Safety of Pessimistic Distributed Transactional Memory

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

19 XI 2013

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

Optimistic Approach

Run simultaneously in case there are no conflicts

$$\{x = 1\} \quad T_1 \left[\ r(x)1, w(x)2\ \right] \ \Big| \ T_2 \left[\ r(x)2, w(x)3\ \right] \quad \{x = 3\}$$

Optimistic Approach

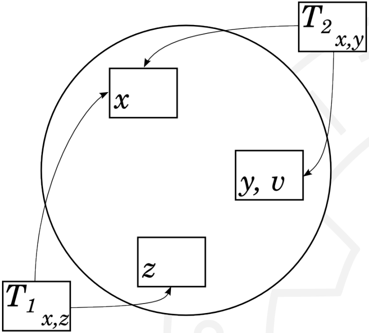
Run simultaneously in case there are no conflicts

$$\{x = 1\} \quad T_1 \left[\ r(x)1, w(x)2\ \right] \ \Big|\ T_2 \left[\ r(x)2, w(x)3\ \right] \quad \{x = 3\}$$

In case of conflicts, rollback and retry

$$\begin{array}{l} \{x = 1\} \quad T_1 \left[\ r(x)1, w(x)2\ \right] \\ \qquad\qquad \mid T_2 \left[\ r(x)1, w(x)2 \circlearrowleft \ldots T_2' \left[\ r(x)2, w(x)3\ \right] \quad \{x = 3\} \right. \end{array}$$

Distributed TM



Distributed Transactions

The Problem of Irrevocable Operations

Irrevocable operations $T_i[\ ..., ir, ...\ ]$

- do not operate on shared data
- visible effects on the system
- effect cannot be withdrawn (barring compensation)

Examples: network messages, system calls, I/O operations

The Problem of Irrevocable Operations

Irrevocable operations $T_i[\ ..., ir, ...\ ]$

- do not operate on shared data
- visible effects on the system
- effect cannot be withdrawn (barring compensation)

Examples: network messages, system calls, I/O operations

$$\{x = 1\}\ T_1\ [\ r(x)1, w(x)2\ ]$$
$$|\ T_2\ [\ r(x)1, ir, w(x)2 \circlearrowleft \ldots T_2'\ [\ r(x)2, ir, w(x)3\ ]\ \{x = 3\}$$

The Problem of Irrevocable Operations

Some workarounds

- forbid irrevocable operations
- buffer irrevocable operations and execute them on commit
- run irrevocable transactions one-at-a-time
- multiple versions of objects
- ignore the problem

Pessimistic Approach

Defer execution to prevent conflicts

$$\{x = 1\} \quad T_1 \left[\ r(x)1, w(x)2\ \right]$$

$$\searrow$$

$$\mid T_2 \left[\ \qquad\qquad r(x)2, w(x)3\ \right] \quad \{x = 3\}$$

Pessimistic Approach

Defer execution to prevent conflicts

$$\{x=1\} \quad T_1 \; [\; r(x)1, w(x)2 \;]$$
$$\searrow$$
$$|\; T_2 \; [\; \qquad r(x)2, w(x)3 \;] \quad \{x=3\}$$

No rollbacks/aborts, irrevocable operations are not re-run

$$\{x=1\} \quad T_1 \; [\; r(x)1, w(x)2 \;]$$
$$\searrow$$
$$|\; T_2 \; [\; \qquad r(x)2, ir, w(x)3 \;] \quad \{x=3\}$$

Pessimistic Approach

Defer execution to prevent conflicts

$$
\begin{array}{lll}
\{x = 1\} \quad T_1 \; [\; r(x)1, w(x)2 \;] & & \\
 & \searrow & \\
\qquad\qquad \mid T_2 \; [& & r(x)2, w(x)3 \;] \quad \{x = 3\}
\end{array}
$$

No rollbacks/aborts, irrevocable operations are not re-run

$$
\begin{array}{lll}
\{x = 1\} \quad T_1 \; [\; r(x)1, w(x)2 \;] & & \\
 & \searrow & \\
\qquad\qquad \mid T_2 \; [& & r(x)2, \color{red}{ir}, w(x)3 \;] \quad \{x = 3\}
\end{array}
$$

There are pros and cons to both approaches:

- high/low contention
- predictability of read sets and write sets

Rollbacks

However, rollback is still needed for

- expressiveness
- efficiency (i.e. limiting network traffic)

Rollbacks

However, rollback is still needed for

- expressiveness
- efficiency (i.e. limiting network traffic)
- necessary for fault tolerance

Supremum Versioning Algorithm

Transactions know which objects they use and how many times (suprema)

start:

lock all used variables
assign variable's next version to transaction release locks

access x:

wait until x is released by transaction with the previous version of x
access x
if last use of x: release x

commit:

release all variables

SVA Characteristics

Early release on last use

$$\{x = 1, y = 1\} \quad T_1 \; [\; r(x)1, w(x)2, r(y)1, w(y)2 \;]$$

$$\mid T_2 \; [\; r(x)2, w(x)3 \;] \quad \{x = 3, y = 2\}$$

No aborts, no issues with irrevocable operations

SVA + Rollback

start:

lock all used variables
assign variables's next version to transaction
release locks

access x:

wait until x is released by transaction with the previous version of x
if first use of x: make copy of x
access x
if last use of x: release x

commit:

wait until transaction with the previous version of x commits
if previous transaction rolls back: also roll back
release all variables

rollback:

wait until transaction with the previous version of x commits
restore all variables from copies and release them

SVA+R Characteristics

Cascading rollback

$$\{x = 1, y = 1\}\ T_1\ [\ r(x)1, w(x)2, r(y)1, w(y)2,\ \text{abort}$$

$$\mid T_2\ [\ r(x)2, w(x)3\ \circlearrowleft \dots$$

SVA+R Characteristics

Cascading rollback

$$\{x = 1, y = 1\}\ T_1\ [\ r(x)1, w(x)2, r(y)1, w(y)2,\ \text{abort}$$

$$\searrow \qquad \searrow$$

$$|\ T_2\ [\ \qquad r(x)2, w(x)3 \qquad \circlearrowleft \ldots$$

Cascading rollback with irrevocable operations

$$\{x = 1, y = 1\}\ T_1\ [\ r(x)1, w(x)2, r(y)1, w(y)2,\ \text{abort}$$

$$\searrow \qquad \searrow$$

$$|\ T_2\ [\ \qquad r(x)2, ir, w(x)3 \qquad \circlearrowleft \ldots$$

Fixing Cascading Rollback in SVA+R

Cascading rollback conditions in SVA:

- There are two or more transactions that access some variable x
- The first of those transactions releases x early
- Some younger transaction accesses x
- The first transaction aborts

Fixing Cascading Rollback in SVA+R

Cascading rollback conditions in SVA:

- There are two or more transactions that access some variable x
- The first of those transactions releases x early
- Some younger transaction accesses x
- The first transaction aborts

Transactions containing irrevocable operations cannot access variables that were released early (by transactions which may abort)

$$T_1 \; [\; r(x)1, w(x)2, r(y)1, w(y)2, \; \text{abort}$$

$$\searrow$$

$$|\; T_2 \; [\; \qquad r(x)2, ir, w(x)2 \;]$$

Properties

- **Serializability** (Safety)

  There exists some equivalent sequential history.

  - Exclusive access between first and last access to variable from version order.

Properties

- **Serializability** (Safety)

  There exists some equivalent sequential history.

  - Exclusive access between first and last access to variable from version order.

- **Strong Progressiveness** (Liveness)

  When two transactions conflict on some object, one of them will not be forced to abort.

  - Impossibile for all transactions to roll back due to cascading rollback conditions and version order

Properties

- **Serializability** (Safety)

  There exists some equivalent sequential history.

  - Exclusive access between first and last access to variable from version order.
- **Strong Progressiveness** (Liveness)

  When two transactions conflict on some object, one of them will not be forced to abort.

  - Impossibile for all transactions to roll back due to cascading rollback conditions and version order
- *Deadlock-freedom* (under some assumptions)
- Probably not *Livelock-freedom*
- Probably susceptible to *Parasitic Transactions*

Properties

- **Opacity** (Safety)

  There is some equivalent sequential history that preserves the real-time order of the transactional history and every transaction in the transactional history is legal in the sequential history.

Properties

- **Opacity** (Safety)

  There is some equivalent sequential history that preserves the real-time order of the transactional history and every transaction in the transactional history is legal in the sequential history.

  - Real-time order from version order

Properties

- **Opacity** (Safety)

  There is some equivalent sequential history that preserves the real-time order of the transactional history and every transaction in the transactional history is legal in the sequential history.

  - Real-time order from version order
  - Legality from access to committed variables

Properties

- **Opacity** (Safety)

  There is some equivalent sequential history that preserves the real-time order of the transactional history and every transaction in the transactional history is legal in the sequential history.

  - Real-time order from version order
  - Legality from access to committed variables
    or to uncommitted variables which are equivalent to committed variables

invariant: $x \neq 0$

Properties

- **Opacity** (Safety)

  There is some equivalent sequential history that preserves the real-time order of the transactional history and every transaction in the transactional history is legal in the sequential history.

  - Real-time order from version order
  - Legality from access to committed variables
    or to uncommitted variables which are equivalent to committed variables

invariant: $x \neq 0$

T_1 $[\ r(x)1, w(x)0, r(y)1, w(y)0,\ \text{abort}$

Properties

- **Opacity** (Safety)

  There is some equivalent sequential history that preserves the real-time order of the transactional history and every transaction in the transactional history is legal in the sequential history.

  - Real-time order from version order
  - Legality from access to committed variables or to uncommitted variables which are equivalent to committed variables

invariant: $x \neq 0$

$$T_1 \; [\; r(x)1, w(x)0, r(y)1, w(y)0, \; \text{abort}$$

$$\mid T_2 \; [\; r(x)0, \ldots$$

Properties

- **Opacity** (Safety)

  There is some equivalent sequential history that preserves the real-time order of the transactional history and every transaction in the transactional history is legal in the sequential history.

  - Real-time order from version order
  - Legality from access to committed variables or to uncommitted variables which are equivalent to committed variables

invariant: $x \neq 0$

$$T_1 \; [\; r(x)1, w(x)0, r(y)1, w(y)0, \text{ abort}$$

$$|\; T_2 \; [\; r(x)0, \ldots$$

Oops... Sorry SPAA'13.

Opaque SVA

start:

lock all used variables
assign variables's next version to transaction
release locks

access x:

wait until x is released by transaction with the previous version of x
if first use of x: make copy of x
access x
if last use of x and transaction does not abort: release x

commit:

release all variables

rollback:

restore all variables from copies and release them

OSVA Characteristics

Early release by non-aborting transactions

$$\{x = 1, y = 1\}\ T_1\ [\ r(x)1, w(x)2, r(y)1, w(y)2\ ]$$

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

OSVA Characteristics

Early release by non-aborting transactions

$$\{x = 1, y = 1\}\ T_1\ [\ r(x)1, w(x)2, r(y)1, w(y)2\ ]$$

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

No early release by aborting transactions

$$\{x = 1, y = 1\}\ T_1\ [\ r(x)1, w(x)2, r(y)1, w(y)2,\ \text{abort}$$

$$\mid T_2\ [\ \searrow r(x)1, w(x)2\ ]$$

OSVA Characteristics

Early release by non-aborting transactions

$\{x = 1, y = 1\}\ T_1\ [\ r(x)1, w(x)2, r(y)1, w(y)2\ ]$

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

No early release by aborting transactions

$\{x = 1, y = 1\}\ T_1\ [\ r(x)1, w(x)2, r(y)1, w(y)2,\ \text{abort}$

$|\ T_2\ [\ r(x)1, w(x)2\ ]$

No cascading rollback or issues with irrevocable operations

Opacity $>$ *what SVA guarantees*

Opacity $>$ *what SVA guarantees* $>$ Serializability

Opacity $>$ *what SVA guarantees* $>$ Serializability

What does SVA guarantee?

Opacity > *what SVA guarantees* > Serializability

What does SVA guarantee?

- serializability + real-time order

Opacity > *what SVA guarantees* > Serializability

What does SVA guarantee?

- serializability + real-time order
- transaction accesses a variable only after the preceding transaction used it for the last time

Opacity > *what SVA guarantees* > Serializability

What does SVA guarantee?

- serializability + real-time order
- transaction accesses a variable only after the preceding transaction used it for the last time
- if transaction accesses a variable which is later aborted, transaction aborts

Last-use Opacity

Opacity $>$ Last-use Opacity $>$ Serializability

Last-use Opacity

- serializability + real-time order
- transaction accesses a variable only after the preceding transaction used it for the last time
- if transaction accesses a variable which is later aborted, transaction aborts

Last-use Opacity

How is it useful?

- more than just serializability
- better parallelization than opacity
- problematic case not common in practice
- easy workaround

Last-use Opacity

How is it useful?

- more than just serializability
- better parallelization than opacity
- problematic case not common in practice
- easy workaround

```
@invariant(x!=0)
x := x - 1
if x == 0:   # last use of x
  rollback()
commit()
```

Last-use Opacity

How is it useful?

- more than just serializability
- better parallelization than opacity
- problematic case not common in practice
- easy workaround

```
@invariant(x!=0)
x := x - 1
if x == 0:   # last use of x
  rollback()
commit()
```

```
@invariant(x!=0)
tmp := x - 1
if tmp == 0:
  rollback()
x := tmp     # last use of x
commit()
```

Optimized SVA

SVA with the following optimizations:

- discriminate between reads and writes
- bufferred accesses
- buffer and release read-only variables
- defer writes in write-only transactions

OptSVA Buffered Access

- if first operation is a write, write to a buffer
- after last write operation on variable, release variable
- whenever a buffer is available, access buffer instead of variable

OptSVA Buffered Access

- if first operation is a write, write to a buffer
- after last write operation on variable, release variable
- whenever a buffer is available, access buffer instead of variable

$T_1 \quad [\ \mathrm{r}(x)0, \mathrm{w}(x)1\ ]$

$T_2 \quad [\ \mathrm{w}(x)2, \mathrm{w}(x)3, \mathrm{r}(x)3\ ]$

$T_3 \quad [\ \mathrm{r}(x)3, \mathrm{w}(x)4\ ]$

OptSVA Buffered Access

- if first operation is a write, write to a buffer
- after last write operation on variable, release variable
- whenever a buffer is available, access buffer instead of variable

$$T_1 \quad [\ \mathrm{r}(x)0, \mathrm{w}(x)1\ ]$$

$$T_2 \quad [\ \mathrm{w}(x)2, \mathrm{w}(x)3, \mathrm{r}(x)3\ ]$$

$$T_3 \quad [\ \mathrm{r}(x)3, \mathrm{w}(x)4\ ]$$

$$T_1 \quad [\ \mathrm{r}(x)0, \mathrm{w}(x)1\ ]$$

$$T_2 \quad [\ \mathrm{w}(\underline{x})2, \mathrm{w}(\underline{x})3, \{x \leftarrow \underline{x}\}, \mathrm{r}(\underline{x})3\ ]$$

$$T_3 \quad [\ \mathrm{r}(x)3, \mathrm{w}(x)4\ ]$$

OptSVA Read-only Variables

- if variable is read-only, read to buffer during start and release
- subsequently read from buffer instead of variable

OptSVA Read-only Variables

- if variable is read-only, read to buffer during start and release
- subsequently read from buffer instead of variable

$$T_1 \quad [\ \mathrm{r}(x)0, \mathrm{r}(x)0, \mathrm{w}(y)0\ ]$$

$$\searrow$$

$$T_2 \quad [\ \qquad \mathrm{r}(x)0, \mathrm{w}(x)1\ ]$$

OptSVA Read-only Variables

- if variable is read-only, read to buffer during start and release
- subsequently read from buffer instead of variable

$$T_1 \quad [\ \mathrm{r}(x)0, \mathrm{r}(x)0, \mathrm{w}(y)0\ ]$$

$$T_2 \quad [\ \mathrm{r}(x)0, \mathrm{w}(x)1\ ]$$

$$T_1 \quad [\ \{\underline{x} \leftarrow x\}, \mathrm{r}(\underline{x})0, \mathrm{r}(\underline{x})0, \mathrm{w}(y)0\ ]$$

$$T_2 \quad [\ \mathrm{r}(x)0, \mathrm{w}(x)1\ ]$$

OptSVA Write-only Transactions

- if all variables are write-only, operate on buffer without synchronization
- on commit get versions and update variables from buffer

OptSVA Write-only Transactions

- if all variables are write-only, operate on buffer without synchronization
- on commit get versions and update variables from buffer

T_1 $[\ \mathrm{r}(x)0, \mathrm{w}(x)1\ ]$

T_2 $[\ \mathrm{w}(x)2, \mathrm{w}(x)3\ ]$

T_3 $[\ \mathrm{r}(x)3\ ]$

OptSVA Write-only Transactions

- if all variables are write-only, operate on buffer without synchronization
- on commit get versions and update variables from buffer

T_1 $[\ \mathrm{r}(x)0, \mathrm{w}(x)1\ ]$

T_2 $[\ \mathrm{w}(x)2, \mathrm{w}(x)3\ ]$

T_3 $[\ \mathrm{r}(x)3\ ]$

T_1 $[\ \mathrm{r}(x)0, \mathrm{w}(x)1\ ]$

T_3 $[\ \mathrm{r}(x)1\ ]$

T_2 $\mathrm{w}(\underline{x})2, \mathrm{w}(\underline{x})3\ [\ \{x \leftarrow \underline{x}\}\ ]$

OptSVA Properties

- **Last-use Opacity** (Safety)

  Serializability + real-time order + access variable after last-use

  - SVA is Last-use Opaque
  - Every OptSVA history is a reduction of an SVA history

OptSVA Properties

- **Last-use Opacity** (Safety)

  Serializability + real-time order + access variable after last-use

  - SVA is Last-use Opaque
  - Every OptSVA history is a reduction of an SVA history

- **Optimality**

  Is OptSVA an optimal Last-use Opaque algorithm?

  - Moving any operation would break last-use opacity

Conclusions

Progress so far

- TM algorithms for distributed systems
- irrevocable operations and rollback in pessimistic TM
- solution to cascading rollback
- Opaque pessimistic TM algorithm
- Last-use Opacity
- Optimized pessimistic TM algorithm

Future Work

- Optimality of OptSVA
- Failure detection and fault tolerance
- Stronger progress properties

Related Papers:

Konrad Siek, Paweł T. Wojciechowski. *Brief Announcement: Towards a Fully-Articulated Pessimistic Distributed Transactional Memory*. In Proceedings of SPAA 2013: the 25th ACM Symposium on Parallelism in Algorithms and Architectures. July 2013.

Paweł T. Wojciechowski, Olivier Rütti and André Schiper. *SAMOA: A Framework for a Synchronisation-Augmented Microprotocol Approach.* In the Proceedings of IPDPS 2004: the 18th IEEE Parallel and Distributed Processing Symposium. April 2004.

Paweł T. Wojciechowski, Konrad Siek. *Pessimistic Distributed Transactional Memory.* Coming soon to a journal near you!

?