Rollbacks in Pessimistic Distributed Transactional Memory

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

14 VI 2013

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

Transaction Abstraction

Transaction:

$$T_i \; [\; op_1, \; op_2, \; ..., \; op_n \;]$$

where $op = \{ \; r(x)v, \; w(x)v, \; ... \; \}$

and x is some shared object

Commitment:

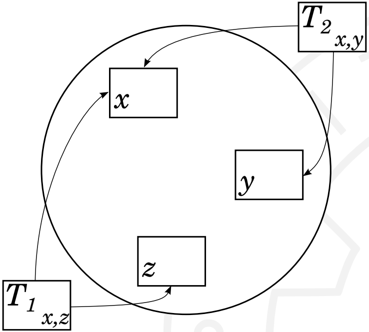
$$\{x = 1\} \quad T_i \; [\; w(x)2 \;] \quad \{x = 2\}$$

Rollback:

$$\{x = 1\} \quad T_i \; [\; w(x)2, \; \text{abort} \quad \{x = 1\}$$

$$\{x = 1\} \quad T_i \; [\; w(x)2, \; \text{retry} \quad \rightarrow \quad T_i' \; [\; w(x)2 \;] \quad \{x = 2\}$$

Distributed TM



Distributed Transactions

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

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

Conflict: two or more transactions access x and at least one of them writes to x.

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- effect cannot be withdrawn (barring compensation)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- effect cannot be withdrawn (barring compensation)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

Pessimistic Approach

Defer execution to prevent conflicts

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

$$\searrow$$

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

Pessimistic Approach

Defer execution to prevent conflicts

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

$$\searrow$$

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

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

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

$$\searrow$$

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

Rollbacks

Rollback is still needed for

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

Rollbacks

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 objects
assign object'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 objects

SVA Characteristics

Early release on last use

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

$$\searrow$$

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

No aborts, no issues with irrevocable operations

SVA + Rollback

start:

lock all used objects
assign object'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 objects

rollback:

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

SVA+R Characteristics

Cascading rollback

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

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

SVA+R Characteristics

Cascading rollback

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

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

Cascading rollback with irrevocable operations

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

$$|\ T_2\ [\qquad \searrow\ r(x)2, \color{red}{ir}, w(x)3 \qquad \searrow\ \circlearrowleft \ldots$$

Fixing Cascading Rollback in SVA+R

Cascading rollback conditions in SVA:

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

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

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

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

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 exclusive access to committed objects

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 exclusive access to committed objects
  - ... or uncommitted objects equivalent to committed objects

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 exclusive access to committed objects
  - ... or uncommitted objects equivalent to committed objects

- **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

- **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 exclusive access to committed objects
  - ... or uncommitted objects equivalent to committed objects

- **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*
- Probably not *Livelock-freedom*
- Probably susceptible to *Parasitic Transactions*

Conclusions

- Transactional Memory for Distributed Systems

Conclusions

- Transactional Memory for Distributed Systems
- Irrevocable operations and Pessimistic TMs

Conclusions

- Transactional Memory for Distributed Systems
- Irrevocable operations and Pessimistic TMs
- Incorporating Rollback into Pessimistic Distributed TM

Conclusions

- Transactional Memory for Distributed Systems
- Irrevocable operations and Pessimistic TMs
- Incorporating Rollback into Pessimistic Distributed TM
- Safety and Progress

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.

?