An Innovative Approach for Achieving Composability in Concurrent Systems using Multi-Version Object Based STMs

Sandeep Kulkarni\textsuperscript{2} \quad \textbf{Sweta Kumari}\textsuperscript{1} \quad Sathya Peri\textsuperscript{1} \quad Archit Somani\textsuperscript{1}

\textsuperscript{1}Department of Computer Science Engineering, IIT Hyderabad
\textsuperscript{2}Department of Computer Science, Michigan State University
Outline

1. Introduction to STMs
2. Correctness Criteria of STMs
3. Problem with read-write STM
4. Object Based STMs
5. Motivation towards MV-OSTM
6. Correctness of MV-OSTM
7. Conclusion
8. Future Work
## Software Transactional Memory

### What is a transaction?
- Sequence of instructions executing in memory.
- Satisfying ACI
Introduction to STMs

Software Transactional Memory

What is a transaction?
- Sequence of instructions executing in memory.
- Satisfying ACI

What is Software Transactional Memory?
- A parallel programming paradigm
- Avoids concurrency overheads at programmers level
- Execute code optimistically
Introduction to STMs

Software Transactional Memory

What is a transaction?
- Sequence of instructions executing in memory.
- Satisfying ACI

What is Software Transactional Memory?
- A parallel programming paradigm
- Avoids concurrency overheads at programmers level
- Execute code optimistically

Methods of STMs:
- Read
- Write
- TryC
Illustration of STMs methods

Figure: Working of STM System
Illustration of STMs methods

Figure: Working of STM System
Illustration of STMs methods

Figure: Working of STM System
Illustration of STMs methods

Figure: Working of STM System
Figure: Working of STM System
Illustration of STMs methods

Figure: Working of STM System
Illustration of STMs methods

Figure: Working of STM System
Illustration of STMs methods

Figure: Working of STM System
Figure: Working of STM System
Illustration of STMs methods

Figure: Working of STM System
Illustration of STMs methods

Figure: Working of STM System
Illustration of STMs methods

Figure: Working of STM System
Outline

1. Introduction to STMs
2. Correctness Criteria of STMs
3. Problem with read-write STM
4. Object Based STMs
5. Motivation towards MV-OSTM
6. Correctness of MV-OSTM
7. Conclusion
8. Future Work
Correctness of STM System

Correcness criteria for STMs (Opacity)

A history $H$ is opaque if there exists a serial history $S$ s.t.
1. Operations of $H$ and $S$ are same
2. $S$ respects real time order $\preceq_{RT}$
3. $\forall$ trans($T_i$) $\in S$ are legal in $S$
Correctness of STM System

Correctness criteria for STMs (Opacity)

A history $H$ is opaque if there exists a serial history $S$ s.t.

1. Operations of $H$ and $S$ are same
2. $S$ respects real time order $\prec^R_{T_H}$ and
3. $\forall \text{trans}(T_i) \in S$ are legal in $S$
Correctness of STM System

Example of opacity

- H: $r_1(x, 0) w_2(x, 10) w_2(y, 10) C_2 r_1(y, A) A_1$

\[ T_1 \begin{array}{ccc}
\bullet & r_1(x, 0) & \bullet \\
\end{array} \quad T_2 \begin{array}{ccc}
\bullet & w_2(x, 10) & \bullet \\
\end{array} \]

\[ C_2 \begin{array}{ccc}
\bullet & w_2(y, 10) & \bullet \\
\end{array} \quad A_1 \begin{array}{ccc}
\bullet & r_1(y, A) & \bullet \\
\end{array} \]

**Figure:** Opaque History H
Correctness of STM System

Example of opacity

- $H: r_1(x, 0) w_2(x, 10) w_2(y, 10) C_2 r_1(y, A) A_1$

**Figure**: Opaque History $H$

**Figure**: Equivalent serial history $S$: $T_1, T_2$
Outline

1. Introduction to STMs
2. Correctness Criteria of STMs
3. Problem with read-write STM
4. Object Based STMs
5. Motivation towards MV-OSTM
6. Correctness of MV-OSTM
7. Conclusion
8. Future Work
Problem with read-write STM

Figure: A sample concurrent object

Figure: Tree Structure: conflicts are $(r_1(k_5), w_2(k_5))$ and $(w_2(k_5), r_1(k_5))$
Problem at read-write level

Figure: Tree Structure
Problem at read-write level

Figure: Tree Structure

Figure: Cycle (Not Serial)
• Object-based STM (OSTM) operate on higher level objects rather than primitive read & writes which act upon memory locations.
- **Object-based STMs (OSTM)** operate on higher level objects rather than primitive read & writes which act upon memory locations.

- OSTM model can be adapted:
  - OSTM for stacks may export `t_push`, `t_pop` & `t_peek`.
  - OSTM for sets may export `t_begin()`, `t_insert()`, `t_del()`, `t_lookup()` and `tryC()`.
**OSTM**

Execution at layer-1

**Figure:** A sample representing a OSTM object

**Figure:** Tree Structure: no conflict at Layer-1
Execution at layer-1

Figure: Pruned Tree
OSTM

Execution at layer-1

**Figure: Pruned Tree**

**Figure: Sequential Schedule**
OSTM
Execution at layer-1

Figure: Pruned Tree

Figure: Sequential Schedule

Figure: Serial History
Figure: Single version OSTM
Outline

1. Introduction to STMs
2. Correctness Criteria of STMs
3. Problem with read-write STM
4. Object Based STMs
5. Motivation towards MV-OSTM
6. Correctness of MV-OSTM
7. Conclusion
8. Future Work
Proposed Algorithm: MV-OSTM

Advantages of multi-version over single version OSTM

Figure: Single version OSTM
Proposed Algorithm: MV-OSTM

Advantages of multi-version over single version OSTM

**Figure: Single version OSTM**

**Figure: Multi-version OSTM (MV-OSTM): (T₁, T₂)**
Proposed Algorithm: MV-OSTM

Illustration of data structure

a) Underlying DS

\[ \begin{array}{ccccccc}
\infty & \rightarrow & k_1 & \rightarrow & k_5 & \rightarrow & k_7 & \rightarrow & k_8 & \rightarrow & +\infty \\
\vdots & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots
\end{array} \]
Proposed Algorithm : MV-OSTM

Illustration of data structure cont’d..

b) DS for maintaining Versions

VL (Version List)

RVL (Return Value List)

IIT Hyderbad
Proposed Algorithm: MV-OSTM

Lookup method

Figure: Lookup on key $k_1$ by $T_{13}$
Proposed Algorithm : MV-OSTM
Lookup method cont’d..

Figure: $T_{13}$ searching appropriate place in version list of $k_1$
Proposed Algorithm: MV-OSTM

Lookup method cont’d.

Figure: $T_{13}$ successfully added into $rvl_1$
Proposed Algorithm : MV-OSTM

tryC : Insert method

**Figure:** Insert a version of key $k_1$ by $T_{40}$
Proposed Algorithm: MV-OSTM

tryC: Insert method cont’d..

VL (Version List)

RVL (Return Value List)

Figure: $T_{40}$ searching appropriate place in version list of $k_1$
Proposed Algorithm : MV-OSTM

tryC : Insert method cont’d..

Figure: $T_{40}$ successfully created a new version of $k_1$
Figure: Insert a version of key $k_1$ by $T_{40}$
Proposed Algorithm: MV-OSTM

tryC: Insert method cont’d..

VL (Version List)

RVL (Return Value List)

Figure: $T_{40}$ searching appropriate place in version list of $k_1$
Proposed Algorithm : MV-OSTM

tryC : Insert method cont’d..

VL (Version List)

RVL (Return Value List)

Figure: Abort $T_{40}$ : $T_{45}$ committed before $T_{40}$
Correctness of MV-OSTM

Theorem

Any history \( H \) generated by MV-OSTM algorithm with a given version order \( \ll \), if \( \text{OPG}(H, \ll) \) is acyclic, then \( H \) is opaque.
1. Introduction to STMs
2. Correctness Criteria of STMs
3. Problem with read-write STM
4. Object Based STMs
5. Motivation towards MV-OSTM
6. Correctness of MV-OSTM
7. Conclusion
8. Future Work
Conclusion

- MV-OSTM is *opaque*.
Conclusion

- MV-OSTM is *opaque*.
- We have proposed a new STM as MV-OSTM which providing the greater concurrency in terms of the number of aborts with the help of multiple versions and composability.
Conclusion

- MV-OSTM is *opaque*.
- We have proposed a new STM as MV-OSTM which providing the greater concurrency in terms of the number of aborts with the help of multiple versions and composability.
- Lookup operation always succeeds.
Conclusion

- MV-OSTM is *opaque*.
- We have proposed a new STM as MV-OSTM which providing the greater concurrency in terms of the number of aborts with the help of multiple versions and composability.
- Lookup operation always succeeds.
- Delete operation is logically deletes, in that sense it’s lazy.
Conclusion

- MV-OSTM is *opaque*.
- We have proposed a new STM as MV-OSTM which providing the greater concurrency in terms of the number of aborts with the help of multiple versions and composability.
- Lookup operation always succeeds.
- Delete operation is logically deletes, in that sense it’s lazy.
- Transactions are *composable* [Harris et.al, 2005], [Ziv et.al, 2015].
Progress

Future Work

- Garbage Collection.
Progress

Future Work

- Garbage Collection.
- We will extend it for K-version MV-OSTM.
Garbage Collection.

We will extend it for K-version MV-OSTM.

We will implement our proposed protocol and compare the performance with existing Object-Based STMs [Hassan et.al, 2014].
Garbage Collection.

We will extend it for K-version MV-OSTM.

We will implement our proposed protocol and compare the performance with existing Object-Based STMs [Hassan et.al, 2014].

Nesting: open [Yang et.al, 2007] and close.
References


4. Kuznetsov, Petr and Peri, Sathya, Non-interference and Local Correctness in Transactional Memory, ICDCN, 2014


Thank You!

Any Questions?