Time-Warp: Lightweight Abort Minimization in Transactional Memory

Nuno Diegues and Paolo Romano

ndiegues@gsd.inesc-id.pt
Transactional Memory

- Powerful abstraction for synchronization in shared memory
Transactional Memory

- Powerful abstraction for synchronization in shared memory
- Executions equivalent to serial ones
Transactional Memory

- Powerful abstraction for synchronization in shared memory
- Executions equivalent to serial ones
- Optimistic
Transactional Memory

- Powerful abstraction for synchronization in shared memory
- Executions equivalent to serial ones
- Optimistic
- Transactions may abort to ensure correctness
  - Typically, more aborts than needed
Problem

Linked List

head → A → D → E → ...

Nuno Diegues
Problem

Linked List

![Diagram of a linked list with nodes A, D, E and a highlighted node B.](image)

Insert B

T
Problem

Linked List

head → A → D → E → ...

U

remove E

T

insert B

...
Problem

Linked List

head → A → D → E → ...

U

remove E

T

insert B

RO

contains D?

...
Problem

Linked List

head → A → D → E → ...

B

U
remove E

T
insert B

RO
contains D?

Nuno Diegues
Problem

Linked List

U  remove E

T  insert B

RO  contains D?
Problem

Linked List

head → A → D → E → ...

U
remove E
read-set: { head.next, A.next, D.next }
write-set: { D.next }

T
insert B
read-set: { head.next, A.next }
write-set: { A.next }

RO
contains D?
read-set: { head.next, A.next }

4/27
Linked List

- head
- A
- D
- E

**Problem**

- remove E
  - read-set: { head.next, A.next, D.next }
  - write-set: { D.next }

- read
  - head.next: A
  - A.next: D
  - write
    - B.next = D
    - A.next = B

- contains D?
  - read-set: { head.next, A.next }

Nuno Diegues
Linked List

- remove E
  - read-set: { head.next, A.next, D.next }
  - write-set: { D.next }

- read
  - head.next: A
  - A.next: D
  - write
    - B.next = D

- read
  - head.next: A
  - A.next: D
  - write
    - A.next = B

Problem
Problem

Linked List

read
head.next: A
write
B.next = D

read
d. next = E
write
d. next = e.next

read
head.next: A
write
B.next = D

read
A.next: D
write
A.next = B

read
head.next: A
read
A.next: D

read
head.next: A
read
A.next: D

read
head.next: A
read
A.next: D

read
head.next: A
read
A.next: D

Nuno Diegues

4/27
Problem

Linked List

- Head
- A
- D
- E

- Read: A.next: D
- Read: A.next: D
- Write: A.next = B
- Read: A.next: D

Nuno Diegues
To guarantee a given correctness level, a TM aborts transactions.

Typical STMs use the following rule:
Problem

To guarantee a given correctness level, a TM aborts transactions.

Typical STMs use the following rule:

function $\text{commit(}\text{Transaction } tx\text{)}$:

\[
\text{for each } \langle \text{datum, version} \rangle \in tx.\text{readSet do}
\]
\[
\text{if not latestVersion(} \text{datum, version} \text{) then}
\]
\[
\text{abort(} tx \text{)}
\]
Problem

Condition:
Abort $T$ if its reads are not up-to-date when it attempts to commit.
Problem

Condition:
Abort $T$ if its reads are not up-to-date when it attempts to commit.

Serializability:
- Necessary condition
- But not sufficient
Condition:
Abort $T$ if its reads are not up-to-date when it attempts to commit.

Serializability:
- Necessary condition
- But not sufficient
- Deemed to be practical
  - without being overly conservative (eg., precluding all concurrency)
Objective

Linked List

- **read**
  - A.next: D

- **write**
  - A.next = B

- **read**
  - A.next: D

Nuno Diegues
Objective

Linked List

```
write
A.next = B
```

```
read
A.next: D
```

```
read
A.next: D
```

```
RO
```

```
U
```

```
T
```

```
read
A.next: D
```

```
write
A.next = B
```

Nuno Diegues

7/27
Objective

Lightweight minimization of spurious aborts:

- More restrictive abort condition
- Always read consistently
- Read-only transactions that *never* abort
Outline

- Problem and Motivation
- Objective
- Existing Work
- Time-Warp
- Evaluation
Existing Work

- Additional versions — fixed number in LSA [DISC06]
- MV-Permissiveness — as many as needed in JVSTM [PPoPP11]
- Permissiveness — AbortsAvoider [SPAA09]
- Interval-Based — AVSTM [DISC08]
Existing Work

Interval-Based:

- AVSTM [DISC08], TSTM [TPDS12], IR_VWC_P [ICA3PP11]
Existing Work

Interval-Based:

- AVSTM [DISC08], TSTM [TPDS12], IR_VWC_P [ICA3PP11]
- bounds for serialization order
- refined with transaction execution
- imposed by concurrent commits
- choose one value in the final interval
Disadvantages:

- A read-only transaction may abort
- An update transaction may abort due to *one miss*
- Scalability issues on commit
T1
read x  read y  write y

T2
read x  write x

rw
T1
read x
read y
write y
rw
T2
read x
write x
T1 serialization
T2
time
Decouple serialization order from commit order
T1
read x
read y
write y
rw
T2
read x
write x
T2 serialization
Time-warp commit
**Time-warp commit**
**Time-warp** commit — versions are produced with *past* version
Abort Condition

When can we **not** apply this idea?
Abort Condition

When can we not apply this idea?

Look out for a specific structure
Abort Condition

When can we **not** apply this idea?

Look out for a specific structure:

- Three transactions connected
  - a triad

![Diagram showing triad transactions with write and read operations]
Abort Condition

When can we **not** apply this idea?

Look out for a specific structure:

- Three transactions connected
  - a triad
- The link between all three
  - the pivot

![Diagram of Abort Condition](image-url)
Abort Condition

When can we **not** apply this idea?

Look out for a specific structure:

- Three transactions connected
  - a triad
- The link between all three
  - the pivot
- Abort if:
  - Completes a triad
  - Whose pivot time-warp commits
Abort Condition

Necessary condition (tighter)
Still cheap enough to check
How to Validate

Upon commit, transaction $T$ performs:

- Validate each write $k$
  - Detect if some concurrent $T'$ read $k$
  - If so, $T'$ witnessed that $T$ did not exist
  - We forbid $T$ from time-warping
- Validate each read $k$
  - Detect if some concurrent $T'$ committed a new version to $k$
  - If so, $T$ must time-warp
- Abort $T$ if it must time-warp, but cannot do so

Semi-visible readers scheme
- Know that some transaction read, not which
- Write transactions amortize the cost during read validation
How to Validate

Upon commit, transaction $T$ performs:

- Validate each write $k$

- Detect if some concurrent $T'$ read $k$
- If so, $T'$ witnessed that $T$ did not exist
- We forbid $T$ from time-warping

- Validate each read $k$
- Detect if some concurrent $T'$ committed a new version to $k$
- If so, $T$ must time-warp
- Abort $T$ if it must time-warp, but cannot do so

Semi-visible readers scheme

- Know that some transaction read, not which
- Write transactions amortize the cost during read validation
How to Validate

Upon commit, transaction \( T \) performs:

- Validate each write \( k \)
  - Detect if some concurrent \( T' \) read \( k \)
  - If so, \( T' \) witnessed that \( T \) did not exist
  - We forbid \( T \) from time-warping

- Validate each read \( k \)
  - Detect if some concurrent \( T' \) committed a new version to \( k \)
  - If so, \( T' \) must time-warp
  - Abort \( T \) if it must time-warp, but cannot do so

Semi-visible readers scheme
- Know that some transaction read, not which
- Write transactions amortize the cost during read validation
How to Validate

Upon commit, transaction $T$ performs:

- Validate each write $k$
  - Detect if some concurrent $T'$ read $k$
  - If so, $T'$ witnessed that $T$ did not exist
  - We forbid $T$ from time-warping

- Validate each read $k$
How to Validate

Upon commit, transaction $T$ performs:

- **Validate each write $k$**
  - Detect if some concurrent $T'$ read $k$
  - If so, $T'$ witnessed that $T$ did not exist
  - We forbid $T$ from time-warping

- **Validate each read $k$**
  - Detect if some concurrent $T'$ committed a new version to $k$
  - If so, $T$ must time-warp
How to Validate

Upon commit, transaction $T$ performs:

- Validate each write $k$
  - Detect if some concurrent $T'$ read $k$
  - If so, $T'$ witnessed that $T$ did not exist
  - We forbid $T$ from time-warping

- Validate each read $k$
  - Detect if some concurrent $T'$ committed a new version to $k$
  - If so, $T$ must time-warp

- Abort $T$ if it must time-warp, but cannot do so
How to Validate

Upon commit, transaction $T$ performs:

- Validate each write $k$
  - Detect if some concurrent $T'$ read $k$
  - If so, $T'$ witnessed that $T$ did not exist
  - We forbid $T$ from time-warping

- Validate each read $k$
  - Detect if some concurrent $T'$ committed a new version to $k$
  - If so, $T$ must time-warp

- Abort $T$ if it must time-warp, but cannot do so

- Semi-visible readers scheme
  - Know that some transaction read, not which
  - Write transactions amortize the cost during read validation
Evaluation Study

Wide variety of benchmarks:

- Micro-benchmarks: skip-list
- Macro-benchmarks: STMBench7 and STAMP
Evaluation Study

Wide variety of benchmarks:

- Micro-benchmarks: skip-list
- Macro-benchmarks: STMBench7 and STAMP

STMs spanning the design space:
Evaluation Study

Wide variety of benchmarks:
- Micro-benchmarks: skip-list
- Macro-benchmarks: STMBench7 and STAMP

STMs spanning the design space:
- NOrec: aimed at low thread count; single commit lock
Evaluation Study

Wide variety of benchmarks:
- Micro-benchmarks: skip-list
- Macro-benchmarks: STMBench7 and STAMP

STMs spanning the design space:
- NOrec: aimed at low thread count; single commit lock
- TL2: commit-time locking
Evaluation Study

Wide variety of benchmarks:
- Micro-benchmarks: skip-list
- Macro-benchmarks: STMBench7 and STAMP

STMs spanning the design space:
- NOrec: aimed at low thread count; single commit lock
- TL2: commit-time locking
- JVSTM: lock-free, multi-version
Evaluation Study

Wide variety of benchmarks:
- Micro-benchmarks: skip-list
- Macro-benchmarks: STMBench7 and STAMP

STMs spanning the design space:
- NOrec: aimed at low thread count; single commit lock
- TL2: commit-time locking
- JVSTM: lock-free, multi-version
- AVSTM: lock-free, probabilistic permissive
Wide variety of benchmarks:

- Micro-benchmarks: skip-list
- Macro-benchmarks: STMBench7 and STAMP

STMs spanning the design space:

- NOrec: aimed at low thread count; single commit lock
- TL2: commit-time locking
- JVSTM: lock-free, multi-version
- AVSTM: lock-free, probabilistic permissive
- TWM: lock-free, multi-version, time-warp
Evaluation Environment

4xAMD Opteron
6272: 64 cores
32GB RAM
Ubuntu 12.04
Oracle JVM 1.7

10 retries minimum
Skip-List
Evaluation: skip-list

- 100 thousand elements
- 25% modifications
- 1.8× speedup over AVSTM
Evaluation: skip-list

- Overhead of instrumentation
- Time-warp benefits from concurrency
- AVSTM lags behind

**Speedup - up to 8 threads**

![Graph showing throughput (1000 * txs/s) vs. threads]
Evaluation: skip-list

- Multi-version is not enough
- Time-Warp is similar to AVSTM
Evaluation: skip-list without contention

Speedup

Conflict-free workload
Unveil overheads and contention points
Evaluation: **skip-list** without contention

Overhead breakdown

- read
- commit
- readSet-val
- writeSet-val

<table>
<thead>
<tr>
<th>threads</th>
<th>1</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>JVSTM</td>
<td>1</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>64</td>
</tr>
<tr>
<td>TL2</td>
<td>1</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>64</td>
</tr>
<tr>
<td>NOrec</td>
<td>1</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>64</td>
</tr>
<tr>
<td>AVSTM</td>
<td>1</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>64</td>
</tr>
<tr>
<td>TWM</td>
<td>1</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>64</td>
</tr>
</tbody>
</table>
Evaluation: **skip-list** without contention

- Increased parallelism bottlenecks on NOrec commit
- TWM and AVSTM have most overheads
- TWM remains close to JVSTM, both lock-free and multi-versioned
STAMP
Evaluation: STAMP summary

Geometric Mean of Speedup

Nuno Diegues
Evaluation: **STAMP** summary

- Always better than AVSTM
- But difference is smaller with more threads
- Takes some concurrency to improve

**Geometric Mean of Speedup**

![Geometric Mean of Speedup Chart]

 JVSTM  TL2  NOrec  AVSTM  TWM

speedup of TWM relative to STM

1 thread  4 threads  8 threads  16 threads  32 threads  64 threads
Evaluation: **STAMP** Kmeans

![Speedup Kmeans](image)

- JVSTM
- TL2
- NOrec
- AVSTM
- TWM

Nuno Diegues 22/27
Evaluation: **STAMP** Vacation

![Graph showing speedup for different vacation STAMPs]
### Average per benchmark

<table>
<thead>
<tr>
<th>STM</th>
<th>Benchmark</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>genome</td>
<td>intruder</td>
<td>kmeans-l</td>
<td>kmeans-h</td>
<td>labyrinth</td>
<td>ssca2</td>
<td>vac-l</td>
<td>vac-h</td>
<td></td>
</tr>
<tr>
<td>TWM</td>
<td>3.8</td>
<td>3.8</td>
<td>1.4</td>
<td>4.2</td>
<td>8.8</td>
<td>10.5</td>
<td>6.4</td>
<td>17.8</td>
<td></td>
</tr>
<tr>
<td>JVSTM</td>
<td>15.4</td>
<td>3.2</td>
<td>1.6</td>
<td>4.9</td>
<td>12.3</td>
<td>11.3</td>
<td>12.1</td>
<td>41.1</td>
<td></td>
</tr>
<tr>
<td>TL2</td>
<td>12.1</td>
<td>4.8</td>
<td>3.8</td>
<td>3.4</td>
<td>13.8</td>
<td>11.7</td>
<td>10.0</td>
<td>41.4</td>
<td></td>
</tr>
<tr>
<td>NOrec</td>
<td>21.1</td>
<td>6.0</td>
<td>3.8</td>
<td>6.4</td>
<td>27.6</td>
<td>14.9</td>
<td>19.9</td>
<td>55.0</td>
<td></td>
</tr>
<tr>
<td>AVSTM</td>
<td>13.0</td>
<td>3.5</td>
<td>2.6</td>
<td>4.8</td>
<td>10.4</td>
<td>11.5</td>
<td>9.4</td>
<td>18.9</td>
<td></td>
</tr>
</tbody>
</table>
### Average per benchmark

<table>
<thead>
<tr>
<th>STM</th>
<th>TWM</th>
<th>JVSTM</th>
<th>TL2</th>
<th>NOrec</th>
<th>AVSTM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>STM</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Benchmark</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>genome</td>
<td>3.8</td>
<td>15.4</td>
<td>12.1</td>
<td>21.1</td>
<td>13.0</td>
</tr>
<tr>
<td>intruder</td>
<td>3.8</td>
<td>3.2</td>
<td>4.8</td>
<td>6.0</td>
<td>3.5</td>
</tr>
<tr>
<td>kmeans-l</td>
<td>1.4</td>
<td>1.6</td>
<td>3.8</td>
<td>3.8</td>
<td>2.6</td>
</tr>
<tr>
<td>kmeans-h</td>
<td>4.2</td>
<td>4.9</td>
<td>3.4</td>
<td>6.4</td>
<td>4.8</td>
</tr>
<tr>
<td>labyrinth</td>
<td>8.8</td>
<td>12.3</td>
<td>13.8</td>
<td>27.6</td>
<td>10.4</td>
</tr>
<tr>
<td>ssca2</td>
<td>10.5</td>
<td>11.3</td>
<td>11.7</td>
<td>14.9</td>
<td>11.5</td>
</tr>
<tr>
<td>vac-l</td>
<td>6.4</td>
<td>12.1</td>
<td>10.0</td>
<td>19.9</td>
<td>9.4</td>
</tr>
<tr>
<td>vac-h</td>
<td>17.8</td>
<td>41.1</td>
<td>41.4</td>
<td>55.0</td>
<td>18.9</td>
</tr>
</tbody>
</table>
## Average per threads

<table>
<thead>
<tr>
<th></th>
<th>Threads</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td>STM</td>
<td></td>
</tr>
<tr>
<td>TWM</td>
<td>1.2</td>
</tr>
<tr>
<td>JVSTM</td>
<td>1.8</td>
</tr>
<tr>
<td>TL2</td>
<td>2.6</td>
</tr>
<tr>
<td>NOrec</td>
<td>3.4</td>
</tr>
<tr>
<td>AVSTM</td>
<td>2.5</td>
</tr>
</tbody>
</table>
Summary

Time-Warp:
- Also allow write transactions to *commit in the past*
- Efficient validation rules that scale
- Average improvement 65% in high concurrency

Coming next:
- Adaptive validation
- Hybridize with Intel TSX
- Time-Warp in single-version STMs
Thank You

Questions?