Relaxing Opacity in Pessimistic Transactional Memory

Konrad Siek and Paweł T. Wojciechowski
Poznań University of Technology
{konrad.siek,pawel.t.wojciechowski}@cs.put.edu.pl

13 X 2014

http://dsg.cs.put.poznan.pl
def thread:
    lock_a.acquire()
    lock_b.acquire()
    a = b
    lock_a.release()
    b = b + 1
    lock_b.release()

Advantages:

- ease of use on top
- efficient concurrency control under the hood
Pessimistic vs Optimistic TM

Optimistic approach

\[ \begin{align*}
T_1 & \left[ r(x)1, w(x)2 \right] \\
T_2 & \left[ r(x)1, w(x)2 \right] \xrightarrow{\text{w}(x)2} T_2' \left[ r(x)2, w(x)3 \right]
\end{align*} \]
Pessimistic vs Optimistic TM

Optimistic approach

\[ T_1 \left[ r(x)1, w(x)2 \right] \]
\[ T_2 \left[ r(x)1, w(x)2 \right] \xrightarrow{w(x)2} T_2' \left[ r(x)2, w(x)3 \right] \]

Pessimistic approach

\[ T_1 \left[ r(x)1, w(x)2 \right] \]
\[ T_2 \left[ r(x)2, w(x)3 \right] \]
Pessimistic vs Optimistic TM

Optimistic approach

\[
T_1 \begin{bmatrix} r(x)1, w(x)2 \end{bmatrix}
\]

\[
T_2 \begin{bmatrix} r(x)1, w(x)2 \end{bmatrix} \Rightarrow T_2' \begin{bmatrix} r(x)2, w(x)3 \end{bmatrix}
\]

Pessimistic approach

\[
T_1 \begin{bmatrix} r(x)1, w(x)2 \end{bmatrix}
\]

\[
T_2 \begin{bmatrix} r(x)2, w(x)3 \end{bmatrix}
\]

- Prevent aborts
- Tolerate high contention
- Safe for irrevocable operations
The joys of early release

Committing conflicting transactions

Early release on last use

\[
T_1 \left[ r(x)1, w(x)2, r(y)1, w(y)2 \right]
\]

\[
T_2 \left[ r(x)2, w(x)3 \right]
\]
The joys of early release

Committing conflicting transactions

Early release on last use

\[ T_1 \left[ r(x)_1, w(x)_2, r(y)_1, w(y)_2 \right] \]
\[ T_2 \left[ r(x)_2, w(x)_3 \right] \]

Performance boost:

![Graphs showing performance boost]
Manual aborts

Cascading abort in case of arbitrary abort

\[ T_1 \left[ r(x)1, w(x)2, r(y)1, w(y)2, \rightarrow \right. \]
\[ T_2 \left[ r(x)2, w(x)3 \rightarrow \rightarrow \right. \]

\(T_2\) observes an **inconsistent view** → broken invariants, segfaults, infinite looping, etc.
Safety

Allows reading from live transactions → not opaque

Precludes overwriting:

\[ T_i \begin{bmatrix} w(x)0, & w(x)1 \end{bmatrix} \]
\[ T_j \begin{bmatrix} r(x)0 & \rightarrow T'_j \begin{bmatrix} r(x)1, & w(x)2 \end{bmatrix} \]
Safety properties for TMs with early release

- Serializability
- Elastic Opacity
- Virtual World Consistency
- TMS1 & TMS2

- Recoverability
- Avoiding Cascading Aborts
- Strictness
- Rigorousness

serializability + recoverability

serializability + ACA

serializability

Last-use opacity

Opacity:
- Serializability
- Real-time order
- Consistency
  - read $x$ from a committed or commit-pending transaction
Last-use opacity

Opacity:
- Serializability
- Real-time order
- Consistency
  - read $x$ from a committed or commit-pending transaction

Last-use opacity
- Serializability
- Real-time order
- Recoverable last-use consistency
  - read $x$ from a committed or commit-pending transaction
  - or a transaction that will no longer use $x$
  - commit order preserves object access order
Characteristics

- Every LU opaque history is strict serializable, recoverable.
- Every opaque history is LU opaque.
- LU opacity prevents overwriting, allows cascading aborts.

Thank you