Today’s Plan

- Admin
- Last week’s homework
- Proof: $\text{Gen}(FOCC) \subseteq CSR$
- Example: Properties of MVSR
- Multiversion Concurrency Control (MVCC)
- Homework
Of the 12 submitters of assignment 6, 11 currently have > 75% of the points → 0.3 bonus

- Probably written exam,
  final decision next week (> 4 weeks before the exam)
Correctness of FOCC

Theorem 4.18:
Gen (FOCC) ⊂ CSR.

Proof:
Assume that G(s) has been acyclic and that validating \( t_j \) would create a cycle. So \( t_j \) would have to have an outgoing edge to an already committed \( t_k \). However, for all previously committed \( t_k \) the following holds:

• If \( t_k \) was committed before \( t_j \) started, then no edge \((t_j, t_k)\) is possible.
• If \( t_j \) was in its read phase when \( t_k \) validated, then WS(\( t_k \)) must be disjoint with RS*(\( t_j \)) and all later reads of \( t_j \) and all writes of \( t_j \) must follow \( t_k \) (because of the strong critical section); so neither a wr nor a ww/rw edge \((t_j, t_k)\) is possible.
Properties of MVSR

**Theorem 5.1:** \( \text{VSR} \subset \text{MVSR} \)

**Example:** \( s = r_1(x) \ w_1(x) \ r_2(x) \ w_2(y) \ r_1(y) \ w_1(z) \ c_1 \ c_2 \)

**Theorem 5.2:** Deciding if a mv history is in MVSR is NP-complete.

**Theorem 5.3:**
The conflict graph of an mv schedule \( m \) is a directed graph \( G(m) \) with transactions as nodes and an edge from \( t_i \) to \( t_j \) if \( r_j(x_i) \in \text{op}(m) \).
For all mv schedules \( m, m' \): \( m \approx_v m' \Rightarrow G(m) = G(m') \).

**Example:**
\[
\begin{align*}
m &= w_1(x_1) \ r_2(x_0) \ w_1(y_1) \ r_2(y_1) \ c_1 \ c_2 \\
m' &= w_1(x_1) \ w_1(y_1) \ c_1 \ r_2(x_1) \ r_2(y_0) \ c_2
\end{align*}
\]

\( G(m) = G(m') \), but not \( m \approx_v m' \).
MVCC

- Credits: Dr. Andrey Gubichev, 2013
Multiversion concurrency control: ideas

- Write operations are no longer “in place”. Each write operation creates a version of the data item
- Old values are always accessible
- Scheduler has to decide which version to read
A function $h$:

- $h(w_i(x)) = w_i(x)$, $w_i(x)$ writes $x_i$
- $h(r_i(x)) = w_j(x)$, for some $w_j(x) < s r_i(x)$, $r_i$ reads $x_j$

Multiversion history: history + version function
Example

- $m = r_1(x_0)w_1(x_1)r_2(x_1)w_2(y_2)r_1(y_0)w_1(z_1)c_1c_2$
- $r_1$ reads the initial version of $x$: $x_0$ etc
- $w_2$ writes a version of $y$ ($y_2$), but $r_1$ reads the initial version of $y$ ($y_0$)
Multi- vs Monoversion histories

- Generally, version function: $r$ reads the item written by some preceding $w$
  
  - $m = r_1(x_0)w_1(x_1)r_2(x_1)w_2(y_2)r_1(y_0)w_1(z_1)c_1 c_2$

- Monoversion history (everything before now): $r$ reads the item written by the last preceding $w$
  
  - $m = r_1(x_0)w_1(x_1)r_2(x_1)w_2(y_2)r_1(y_2)w_1(z_1)c_1 c_2$
Multiversion serializability

- Versions are invisible to user
- Correct multiversion schedule should be equivalent to the monoversion schedule
Reads-from relationship

- **Motivation**
  - \( s = w_0(x)c_0w_1(x)c_1r_2(x)w_2(y)c_2 \)
  - \( m = w_0(x_0)c_0w_1(x_1)c_1r_2(x_0)w_2(y_2)c_2 \)
  - yield different values of \( y \) (under Herbrand semantics)

- \( RF(m) = \{ (t_i, x, t_j) | r_j(x_i) \in op(m) \} \)

- View equivalence for \( m, m' \): same set of transactions, \( RF(m) = RF(m') \)
How to define view serializability? "View equivalent to a serial multiversion schedule"?

No: serial multiversion schedule does not have to be compatible with serial monoversion schedule:

- \[ m = w_0(x_0)w_0(y_0)c_0 r_1(x_0)r_1(y_0)w_1(x_1)w_1(y_1)c_1 r_2(x_0)r_2(y_1)c_2 \]
- \[ s = w_0(x_0)w_0(y_0)c_0 r_1(x)r_1(y)w_1(x)w_1(y)c_1 r_2(x)r_2(y)c_2 \]

- both serial, but different reads-from relation

MV view serializable schedule \( m \): there exists a serial monoversion schedule, view equivalent to \( m \).
How to test MVSR?

- There is a construct called Multiversion Serialization Graph
- If it is acyclic, the schedule is in MVS
- Important: NP-hard
- See lectures + homework
Formal definition of MVSG

- Nodes are transactions
- Edges: (consider $w_j(x_j)$, $r_k(x_j)$ and $w_i(x_i)$)
  - for $r_k(x_j)$ edge $T_j \rightarrow T_k$
  - if $x_i \ll x_j$: edge $T_i \rightarrow T_j$
  - if $x_j \ll x_i$: $T_k \rightarrow T_i$
- Edges: the order of transactions in the serial schedule
Another example of MVSR

\[ m = w_0(x_0)w_0(y_0)c_0r_1(x_0)w_1(x_1)r_2(x_1)w_2(y_2)w_1(y_1)w_3(x_3) \]

\[ x_0 << x_1 << x_3, \ y_0 << y_1 << y_2 \]

\[ T_0 \rightarrow T_1, \ T_1 \rightarrow T_2 \]

\[ T_2 \rightarrow T_3, \ T_1 \rightarrow T_3 \]
MVSR vs MCSR

MVSR
- Conflict graph $G(m)$
- Order function
- Multiversion serialization graph
- NP-complete to test

MCSR
- Multiversion conflict graph
- Test in polynomial time
Multiversion Conflicts

- Previously, the conflicts were of the form
  - rw
  - wr
  - ww
- For multiversion schedule, only rw is a conflict
  - Why?
    - ww is no conflict: everyone writes his own version
    - wr: commuting the pair is no problem (restricts the version choices for read that still render the correct schedule)
    - rw: commuting the pair creates a problem (adds more choices for reading)
Multiversion Conflicts

- Previously, the conflicts were of the form
  - rw
  - wr
  - ww
- For multiversion schedule, only rw is a conflict
- Why?
  - ww is no conflict: everyone writes his own version
  - wr: commuting the pair is no problem (restricts the version choices for read that still render the correct schedule)
  - rw: commuting the pair creates a problem (adds more choices for reading)
Multiversion Conflict Serializability

- MV Conflict reducibility
- MCSR: there is a serial monoversion history with the same ordering of multiversion conflicts
- Multiversion conflict graph: edges of the form $r_i(x_j) < w_k(x_k)$ for the same data item $x$
- MCSR: Multiversion conflict graph is acyclic
\[ r_i(x) \rightarrow r_i(x_k), \quad x_k \text{ has the largest timestamp before } t_i \]

\[ w_i(x): \text{ if too late } (ts(t_k) < ts(t_i) < ts(t_j) \text{ and there is } r_j(x_k)), \text{ reject and abort} \]

\[ \text{otherwise, } w_i(x) \rightarrow w_i(x_i) \]

\[ \text{delay the commit } c_i \text{ until all transactions } T_j \text{ that have written new versions of data items read by } T_i, \text{ have committed} \]
Example 1

\[ w_1(x) c_1 r_2(x) r_3(x) c_2 r_4(x) w_3(x) c_4 c_3 \]
Example 1

- $w_1(x)c_1r_2(x)r_3(x)c_2r_4(x)w_3(x)c_4c_3$
- $T_3$ aborted
Example 2

- \( r_1(x)r_2(x)r_3(y)w_2(x)w_1(y)c_1w_2(z)w_3(z)r_3(x)c_3r_2(y)c_2 \)
Example 2

- \( r_1(x) r_2(x) r_3(y) w_2(x) w_1(y) c_1 w_2(z) w_3(z) r_3(x) c_3 r_2(y) c_2 \)
- \( T_1 \text{ aborted, } T_3 \text{ waits for } T_2 \)
2V2PL

- every data item has exactly two versions (before update, after update)
- have to make sure at most one uncommitted version is present
- read the last committed version only
- locks for read, write, commit
- commit lock is set on every item that was written by transaction

<table>
<thead>
<tr>
<th></th>
<th>$rl(x)$</th>
<th>$wl(x)$</th>
<th>$cl(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$rl(x)$</td>
<td>√</td>
<td></td>
<td>−</td>
</tr>
<tr>
<td>$wl(x)$</td>
<td>√</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>$cl(x)$</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
</tbody>
</table>
Example 1

- $r_1(x)w_2(y)r_1(y)w_1(x)c_1r_3(y)r_3(z)w_3(z)w_2(x)c_2w_4(z)c_4c_3$

- $T_2$ has to wait before committing

- $T_4$ has to wait before writing $z$
Example 1

- \( r_1(x)w_2(y)r_1(y)w_1(x)c_1r_3(y)r_3(z)w_3(z)w_2(x)c_2w_4(z)c_4c_3 \)
- \( T_2 \) has to wait before committing
- \( T_4 \) has to wait before writing \( z \)
Example 2

\[
\begin{align*}
  r_1(x) r_2(x) r_3(y) w_2(x) w_1(y) c_1 w_2(z) w_3(z) r_3(x) c_3 r_2(y) c_2
\end{align*}
\]

▶ deadlock

11 / 1
Example 2

- $r_1(x)r_2(x)r_3(y)w_2(x)w_1(y)c_1w_2(z)w_3(z)r_3(x)c_3r_2(y)c_2$
- deadlock
Example 3

\[ r_1(x)w_1(x)r_2(x)w_2(y)r_1(y)w_2(x)c_2w_1(y)c_1 \]
Example 3

- $r_1(x)w_1(x)r_2(x)w_2(y)r_1(y)w_2(x)c_2w_1(y)c_1$
- deadlock
Credits: Dr. Andrey Gubichev, 2013
Homework

- Already uploaded to our website.