Towards a Fully-articulated Pessimistic Distributed Transacitonal Memory

Konrad Siek
konrad.siek@cs.put.edu.pl

14 V 2013

Distributed Systems Group

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()
```

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()
```

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

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:

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

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:

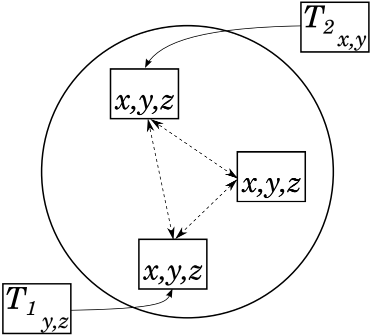
$$\{x = 1\} \quad T_i \; [\; w(x)2 \;] \quad \{x = 2\}$$

Rollback:

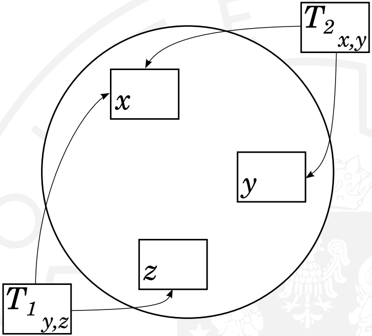
$$\{x = 1\} \quad T_i \; [\; w(x)2,\; \text{abort} \quad \{x = 1\}$$

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

Distributed TM



Replicated TM



Distributed TM

Optimistic Approach

Run simultaneously in case there are no conflicts

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

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

$$\{x = 1\} \quad T_1 \left[\ r(x)1, w(x)2\ \right]$$
$$\mid T_2 \left[\ r(x)1, w(x)2, \text{retry} \rightarrow T_2' \left[\ r(x)2, w(x)3\ \right] \quad \{x = 3\}\right.$$

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

$$\begin{aligned} \{x = 1\} \quad & T_1 \left[\ r(x)1, w(x)2\ \right] \\ & \mid T_2 \left[\ r(x)1, w(x)2, \text{retry} \rightarrow T_2' \left[\ r(x)2, w(x)3\ \right] \quad \{x = 3\} \right. \end{aligned}$$

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

$$\{x=1\}\ T_1\ [\ r(x)1, w(x)2\ ]$$
$$\mid T_2\ [\ r(x)1, ir, w(x)2, \text{retry} \rightarrow T_2'\ [\ r(x)2, ir, w(x)3\ ]\ \{x=3\}$$

The Problem of Irrevocable Operations

Workarounds

- forbid irrevocable operations

  Haskell

The Problem of Irrevocable Operations

Workarounds

- forbid irrevocable operations

    Haskell

- buffer irrevocable operations and execute them on commit

The Problem of Irrevocable Operations

Workarounds

- forbid irrevocable operations

  Haskell

- buffer irrevocable operations and execute them on commit
- run irrevocable transactions one-at-a-time

  A. Welc, B. Saha, and A.-R. Adl-Tabatabai
  Irrevocable transactions and their applications
  SPAA'08

The Problem of Irrevocable Operations

Workarounds

- forbid irrevocable operations

  Haskell

- buffer irrevocable operations and execute them on commit
- run irrevocable transactions one-at-a-time

  A. Welc, B. Saha, and A.-R. Adl-Tabatabai
  Irrevocable transactions and their applications
  SPAA'08

- multiple versions of objects

  H. Attiya and E. Hillel
  Single-version STMs can be multi-version permissive
  ICDCD'11

Pessimistic Approach

Defer execution to prevent conflicts

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

$$\searrow$$

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

Pessimistic Approach

Defer execution to prevent conflicts

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

Rollbacks are not forced, 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

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

Rollbacks are not forced, irrevocable operations are not re-run

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

There are pros and cons to both approaches:

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

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)

```
def thread:
    transaction.start()
    flight.reserved = MY_ID

    if not hotel.reserved:
        hotel.reserved = MY_ID
        transaction.commit()
    else:
        transaction.rollback()
```

Rollbacks

Rollback is still needed for

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

```
def thread:
    transaction.start()
    flight.reserved = MY_ID

    if not hotel.reserved:
        hotel.reserved = MY_ID
        transaction.commit()
    else:
        transaction.rollback()
```

```
def thread:
    transaction.start()
    flight_copy = copy(flight)
    flight.reserved = MY_ID

    if not hotel.reserved:
        hotel.reserved = MY_ID
        transaction.commit()
    else:
        flight = copy(flight_copy)
        del flight_copy
        transaction.commit()
```

Rollbacks

Rollback is still needed for

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

```
def thread:
    transaction.start()
    flight.reserved = MY_ID

    if not hotel.reserved:
        hotel.reserved = MY_ID
        transaction.commit()
    else:
        transaction.rollback()
```

```
def thread:
    transaction.start()
    flight_copy = copy(flight)
    flight.reserved = MY_ID

    if not hotel.reserved:
        hotel.reserved = MY_ID
        transaction.commit()
    else:
        flight = copy(flight_copy)
        del flight_copy
        transaction.commit()
```

- necessary for fault tolerance

Rollback and Pessimistic TM

Balancing correctness and rollback capability

- programmer-induced rollback
- never abort transactions with irrevocable operations

Maintaining efficiency and distribution

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

Transactions only block objects they use

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

SVA Characteristics

Transactions only block objects they use

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

Exclusive access (in order of versions)

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

$$\searrow$$

$$|\ T_2 \left[\qquad\qquad w(x)3\ \right] \quad \{x = 3\}$$

SVA Characteristics

Transactions only block objects they use

$$\{x = 1, y = 1\} \quad T_1 \; [\; w(x)2 \;] \; | \; T_2 \; [\; w(y)2 \;] \quad \{x = 2, y = 2\}$$

Exclusive access (in order of versions)

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

$$\searrow$$

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

Early release on last use

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

$$\searrow$$

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

SVA Characteristics

Transactions only block objects they use

$$\{x=1, y=1\} \quad T_1 \; [\; w(x)2 \;] \;|\; T_2 \; [\; w(y)2 \;] \quad \{x=2, y=2\}$$

Exclusive access (in order of versions)

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

$$\searrow$$

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

Early release on last use

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

$$\searrow$$

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

No rollbacks, 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

Wait for commit of previous transactions

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

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

SVA+R Characteristics

Wait for commit of previous transactions

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

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

Cascading rollback

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

$$\mid T_2\ \big[\ \searrow r(x)2, w(x)3\ \searrow \text{retry} \to \ldots$$

SVA+R Characteristics

Wait for commit of previous transactions

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

$$\searrow \qquad \searrow$$

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

Cascading rollback

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

$$\searrow \qquad \searrow$$

$$|\ T_2\ [\qquad r(x)2, w(x)3\qquad \text{retry} \rightarrow \ldots$$

Cascading rollback with irrevocable operations

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

$$\searrow \qquad \searrow$$

$$|\ T_2\ [\qquad r(x)2, ir, w(x)3\qquad \text{retry} \rightarrow \ldots$$

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 rolls back

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 rolls back

Transactions containing irrevocable operations cannot access objects that were released early

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 rolls back

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

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

$$\searrow$$

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

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.

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.

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.

  - Impossibility of all transactions rolling back from 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.

  - Impossibility of all transactions rolling back from cascading rollback conditions and version order

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

?