed acyclic graph
  – the nodes of the graphs are from $\Sigma_i$ defined by the program
  – the edge relation is $\prec_i$ is given by the program.
  – must be acyclic because the definition of a partial ordering restricts the relation
Transaction Example

• $\Sigma_i = \{R(x), R(y), W(x), W(y), C\}$
• $\prec_i = \{(R(x),W(x)), (R(y),W(y))
R(y),W(x)), (*,C)\}$ for * in $R(x),R(y),W(x),W(y)$

• This ordering is given by the application.
  – Operations on same variable must be defined
  – Orderings between x and y are determined by data usage

| read x |
| read y |
| x=y+1 |
| y=y+1 |
| write x |
| write y |
| Commit |

Serializability Theory
Transaction Semantics

- Not all operations need to be ordered, for example, $y$ can be written any time after it has been read and could even be written before $R(x)$
- Data must be read before it used
- Variables used together in an expression create an ordering dependence
  - $x = y + 1$ requires $R(y)$ before $W(x)$
DAG Representation

- The corresponding digraph looks like
  - example

- A DAG is a nice and succinct representation, because we can omit any edges that are implied by transitive closure.
- The partial ordering/DAG definition of a transaction defines
  - Legal re-orderings and re-writes of a transaction
  - Useful for analyzing concurrency control
Serializability Theory

• Built on the formal concept of a transaction
  – A transaction is an ordering on only the operations that it conducts, a schedule is an ordering on all operations from all concurrent transactions
  – A schedule $S$ also called a history is defined over a set $T$ of transactions and specifies an interleaved order of their operations (reads, writes and termination conditions) $T = \{T_1, T_2, \ldots, T_n\}$

• The type of serializability we study is called conflict serializability
  – There are other types (e.g. view serializability) that are more relaxed
  – Every conflict serializable schedule is view serializable
Data Conflicts

• A conflict exists between $O_{ij}(x)$ and $O_{kl}(x)$ if either of the operations is a write
  – Two types of conflicts read-write and write-read
  – *Why are there not 3 types, i.e. why are read-write and write-read the same?* They are often considered to be different. But, for database, they point is that operations conflict and need to be ordered. Their order is specified by the equivalent serialization, so there is not any motivation to treat the types of conflicts differently (at this point).
  – Note that this definition is general enough to capture conflicts in the same transaction as well as between different transactions.
Complete Schedule

• A complete schedule (orders all operations) is defined over $T = \{T_1, T_2, \ldots, T_n\}$ and is a partial order $S_T^c = \{\Sigma_T, \prec_T\}$
  
  - $\Sigma_T = \bigcup_{i=1}^{n} \Sigma_i$
  - $\prec_T \supseteq \bigcup_{i=1}^{n} \prec_i$
  - For any conflicting $O_{ij}, O_{kl} \in \Sigma_T$ either $O_{ij} \prec_T O_{kl}$

• Or in English
  
  - The schedule orders all operations in all transactions
  - The partial order preserves all orderings internal to transactions
  - There is a global execution order on all conflicts

Serializability Theory
Ordering Transactions

• For transactions $T_1$, $T_2$, $T_3$, a serial schedule is when all of the operations of a transaction precede the next transaction
  – $T_1 \prec T_2 \prec T_3$ or $T_1 \rightarrow T_2 \rightarrow T_3$

• Based on precedence enforced by partial orders, we can write the ordering dependency graphs
  – example
  – Any “conflict equivalent” schedule is serializable, *i.e.* has the same effect on the database.
Serializerizability Theory

**Distributed Global Schedules**

- Serialization extends to distributed databases directly when they are not replicated.
- For replicated DBs, we require a *global schedule* in addition to the *local schedule* enforced by transactions.
  - Local schedule – execution order of transactions at any one site.
  - Global schedule – execute transactions in same local order at all sites.
- Example.

- Legal orderings at each site *local schedule* violates database consistency guarantees.
  - Part of the overhead of replication.
What was I supposed to learn?

• The commands within a transaction are ordered by the application
  – Include orderings on different data items that are used together in an expression, \( x = y + 1 \).

• Transaction orderings can be put together to create a global ordering
  – Based only on conflicting reads and writes to shared data items
    • No dependencies between read(\(x\)) in T1 and write(\(y\)) in T2
    • Transactions encapsulate all dependencies between data items

• All of this can be represented as a DAG
  – Linear orderings can be generated from the DAG
  – Called a precedence graph

Serializability Theory