Atomic RMI: a Distributed Transactional Memory Framework

Konrad Siek and Paweł T. Wojciechowski

Poznań University of Technology

{konrad.siek,pawel.t.wojciechowski}@cs.put.edu.pl

3 VII 2014

Distributed Systems Group

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

Software Transactional Memory

Concurrency control is notoriously difficult:

- interaction between unrelated threads
- additional structural code
- deadlocks, livelocks, priority inversion

```
synchronized{aLock} {
    synchronized{bLock} {
        a = b
    }
    b = b + 1
}
```

Software Transactional Memory

Concurrency control is notoriously difficult:

- interaction between unrelated threads
- additional structural code
- deadlocks, livelocks, priority inversion

```
synchronized{aLock} {
    synchronized{bLock} {
        a = b
    }
    b = b + 1
}
```

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

Software Transactional Memory

Concurrency control is notoriously difficult:

- interaction between unrelated threads
- additional structural code
- deadlocks, livelocks, priority inversion

```
synchronized{aLock} {
    synchronized{bLock} {
        a = b
    }
    b = b + 1
}
```

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

Transactional Memory:

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

Transaction Abstraction

Transaction:

$$T_i \ \llbracket \ op_1, \ op_2, \ ..., \ op_n \ \rrbracket$$

where $op = \{ \ r(x)v, \ w(x)v, \ ... \ \}$
and x is some shared object

Commitment:

$$\{x = 1\} \quad T_i \ \llbracket \ w(x)2 \ \rrbracket \quad \{x = 2\}$$

Rollback:

$$\{x = 1\} \quad T_i \ \llbracket \ w(x)2, \ \circlearrowleft \quad \{x = 1\}$$

$$\{x = 1\} \quad T_i \ \llbracket \ w(x)2, \ \circlearrowleft \quad \rightarrow \quad T_i' \ \llbracket \ w(x)2 \ \rrbracket \quad \{x = 2\}$$

Transaction Abstraction

Transaction:

$$T_i \ \llbracket \ op_1, \ op_2, \ \ldots, \ op_n \ \rrbracket$$

where $op = \{ \ r(x)v, \ w(x)v, \ \ldots \ \}$
and x is some shared object

Commitment:

$$\{x = 1\} \quad T_i \ \llbracket \ w(x)2 \ \rrbracket \quad \{x = 2\}$$

Rollback:

$$\{x = 1\} \quad T_i \ \llbracket \ w(x)2, \ \circlearrowleft \quad \{x = 1\}$$

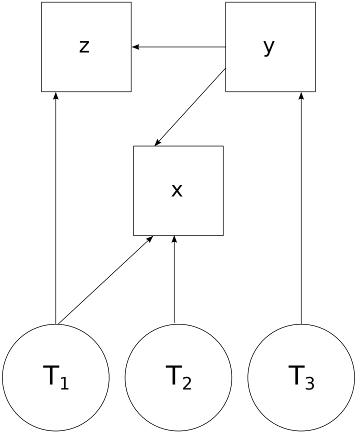
$$\{x = 1\} \quad T_i \ \llbracket \ w(x)2, \ \circlearrowleft \quad \rightarrow \quad T_i' \ \llbracket \ w(x)2 \ \rrbracket \quad \{x = 2\}$$

Conflict resolution:

$$\{x = 1\} \ T_1 \ \llbracket \ r(x)1, w(x)2 \ \rrbracket$$

$$\mid \ T_2 \ \llbracket \ r(x)1, w(x)2 \circlearrowleft \ldots T_2' \ \llbracket \ r(x)2, w(x)3 \ \rrbracket \ \{x = 3\}$$

Distributed Transactional Memory



Distributed Transactions

Problems with Optimistic TM

Optimistic TM relies on aborts:

Problems with Optimistic TM

Optimistic TM relies on aborts:

- low performance in high contention

$T_1 \; \llbracket \; r(x)1, w(x)2 \; \rrbracket$

Problems with Optimistic TM

Optimistic TM relies on aborts:

- low performance in high contention

$$
\begin{array}{l}
\quad T_1 \;\llbracket\; r(x)1, w(x)2 \;\rrbracket \\
\mid T_2 \;\;\llbracket\; r(x)1, w(x)2
\end{array}
$$

Problems with Optimistic TM

Optimistic TM relies on aborts:

- low performance in high contention

$$
\begin{array}{ll}
& T_1 \ \llbracket\ r(x)1, w(x)2\ \rrbracket \\
\mid & T_2 \ \llbracket\ r(x)1, w(x)2 \ \circlearrowleft \dots \ \llbracket\ r(x)2, w(x)3\ \rrbracket
\end{array}
$$

Problems with Optimistic TM

Optimistic TM relies on aborts:

- low performance in high contention

$$
\begin{array}{l}
\quad T_1 \ \llbracket \ r(x)1, w(x)2 \ \rrbracket \\
\mid T_2 \ \ \llbracket \ r(x)1, w(x)2 \ \ \circlearrowleft \ldots \ \llbracket \ r(x)2, w(x)3 \ \rrbracket \\
\mid T_3 \quad \llbracket \ r(x)1, w(x)2
\end{array}
$$

Problems with Optimistic TM

Optimistic TM relies on aborts:

- low performance in high contention

$$\begin{aligned}
&\; T_1 \;\llbracket\; r(x)1, w(x)2 \;\rrbracket \\
&\mid T_2 \;\;\llbracket\; r(x)1, w(x)2 \;\circlearrowleft \ldots \;\llbracket\; r(x)2, w(x)3 \;\rrbracket \\
&\mid T_3 \;\;\;\llbracket\; r(x)1, w(x)2 \;\circlearrowleft \ldots \;\llbracket\; r(x)2, w(x)3
\end{aligned}$$

Problems with Optimistic TM

Optimistic TM relies on aborts:

- low performance in high contention

$$
\begin{array}{ll}
 & T_1 \ \llbracket\ r(x)1, w(x)2\ \rrbracket \\
| & T_2 \ \llbracket\ r(x)1, w(x)2 \ \circlearrowleft \ldots \ \llbracket\ r(x)2, w(x)3\ \rrbracket \\
| & T_3 \ \llbracket\ r(x)1, w(x)2 \ \circlearrowleft \ldots \ \llbracket\ r(x)2, w(x)3 \ \circlearrowleft \ldots \ \llbracket\ r(x)3, w(x)4\ \rrbracket
\end{array}
$$

Problems with Optimistic TM

Optimistic TM relies on aborts:

- low performance in high contention

$$
\begin{array}{l}
\ \ T_1 \ \llbracket\ r(x)1, w(x)2\ \rrbracket \\
\mid T_2 \ \ \llbracket\ r(x)1, w(x)2 \ \circlearrowleft \dots \ \llbracket\ r(x)2, w(x)3\ \rrbracket \\
\mid T_3 \ \ \ \llbracket\ r(x)1, w(x)2 \ \circlearrowleft \dots \ \llbracket\ r(x)2, w(x)3 \ \circlearrowleft \dots \ \llbracket\ r(x)3, w(x)4\ \rrbracket
\end{array}
$$

- problems with irrevocable operations

Problems with Optimistic TM

Optimistic TM relies on aborts:

- low performance in high contention

$$
\begin{array}{l}
\;\; T_1 \; [\![\; r(x)1, w(x)2 \;]\!] \\
\mid T_2 \; [\![\; r(x)1, w(x)2 \; \circlearrowleft \ldots \; [\![\; r(x)2, w(x)3 \;]\!] \\
\mid T_3 \; [\![\; r(x)1, w(x)2 \; \circlearrowleft \ldots \; [\![\; r(x)2, w(x)3 \; \circlearrowleft \ldots \; [\![\; r(x)3, w(x)4 \;]\!]
\end{array}
$$

- problems with irrevocable operations: $T_i [\![\; ..., \color{red}{ir}, ... \;]\!]$
  - do not operate on shared data
  - have visible effects on the system
  - effects cannot be withdrawn (must be compensated)
  - examples: network communication, locks, system calls, I/O operations

Problems with Optimistic TM

Optimistic TM relies on aborts:

- low performance in high contention

$$
\begin{array}{l}
\quad T_1 \ \llbracket\ r(x)1, w(x)2\ \rrbracket \\
\mid T_2 \ \ \llbracket\ r(x)1, w(x)2 \ \circlearrowleft \ldots\ \llbracket\ r(x)2, w(x)3\ \rrbracket \\
\mid T_3 \ \ \ \llbracket\ r(x)1, w(x)2 \ \circlearrowleft \ldots\ \llbracket\ r(x)2, w(x)3 \ \circlearrowleft \ldots\ \llbracket\ r(x)3, w(x)4\ \rrbracket
\end{array}
$$

- problems with irrevocable operations: $T_i \llbracket\ \ldots, ir, \ldots\ \rrbracket$
  - do not operate on shared data
  - have visible effects on the system
  - effects cannot be withdrawn (must be compensated)
  - examples: network communication, locks, system calls, I/O operations

$$
\begin{array}{l}
\quad T_1 \ \llbracket\ r(x)1, w(x)2\ \rrbracket \\
\mid T_2 \ \llbracket\ r(x)1, ir, w(x)2
\end{array}
$$

Problems with Optimistic TM

Optimistic TM relies on aborts:

- low performance in high contention

$$
\begin{array}{l}
\quad T_1 \; [\![\; r(x)1, w(x)2 \;]\!] \\
\mid T_2 \; [\![\; r(x)1, w(x)2 \; \circlearrowleft \ldots \; [\![\; r(x)2, w(x)3 \;]\!] \\
\mid T_3 \; [\![\; r(x)1, w(x)2 \; \circlearrowleft \ldots \; [\![\; r(x)2, w(x)3 \; \circlearrowleft \ldots \; [\![\; r(x)3, w(x)4 \;]\!]
\end{array}
$$

- problems with irrevocable operations: $T_i [\![\; ..., {\color{red} ir}, ... \;]\!]$
  - do not operate on shared data
  - have visible effects on the system
  - effects cannot be withdrawn (must be compensated)
  - examples: network communication, locks, system calls, I/O operations

$$
\begin{array}{l}
\quad T_1 \; [\![\; r(x)1, w(x)2 \;]\!] \\
\mid T_2 \; [\![\; r(x)1, {\color{red} ir}, w(x)2 \circlearrowleft \ldots \; [\![\; r(x)2, {\color{red} ir}, w(x)3 \;]\!] \; \{x = 3\}
\end{array}
$$

Some Solutions (a very incomplete list)

Aborts in high contentions:

- contention managers

  W. N. Scherer III, M. L. Scott. *Advanced Contention Management for Dynamic Software Transactional Memory*. PODC'05.

- collision avoidance

  S. Dolev, D. Hendler, A. Suissa. *CAR-STM: scheduling-based collision avoidance and resolution for software transactional memory*. PODC'08.

Irrevocable operations:

- 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

  R. L. Bocchino, V. S. Adve, and B. L. Chamberlain. *Software transactional memory for large scale clusters*. PPoPP'08.
  H. Attiya and E. Hillel *Single-version STMs can be multi-version permissive* ICDCD'11.

Pessimistic TM

Optimistic TM:

- run simultaneously in case there are no conflicts
- rollback and retry if there are conflicts

Pessimistic TM

~~Optimistic TM:~~ Pessimistic TM:

- run simultaneously in case there are no conflicts
- rollback and retry if there are conflicts

Pessimistic TM

~~Optimistic TM:~~ Pessimistic TM:

- ~~run simultaneously in case there are no conflicts~~
  defer execution to prevent conflict
- rollback and retry if there are conflicts

Pessimistic TM

~~Optimistic TM:~~ Pessimistic TM:

- ~~run simultaneously in case there are no conflicts~~
  defer execution to prevent conflict
- ~~rollback and retry if there are conflicts~~
  avoid (most) forced aborts

Pessimistic TM

~~Optimistic TM:~~ Pessimistic TM:

- ~~run simultaneously in case there are no conflicts~~
  defer execution to prevent conflict
- ~~rollback and retry if there are conflicts~~
  avoid (most) forced aborts

$$
\begin{array}{lllll}
 & T_1 & [\![\ r(x)1, w(x)2\ ]\!] & & \\
| & T_2 & [\![& \searrow r(x)2, w(x)3\ ]\!] & \\
| & T_3 & [\![& & \searrow r(x)3, w(x)4\ ]\!]
\end{array}
$$

Pessimistic TM

~~Optimistic TM:~~ Pessimistic TM:

- ~~run simultaneously in case there are no conflicts~~
  defer execution to prevent conflict
- ~~rollback and retry if there are conflicts~~
  avoid (most) forced aborts

$$\begin{array}{lll} & T_1 & [\![\ r(x)1, w(x)2\ ]\!] \\ | & T_2 & [\![\qquad\qquad\qquad \searrow r(x)2, w(x)3\ ]\!] \\ | & T_3 & [\![\qquad\qquad\qquad\qquad\qquad\qquad\qquad \searrow r(x)3, w(x)4\ ]\!] \end{array}$$

- perform better in high contention
- easy handling irrevocable operations

Pessimistic TM

~~Optimistic TM:~~ Pessimistic TM:

- ~~run simultaneously in case there are no conflicts~~
  defer execution to prevent conflict
- ~~rollback and retry if there are conflicts~~
  avoid (most) forced aborts

$$
\begin{array}{l}
T_1 \ [\![\ r(x)1, w(x)2 \ ]\!] \\
\mid T_2 \ [\![\qquad\qquad\qquad \searrow r(x)2, w(x)3 \ ]\!] \\
\mid T_3 \ [\![\qquad\qquad\qquad\qquad\qquad\qquad\qquad \searrow r(x)3, w(x)4 \ ]\!]
\end{array}
$$

- perform better in high contention
- easy handling irrevocable operations

P. T. Wojciechowski. *Isolation-only Transactions by Typing and Versioning.* PPDP'05.
A. Matveev, N. Shavit. *Towards a Fully Pessimistic STM Model.* TRANSACT '12.

Supremum Versioning Algorithm (SVA)

Pessimistic approach

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

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

Supremum Versioning Algorithm (SVA)

Pessimistic approach

$$T_1 \ \llbracket \ r(x)1, w(x)2 \ \rrbracket$$

$$\mid T_2 \ \llbracket \qquad\qquad \searrow r(x)2, w(x)3 \ \rrbracket$$

Early release on last use

$$T_1 \ \llbracket \ r(x)1, w(x)2, r(y)1, w(y)2 \ \rrbracket$$

$$\mid T_2 \ \llbracket \qquad\qquad \searrow r(x)2, w(x)3 \ \rrbracket$$

Supremum Versioning Algorithm (SVA)

Pessimistic approach

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

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

Early release on last use

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

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

Wait for commit of previous transactions

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

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

Supremum Versioning Algorithm (SVA)

Pessimistic approach

$$T_1 \; [\![\; r(x)1, w(x)2 \;]\!]$$
$$| \; T_2 \; [\![\qquad \searrow r(x)2, w(x)3 \;]\!]$$

Early release on last use

$$T_1 \; [\![\; r(x)1, w(x)2, r(y)1, w(y)2 \;]\!]$$
$$| \; T_2 \; [\![\qquad \searrow r(x)2, w(x)3 \;]\!]$$

Wait for commit of previous transactions

$$T_1 \; [\![\; r(x)1, w(x)2, r(y)1, w(y)2 \;]\!]$$
$$| \; T_2 \; [\![\qquad \searrow r(x)2, w(x)3 \; \searrow]\!]$$

Manual rollback

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

Supremum Versioning Algorithm (SVA)

Pessimistic approach

$$T_1 \; [\![\; r(x)1, w(x)2 \;]\!]$$
$$\mid T_2 \; [\![\qquad\qquad \searrow r(x)2, w(x)3 \;]\!]$$

Early release on last use

$$T_1 \; [\![\; r(x)1, w(x)2, r(y)1, w(y)2 \;]\!]$$
$$\mid T_2 \; [\![\qquad\qquad \searrow r(x)2, w(x)3 \;]\!]$$

Wait for commit of previous transactions

$$T_1 \; [\![\; r(x)1, w(x)2, r(y)1, w(y)2 \;]\!]$$
$$\mid T_2 \; [\![\qquad\qquad \searrow r(x)2, w(x)3 \; \searrow]\!]$$

Manual rollback

$$T_1 \; [\![\; r(x)1, w(x)2, r(y)1, w(y)2, \; \circlearrowleft$$
$$\mid T_2 \; [\![\qquad\qquad \searrow r(x)2, w(x)3 \; \searrow \circlearrowleft \ldots$$
$$\mid T_3 \; [\![\qquad\qquad\qquad\qquad\qquad \searrow r(x)2, w(x)3 \;]\!]$$

Supremum Versioning Algorithm (SVA)

Pessimistic approach

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

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

Early release on last use

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

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

Wait for commit of previous transactions

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

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

Manual rollback

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

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

$\mid T_3 \; [\![\qquad \searrow r(x)2, w(x)3 \;]\!]$

Completely distributed (no leader, dispatcher, etc.)

Supremum Versioning Algorithm (SVA)

Pessimistic approach

$$T_1 \ [\![\ r(x)1, w(x)2 \ ]\!]$$
$$\mid T_2 \ [\![\qquad \searrow r(x)2, w(x)3 \ ]\!]$$

Early release on last use

$$T_1 \ [\![\ r(x)1, w(x)2, r(y)1, w(y)2 \ ]\!]$$
$$\mid T_2 \ [\![\qquad \searrow r(x)2, w(x)3 \ ]\!]$$

Wait for commit of previous transactions

$$T_1 \ [\![\ r(x)1, w(x)2, r(y)1, w(y)2 \ ]\!]$$
$$\mid T_2 \ [\![\qquad \searrow r(x)2, w(x)3 \ \searrow]\!]$$

Manual rollback

$$T_1 \ [\![\ r(x)1, w(x)2, r(y)1, w(y)2, \circlearrowleft$$
$$\mid T_2 \ [\![\qquad \searrow r(x)2, w(x)3 \ \searrow \circlearrowleft \ldots$$
$$\mid T_3 \ [\![\qquad \qquad \searrow r(x)2, w(x)3 \ ]\!]$$

Completely distributed (no leader, dispatcher, etc.)

K. Siek, P. T. Wojciechowski. *Brief announcement: Towards a Fully-Articulated Pessimistic Distributed Transactional Memory.* SPAA'13.

Supremum Versioning Algoritm (SVA)

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

rollback:

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

commit:

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

manual release x:

wait until x is released by transaction with the previous version of x
release x

Atomic RMI

Java RMI TM framework implementing SVA

- completely distributed
- rollback support
- early release
- irrevocable operations
- fault tolerance
- support for recurrency
- limited support for nesting

Atomic RMI

Java RMI TM framework implementing SVA

- completely distributed
- rollback support
- early release
- irrevocable operations

SVA

- fault tolerance
- support for recurrency
- limited support for nesting

Atomic RMI

Java RMI TM framework implementing SVA

- completely distributed
- rollback support
- early release
- irrevocable operations

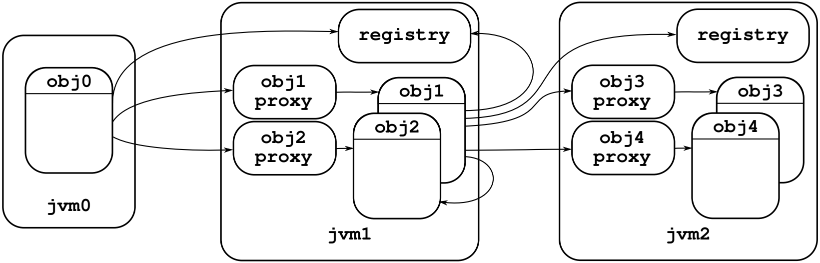
SVA

- fault tolerance
- support for recurrency
- limited support for nesting

Atomic RMI API

```
Transaction t = new Transaction(...);
a = t.accesses(registry.lookup("A"), 2);
b = t.accesses(registry.lookup("B"), 1);
t.start();

a.withdraw(100);
b.deposit(100);

if (a.getBalance() > 0)
    t.commit();
else
    t.rollback();
```

Atomic RMI architecture



Effecting Early Release

Early release:

- manual early release (`release`)
- automatic release from upper bounds (`accesses`)

Upper bounds can be derived by static analysis (and by other methods)

K. Siek, P. T. Wojciechowski. *A Formal Design of a Tool for Static Analysis of Upper Bounds on Object Calls in Java.* FMICS'12.

Why Use Upper Bounds?

```
t = new Transaction(...)
t.start();

for (i = 0; i < n; i++) {
  a.run();
  b.run();
}

// local operations
t.commit();
```

Why Use Upper Bounds?

```
t = new Transaction(...)
a = t.accesses(a);
b = t.accesses(b);
t.start();

for (i = 0; i < n; i++) {
  a.run();
  b.run();
}
t.release(a);
t.release(b);


// local operations
t.commit();
```

Why Use Upper Bounds?

```
t = new Transaction(...)
a = t.accesses(a);
b = t.accesses(b);
t.start();

for (i = 0; i < n; i++) {
  a.run();
  b.run();
}
t.release(a);
t.release(b);


// local operations
t.commit();
```

```
t = new Transaction(...)
a = t.accesses(a);
b = t.accesses(b);
t.start();

for (i = 0; i < n; i++) {
  a.run();
  if (i == n)
      a.release();
  b.run()
}
t.release(b);

// local operations
t.commit();
```

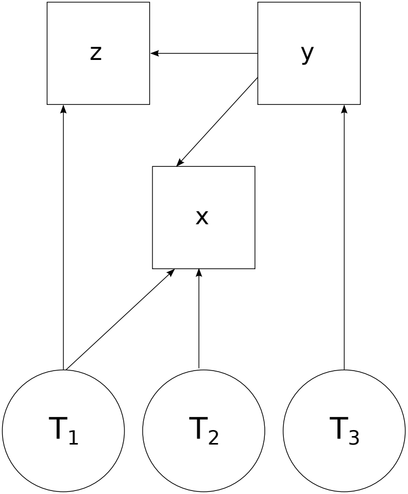
Why Use Upper Bounds?

```
t = new Transaction(...)
a = t.accesses(a, n);
b = t.accesses(b, n);
t.start();

for (i = 0; i < n; i++) {
  a.run(); // nth call: release
  b.run(); // nth call: release
}

// local operations
t.commit();
```

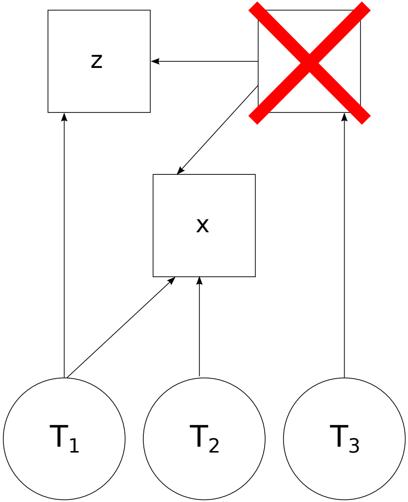
Why Use Manual Release?

```
t = new Transaction(...)
for (h : hotels)
    h = t.accesses(h, 2);
t.start();

for (h : hotels) {
    if (h.hasVacancies())
        h.bookRoom();
    else
        t.release(h);
}
t.commit();
```

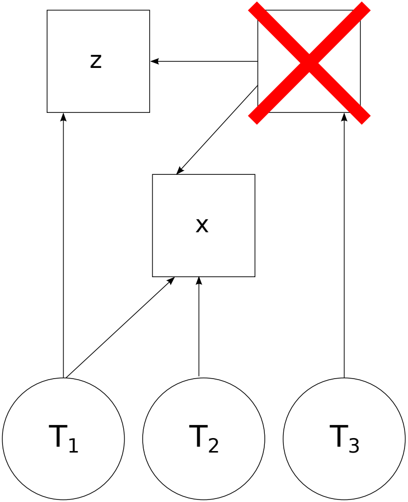
Fault Tolerance



Fault Tolerance



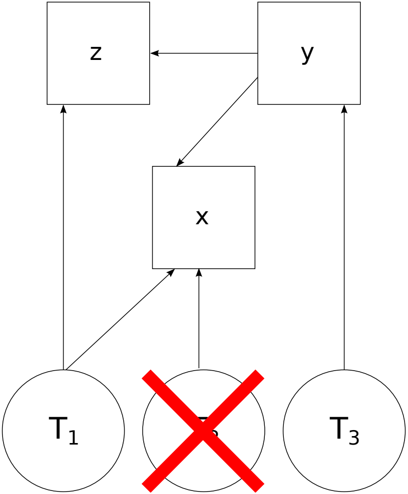
Fault Tolerance



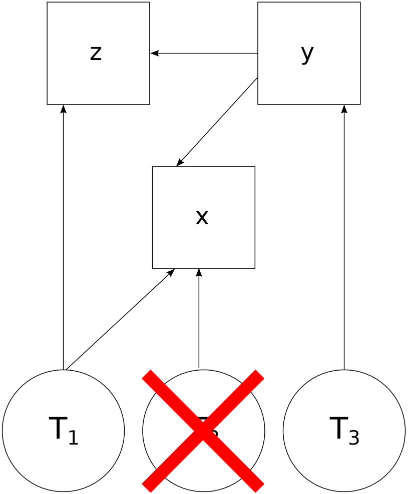
Shared object crash:

- timeout
- throw exception
- abort
  (or compensate)

Fault Tolerance



Fault Tolerance



Transaction crash:

- heartbeat
- revert object state
- update object version

Evaluation

Frameworks:

- Atomic RMI (SVA)
- Fine grained locking:
  - exclusion locks
  - R/W locks
- HyFlow (DTL2)

  M. M. Saad, B Ravindran. *HyFlow: A High Performance Distributed Transactional Memory Framework.* HPDC'11.

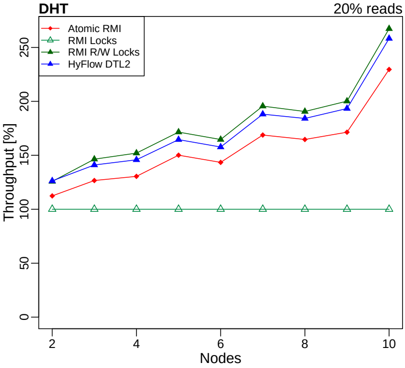
Benchmarks:

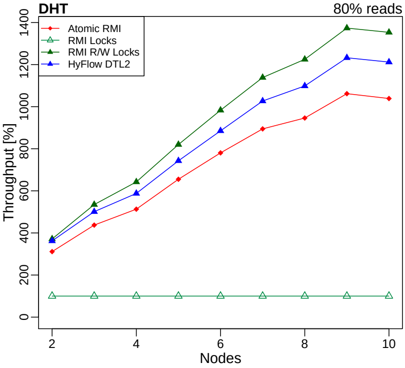
- Distributed Hash Table (DHT)
- Bank
- Loan
- Vacation

Environment:

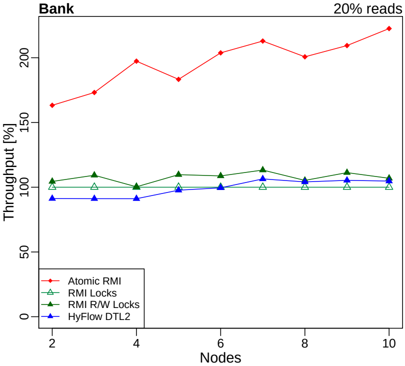
- $10 \times 2 \times$ quad-core Intel Xeon L3260 (2.83 GHz), 4 GB RAM
- OpenSUSE 13.1
- JREs (64 bit):
  - Open-JDK 1.7.0 51, IcedTea 2.4.4
  - Oracle 1.7.0_55-b13, Hotspot 24.55-b03
  - Oracle 1.8.0_05-b13, Hotspot 25.5-b02

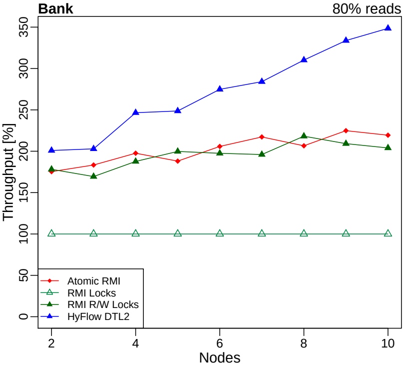
DHT Benchmark



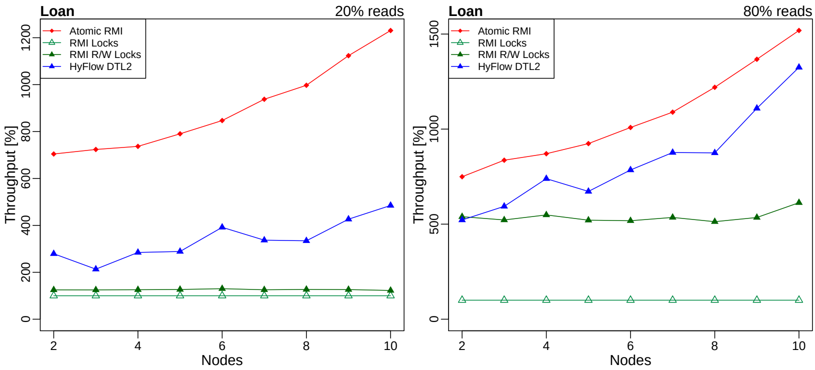


Bank Benchmark

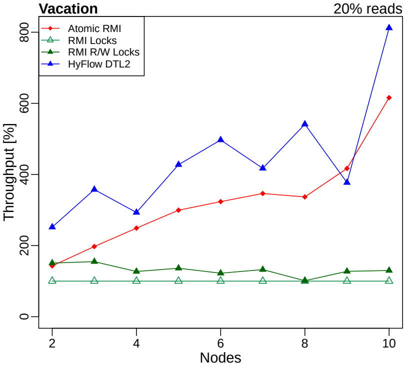


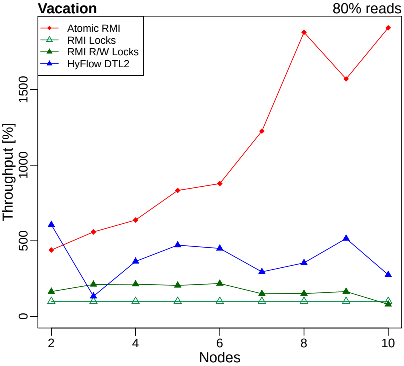


Loan Benchmark



Vacation Benchmark





Conclusions

In comparison to primitives, Atomic RMI

- performs better than exclusive locks
- performs as well or better than R/W locks (without read-only transaction support)

In comparison to HyFlow, performance of Atomic RMI depends on

- contention
  - good performance in high contention: early release, no aborts
  - higher overhead than HyFlow
- read/write operation ratio
  - no optimization of read-only transactions
  - early release parallelizes any operation

?