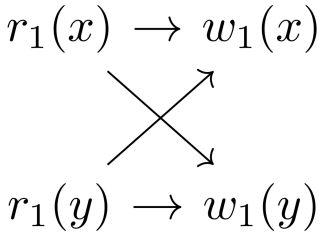
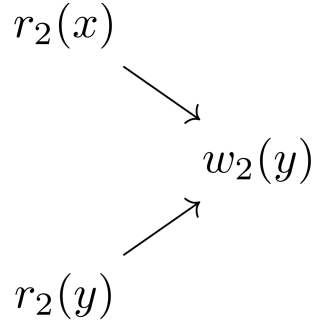
Data Management Algorithms
Consistency, Concurrency control, and Recovery
Consistency: Correctness Criteria

Boris Novikov

Transactions in Page Model

- Semantics of operations: *r(x)* reads last preceeding *w(x)*
- Transactions: partially ordered finite sets of operations
- Partial vs. total ordering





Histories and Scheduls

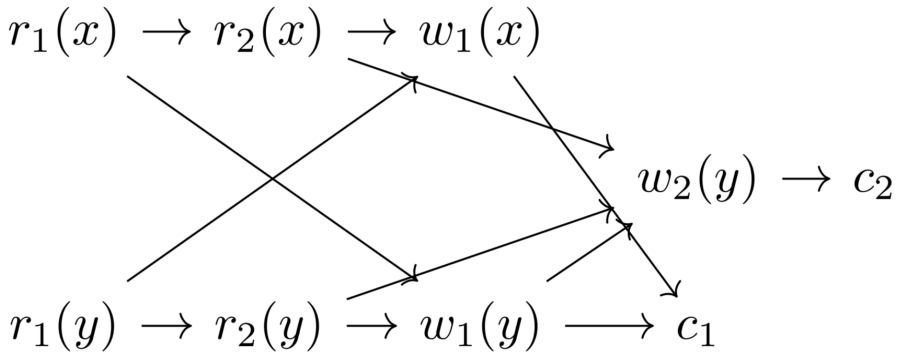
A history H for finite set of transactions $t_1, t_2, \ldots t_n$:

- Operations $op(H) = \bigcup_i op(t_i) \cup (\{c_i \mid a_i\})$
- Partial ordering compatible with ordering in t_i;$p_i, q_i \in op(t_i) \quad p_i <_i q_i \Rightarrow p_i <_s q_i$
- For any x operations on x are ordered: $p_i(x) < q_j(x)$ or $q_j(x) < p_i(x)$
- Commit or abort follows all operations of th etransaction: $p_i < c_i \vee p_i < a_i$

A schedule s is a prefix of H

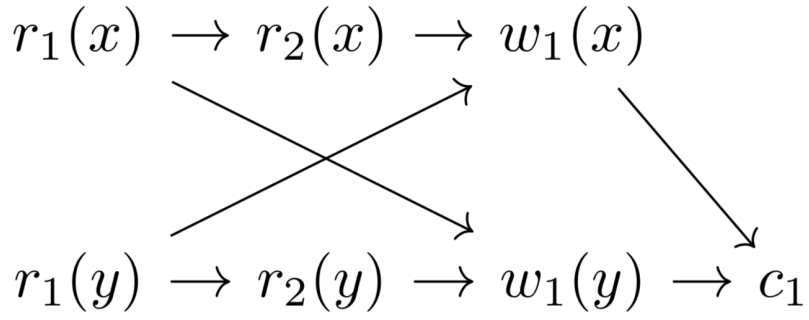
Histories and Schedules: Examples

- Partially ordered history



- Totally ordered history
  $r_1(x)\ r_2(x)\ r_1(y)\ w_1(x)\ r_2(y)\ w_1(y)\ c_1\ w_2(y)\ c_2$

- Schedule



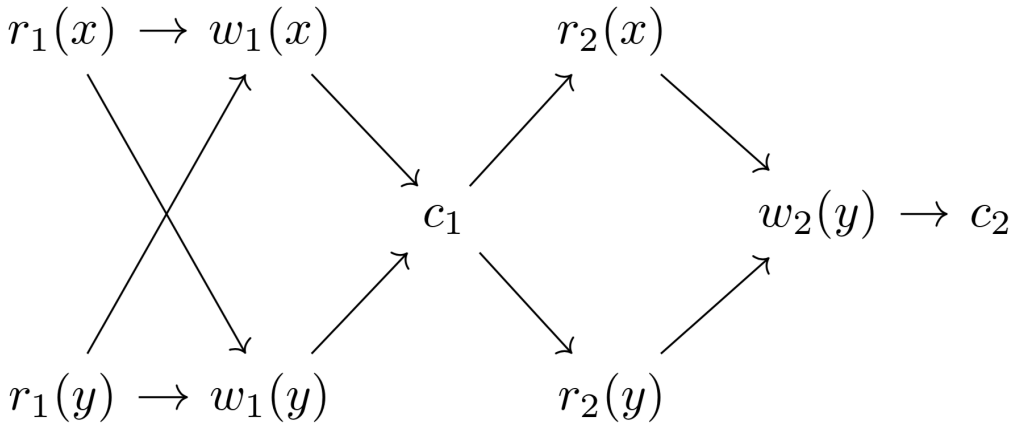
Anomalies of Concurrent Execution

- Lost update: $r_1(x)\ r_q(x)\ w_1(x)\ w_2(x)\ c_1\ c_2$
- Inconsistent read (read skew): $r_1(x)\ r_2(x)\ w_1(x)\ w_1(y)\ r_2(y)\ c_1\ c_2$
- Dirty read: $r_1(x)\ w_1(x) r_2(x) a_1 c_2$
- 20+ more
- Some of anomalies cannot be expressed in the page model

Serial Histories and Schedules

History is serial if:

- Transactions (but not necessarily operations) can be totally ordered:
  $t_1 < t_2 < \ldots < t_n$
- If $t_i < t_j$ then $\forall p_i \forall q_j \quad p_i <_s q_j$
- A serial schedule is a prefix of a serial history
- Serial histories are correct. Why?



Transaction Sets for a Schedule

- All transactions: *trans(s)*
- Committed: *commit(s)*
- Aborted *abort(s)*
- Not yet compelted: *active(s)*
-

Requirements for Correctness Criteria

- The class of correct schedules must be wide
- The correcteness criteria should be computationally efficient
- Aborted transactions are not essential for correctness
-

The Approach

- Define an equivalence of schedules
- A schedule is correct iff it is equivalent to some serial schedule on the same set of transactions
- For any equivalence of schedules EQ, the correct schedules are also called **EQ-serializable**
- Several different equivalences will be considered

Herbrand Semanties: Informally

- Any read $r_i(x)$ returns the value written by the last preceeding write $w_j(x)$ (executed by any transaction t_j)
- A value written by $W_i(x)$ depends on
  - ✴The functions of an application that isued th etransaction (including any user input)
  - ✴Any read operations $r_i(y)\ldots$ of the same transaction before $w_i(x)$

Herbrand Semantics for Operations

- Transaction t_0 writes the initial state of all pages x in the database

$$H_s(r_i(x)) = H_s(w_j(x)) \quad w_j(x) <_s r_i(x), \quad \forall w_k(x)\ w_k(x) < r_i(x) \Rightarrow w_k(x) < w_j(x)$$

$$H_s(w_i(x)) = f_{ix}(r_i(y^1), \ldots r_i(y^m)) \quad r_i(y^k) < sw_i(x)$$

Where f_{ix} represent the application logic.

Herbrand Semantics for Schedules

- Herbrand universe is a set of expressions $HU = \{f_{ix}(e_1, \ldots e_m), \quad e_k \in HU\}$ (this definition is recursive)
- Herbrand semantics for a schedue is a mapping from the databse to Herbrand universe: $H(s) : D \rightarrow HU$

Final State Serializability - FSR

- Schedules $s,\ s'$ are final-state equivalent iff $op(s) = op(s') \wedge H(s) = H(s')$.
- A schedule is final state serializable, $s \in$ FSR, if it is FS-equivalent to a serial chedule

FSR Examples

$$r_1(x)\ r_2(y)\ w_1(y)\ r_3(z)\ w_3(z)\ r_2(x)\ w_2(z)\ w_1(x)$$

$$r_3(z)\ w_3(z)\ r_2(y) r_2(x)\ w_2(z)\ r_1(x)\ w_1(y)\ w_1(x)$$

are FS-equivalent

$$r_1(x)\ r_2(y\ )_w 1(y)\ w_2(y)\ c_1\ c_2 \quad f_{2y}(f_{0y}())$$

$$r_1(x)\ w_1(y)\ r_2(y)\ w_2(y)\ c_1\ c_2 \quad f_{2y}(f_{1y}(f_{0y}()))$$

Are not equivalent

Reads_From

- Relation reads_From (t_j, x, r_i) is trje iff $w_j(x)$ is the last write operation preceeding $r_i(x)$
- Relations *Reads_From* coincide for two hcedules if Herbrand semantics for all operations coincide

View Equivalence

- FSR prevents lost updates, but does not prevent inconsistent reads
- All transactions in equivalent schedules should see the same state of the database
- Schedules are view equivalent if $H_s(r_i(x)) = H_{s'}(r_i(x))$ for all read operations of all transactions
- Actually Herbrand semantics coincide also for write operations

View Serializability - VSR

- Two schedules are VS-equivalent iff the relations *Reads_From* coincide,
- FSR ≠ VSR.
- Proof: $w_1(x)\ r_2(x)\ r_2(y)\ w_1(y)\ c_1\ c_2$ ∈ FSR\VSR

VSR Properties

- Perfect from application semantics viewpoint
- Cannot be efficiently verified

Conflicts

Two operations of page model are inconflict if:

- Both process same page (implies the ordering)
- Belong to differne4t transactions
- At least one of two is the write operation.

The first operation in the conflict always preceeds the second in the schedule.

The set of all conflicts in a schedule s is $conf(s)$.

Conflict Serializability - CSR

- Schedules $s,\ s'$ are conflict-equivalent iff $op(s) = op(s') \wedge conf(s) = conf(s')$
- A schedue is conflict-serializable ($s \in$ CSR) if it is conflict-equivalent to a serial schedule.

CSR ⊂ VSR

CSR ⊆VSR:

- If $r_i(x)$ reads from $w_j(x)$ in s and from $w_k(x)$ in s' then $(w_k(x), w_j(x)) \in conf(s)$ but $(w_j(x), w_k(x)) \in conf(s)$

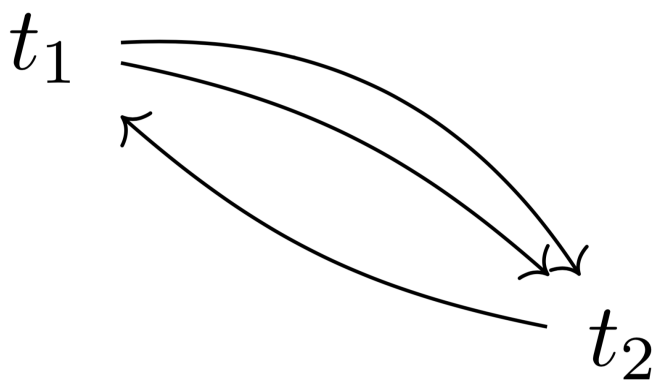
CSR ≠VSR:

- $s = w_1(x)\ w_2(x)\ w_2(y)\ c_2\ w_1(y)\ {}_c1\ w_3(x)\ w_3(y)\ c_3$

Conflict Graph

- Graph G(*s*):
  - ✳nodes: *trans(s),*
  - ✳edges*:* $conf(s)$

$$r_1(x)\ r_1(y)\ w_2(x)\ w_2(y)\ w_1(x)\ c_1\ c_2$$



CSR Criterion

$s \in$ CSR iff G(s) is acyclic

- Same conflicts produce samegraph for origianl and equivalent serial. The graph for serial schedue is acyclic.

- If the graph is acyclic, the topological sor produces equivalent serial graph.

Commutativity

- Commutativity rules
  - C1: $r_i(x) rJ(x) \sim r_j(x) r_i(x) \quad j \neq j$
  - C2: $r_i(x) w_j(isy) \sim w_j(y) r_i(x) \quad x \neq y, i \neq j$
  - C3: $w_i(x) w_j(y) \sim w_j(y) w_i(x) \quad x \neq y, i \neq j$
  - C4: $p_i(x), q_j(x)$ unordered $\Rightarrow p_i(x) q_j(y) \quad x \neq y \vee (p = r \wedge g = r)$

Commutativity Equivalence

- Equivalence: finite sequence of transformations C1 - C4 transforms $s \rightarrow s'$.
- A schedule s is commutativity-serializable iff $s \in$ CSR.
- Operations commute iff they are not inconflict.
- The advantage is that semantics is hidden.

OCSR - Transaction Order Preserving CSR

$$w_1(x)\ r_2(x)\ c_2\ w_3(y)\ c_3\ w_1(y)\ c_1$$

- Serialization order: $t_3 \rightarrow t_1 \rightarrow t_2$,
- actual order $t_2\ t_3$
- OCSR order of transactions is preserved
- OCSR $\subset$ CSR
  - Proof?

COCSR: Commit Order Preserving CSR

- Definition; $s \in$ COCSR iff for any conflict $p_i < q_j nnn \Rightarrow c_i < c_j$
- COCSR $\subset$ CSR
- COCSR $\subset$ OCSR

Relationships between CSR Subclasses

- COCSR $\subset$ OCSR $\subset$ CSR
- $s_1 = w_1(x)\ r_2(x)\ c_2 w_3(y)\ c_3\ w_1(y)\ c_1 \in$ CSR$\backslash$OCSR
- $s_2 = w_3(y)\ c_3\ w_1(x)\ r_2(x)\ c_2\ w_1(y)\ c_1 \in$ OCSR$\backslash$COSCR
- $s_3 = w_3(y)\ c_3\ w_1(x)\ r_2(x)\ w_1(y)\ c_1\ c_2 \in$ COCSR$\backslash\{$serial$\}$

Aborts and System Failures

- Transactions in *active(s)* can be aborted
- Correctness should not depend on aborts
- Any prefix of a correct schedue should be correct in the same sense

Prefix and Commit Closeness

- FSR is not prefix-closed
- VSR, CSR are prefix-closed
- *CP(s)* is a projection of a schedule *s* consisting of transactions in *commot(s)*
- A class of schedules X is commit-closed if *CP(s')* $\in$ X for any prefix *s'* of *s* $\in$ *X.*
- CMCSR $\subset$ CMVSR $\subset$ CMFSR
- CMFSR $\subset$ FSR, CMVSR $\subset$ VSR, CVCSR = CSR

Serializability Summary

$s_1 = w_1(x)\ w_2(x)\ w_2(y)\ c_2\ w_1(y)\ c_1 \notin$ FSR

$s_2 = w_1(X)\ r_2(x)\ w_2(y)\ c_2\ r_1(y)\ w_1(y)\ c_1 w_3(x)\ w_3(y)\ c_3$
$\in$ FSR\CMFSR

$s_3 = w_(x)\ r_2(x)\ w_2(y) w_1(y)\ c_1\ c_2$
$\in$ CMFSR $\cup$ VSR\CVVSR

$s_4 = w_1(x)\ w_2(x)\ w_2(y)\ c_2\ w_1(y)\ c_1\ w_3(x)\ w_3(y)\ c_3$
$\in$ VSR\CMFSR

$s_5 = w_1(x)\ r_2(x)\ w_2(y)\ w_1(y)\ c_1\ c_2\ w_3(x)\ w_3(y)\ c_3$
$\in$ CMFSR\VSR

$s_6 = w_1(x)\ w_2xx)\ w_2(y)\ c_2\ w_1(y)\ w_3(x)\ w_3(y)\ c_3 w_1(z)\ c_1$
$\in$ CMVSR\CSR

$s_7 = w_1(x)\ w_2(x)\ w_2(y)\ c_2\ w_1(z)\ c_1$
$\in$ CSR\OCSR

$s_8 = w_3(y)\ c_3 w_1(x)\ r_2(x)\ c_2\ w_1(y)\ c_1$
$\in$ OCSR\COCSR

$s_9 = w_3(y)\ c_3 w_1(x)\ r_2(x)\ w_1(y)\ c_1\ c_2$
$\in$ COCSR\{serial}

$s_{10} = w_1(x) w_1(y)\ c_1\ w_2(x)\ w_2(y)\ c_2 \quad \in \{\text{serial}\}$

Snapshot Isolation - SI

- STS(t), CTS(t) - transaction start and commit timestamps
- E(t) = (STS(t), CTS(t)) - transaction execution time interval
- RS(t), WS(t) - transaction read set and write sets

SI rules:

- ✦SI1 Any read operation oin t returns values commited before STS(t)
- ✦SI2 $E(t_1) \cap E(t_2) = \varnothing \vee WS(t_1) \cap WS(t_2) = \varnothing$

SI and Serializability

- $r_1(x)\ w_1(x)\ r_2(x)\ w_2(x)\ c_2\ r_1(y)\ w_1(y)\ c_1 \in \mathsf{CSR} \setminus \mathsf{SI}$
- $r_1(x)\ r_2(y)\ w_1(y)\ w_2(x)\ c_1\ c_2 \in \mathsf{SI} \setminus \mathsf{FSR}$

SI Anomalies

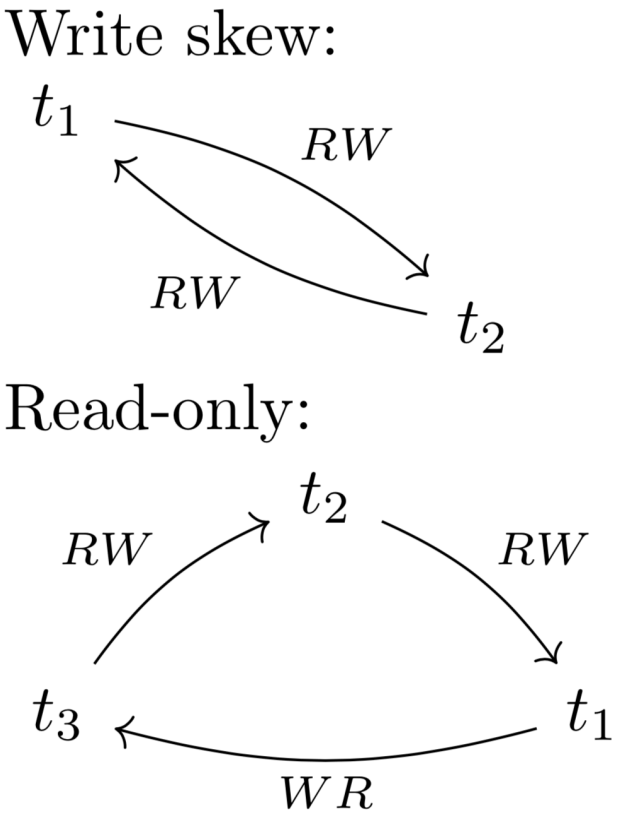
- Write skew $r_1(x)\ r_2(y)\ w_1(y)\ w_2(x)\ c_1\ c_2$
- Read-only transaction anomaly:
  $r_2(x)\ r_2(y)\ r_1(y)\ w_1(y)\ c_1\ r_3(x)\ r_3(y)\ c_3\ w_2(x)\ c_2$

Dependencies

- WR $t_1 \rightarrow t_2 : \ w_1(x) \rightarrow r_2(x)$
- WW $t_1 \rightarrow t_2 : w_1(x) \rightarrow w_2(x)$ - excluded by SI2
- RW $t_1 \rightarrow t_2 \ \ r_1(x) \rightarrow w_2(x)$

Serializable SI - SSI

- A schedule in SSI iff dependency graph is acyclic
- Only two kinds of cycles may exist under SI
- Checking for 2 edges: fast but false positives



Relaxed Isolation

- READ UNCOMMITTED
- READ COMMITTED
- REPEATABLE READ
- SERIALIZABLE