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

Distributed Systems Group

http://dsg.cs.put.poznan.pl

Software Transactional Memory

```
def thread:
    lock_a.acquire()
    lock_b.acquire()
    a = b
    lock_a.release()
    b = b + 1
    lock_b.release()
```

```
def thread:
    transaction.start()
    a = b
    b = b + 1
    transaction.commit()
```

Advantages:

- ease of use on top
- efficient concurrency control under the hood

Pessimistic vs Optimistic TM

Optimistic approach

$T_1 \; [\![\; r(x)1, w(x)2 \;]\!]$

$T_2 \; [\![\quad r(x)1, \quad \searrow w(x)2 \;\; \circlearrowleft \quad \rightarrow \quad T_2' \; [\![\; r(x)2, w(x)3 \;]\!]$

Pessimistic vs Optimistic TM

Optimistic approach

$T_1 \;[\![\; r(x)1, w(x)2 \;]\!]$

$T_2 \;[\![\quad r(x)1, \quad \searrow w(x)2 \;\circlearrowleft \quad \rightarrow \quad T_2' \;[\![\; r(x)2, w(x)3 \;]\!]$

Pessimistic approach

$T_1 \;[\![\; r(x)1, w(x)2 \;]\!]$

$T_2 \;[\![\qquad\qquad \searrow r(x)2, w(x)3 \;]\!]$

Pessimistic vs Optimistic TM

Optimistic approach

$T_1 \ [\![\ r(x)1, w(x)2 \ ]\!]$

$T_2 \ [\![\quad r(x)1, \searrow w(x)2 \ \circlearrowleft \quad \rightarrow \quad T_2' \ [\![\ r(x)2, w(x)3 \ ]\!]$

Pessimistic approach

$T_1 \ [\![\ r(x)1, w(x)2 \ ]\!]$

$T_2 \ [\![\qquad \searrow r(x)2, w(x)3 \ ]\!]$

- Prevent aborts
- Tolerate high contention
- Safe for irrevocable operations

The joys of early release

Committing conflicting transactions

Early release on last use

T_1 $[\![\ r(x)1, w(x)2, r(y)1, w(y)2\ ]\!]$

T_2 $[\![\ \searrow r(x)2, w(x)3\ ]\!]$

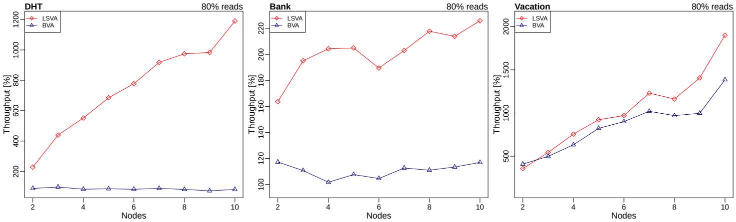
The joys of early release

Committing conflicting transactions

Early release on last use

$$T_1 \ [\![\ r(x)1, \color{red}{w(x)2}, r(y)1, w(y)2 \ ]\!]$$
$$T_2 \ [\![\qquad \qquad \searrow \color{red}{r(x)2}, w(x)3 \ ]\!]$$

Performance boost:



Siek, Wojciechowski. Atomic RMI: a Distributed Transactional Memory Framework. HLPP'14.

Manual aborts

Cascading abort in case of arbitrary abort

$$T_1 \; [\![\; r(x)1, w(x)2, r(y)1, w(y)2, \; \circlearrowleft$$
$$T_2 \; [\![\qquad \searrow r(x)2, w(x)3 \quad \searrow \circlearrowleft \ldots$$

T_2 observes an **inconsistent view** → broken invariants, segfaults, infinite looping, etc.

Safety

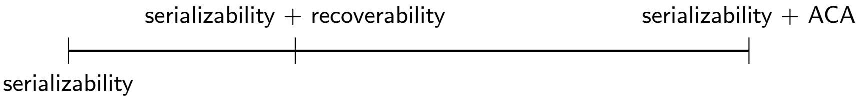
Allows reading from live transactions → **not opaque**

Precludes overwriting:

$$T_i \; [\![\; \mathrm{w}(x)0, \quad \mathrm{w}(x)1 \;]\!]$$
$$T_j \qquad [\![\; \searrow \mathrm{r}(x)0 \quad \searrow \circlearrowleft T_j' \; [\![\; \mathrm{r}(x)1, \mathrm{w}(x)2 \;]\!]$$

Safety properties for TMs with early release

- Serializability
- Elastic Opacity
- Virtual World Consistency
- TMS1 & TMS2
- Recoverability
- Avoiding Cascading Aborts
- Strictness
- Rigorousness



Siek, Wojciechowski. Zen and the Art of Concurrency Control: An Exploration of TM Safety Property Space with Early Release in Mind. WTTM'14.

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.

Dziuma, Fatourou, Kanellou. Consistency for Transactional Memory Computing. EATCS Distributed Column. 2014.

Thank you