भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

A Timestamp-Based STM protocol for Closed Nested Transactions with Opacity Guarantees

Nischay Ranjan, Rohit Kapoor, Sathya Peri

Indian Institute of Technology
Hyderabad, India

Oct 2025

Contents

Introduction to STMs

Overview

- STM facilitates synchronization and communication among concurrent processes using shared memory.

Overview

- STM facilitates synchronization and communication among concurrent processes using shared memory.
- Transactions in STM provide a means for executing code atomically, ensuring consistency even in the presence of other transactions.

Overview

- STM facilitates synchronization and communication among concurrent processes using shared memory.
- Transactions in STM provide a means for executing code atomically, ensuring consistency even in the presence of other transactions.
- A transaction appears indivisible during execution, ensuring its operations are performed **atomically**.

Transactional Objects

- Transactions operate on t-objects, representing shared data in the memory.

Transactional Objects

- Transactions operate on t-objects, representing shared data in the memory.
- Any number of memory operations can be invoked on t-objects within a transaction.

Transactional Outcomes

- If a transaction successfully executes to completion, it is considered a committed transaction.

Transactional Outcomes

- If a transaction successfully executes to completion, it is considered a committed transaction.
- Updates made by committed transactions are visible to others transactions, ensuring **consistency.**

Transactional Outcomes

- If a transaction successfully executes to completion, it is considered a committed transaction.
- Updates made by committed transactions are visible to others transactions, ensuring **consistency.**
- If a transaction encounters an issue or fails to complete successfully, it is aborted.

Transactional Outcomes

- If a transaction successfully executes to completion, it is considered a committed transaction.
- Updates made by committed transactions are visible to others transactions, ensuring **consistency.**
- If a transaction encounters an issue or fails to complete successfully, it is aborted.
- Changes proposed by aborted transactions are denied, preventing their visibility to other transactions.

STMs Implementation

- The operations of a transaction are divided into three phases.

STMs Implementation

- The operations of a transaction are divided into three phases.
- **Phase 1:** Local read/write phase
  - Execute transaction with writes into *private buffer*.

STMs Implementation

- The operations of a transaction are divided into three phases.
- **Phase 1:** Local read/write phase
  - Execute transaction with writes into *private buffer*.
- **Phase 2:** Validation phase
  - Tests whether reads and writes form a consistent view of memory.
  - Tests for conflicts.

STMs Implementation

- The operations of a transaction are divided into three phases.
- **Phase 1:** Local read/write phase
  - Execute transaction with writes into *private buffer*.
- **Phase 2:** Validation phase
  - Tests whether reads and writes form a consistent view of memory.
  - Tests for conflicts.
- **Phase 3:** Commit or Abort Phase
  - Depending on validation phase, the transaction will either commit or abort.
  - If committed, the local writes are performed into the meomory.

Illustration

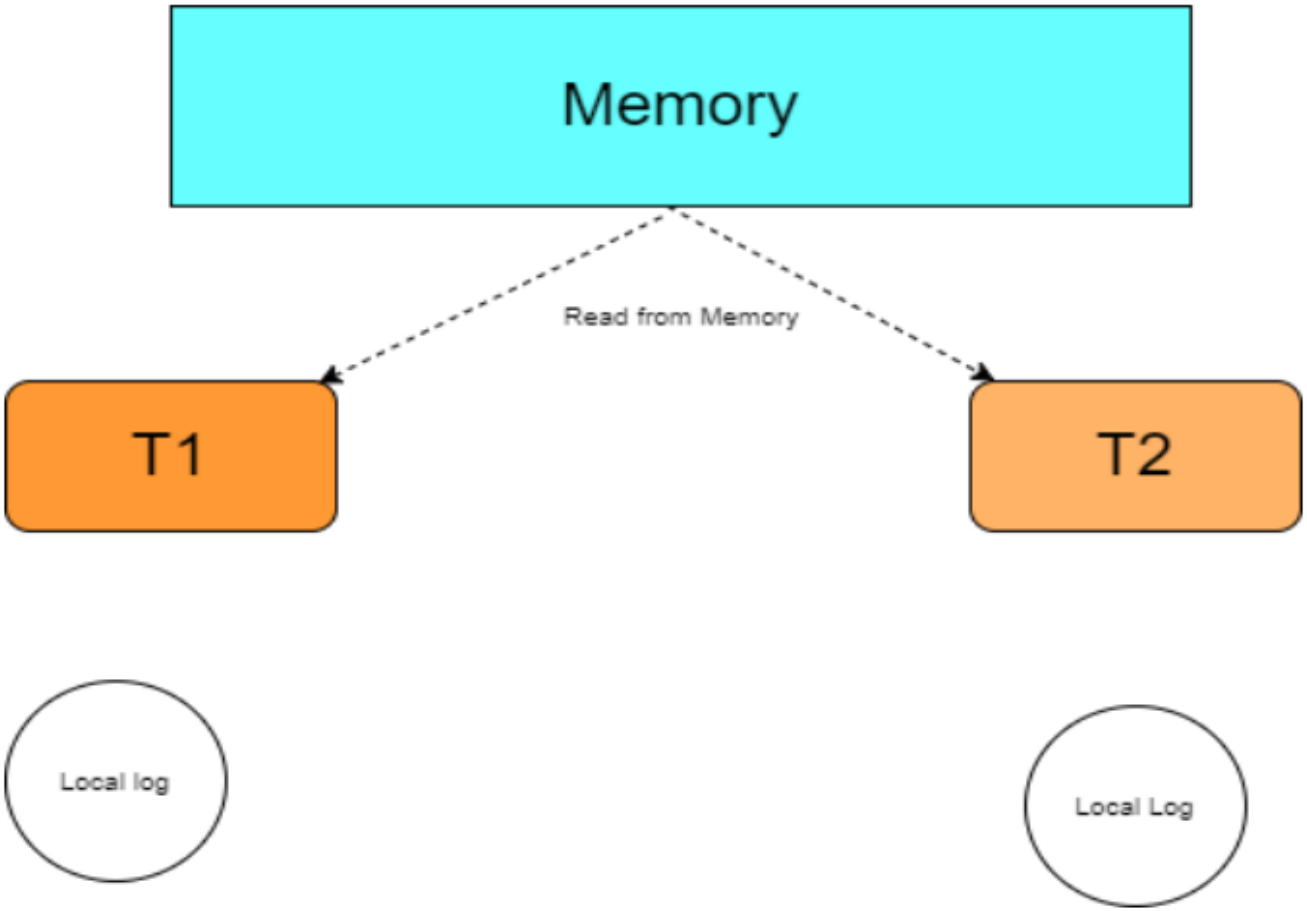


Figure: Transaction reading from shared Memory

Illustration Cont'd..

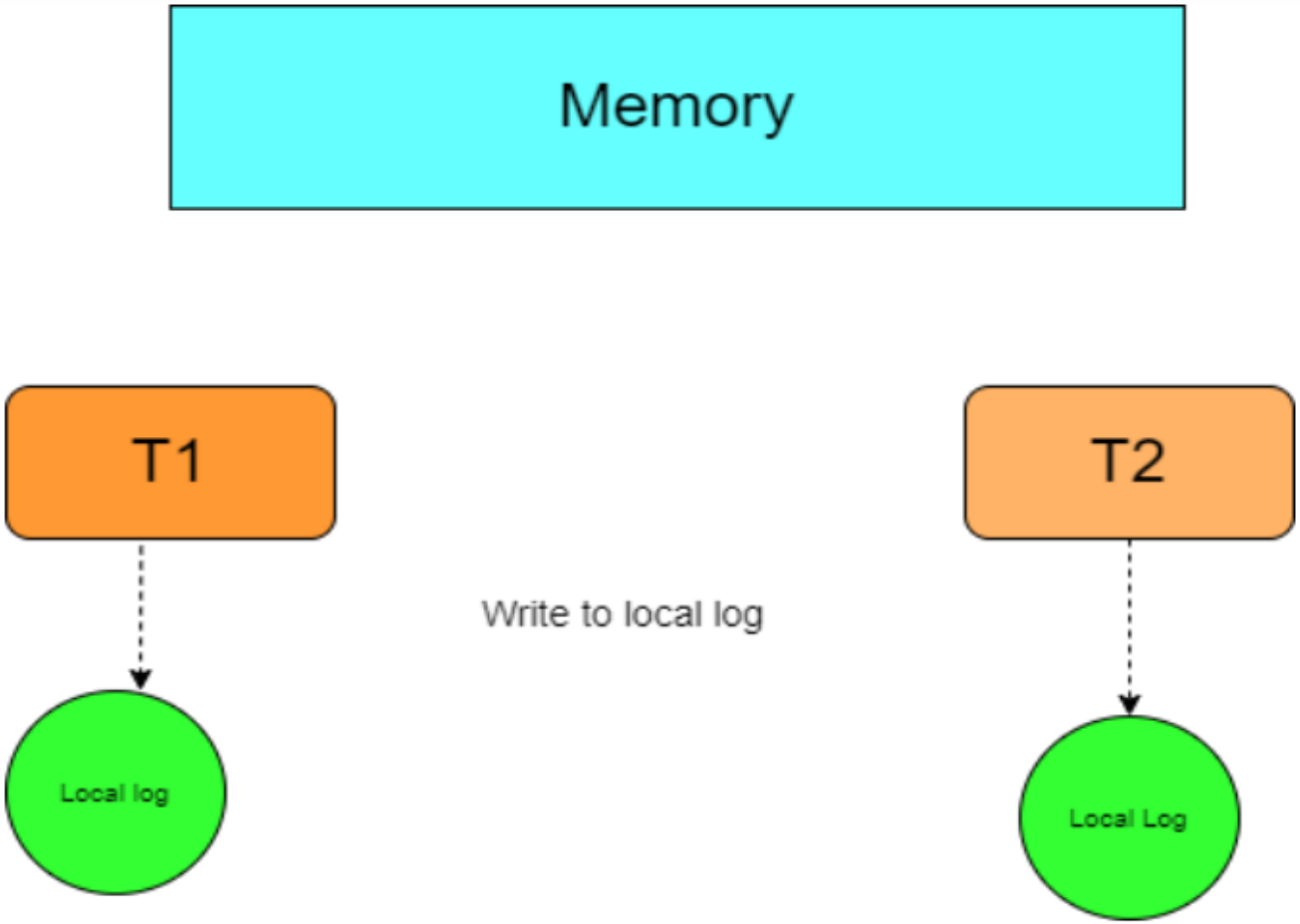


Figure: Writing into local Buffer

Illustration Cont'd..

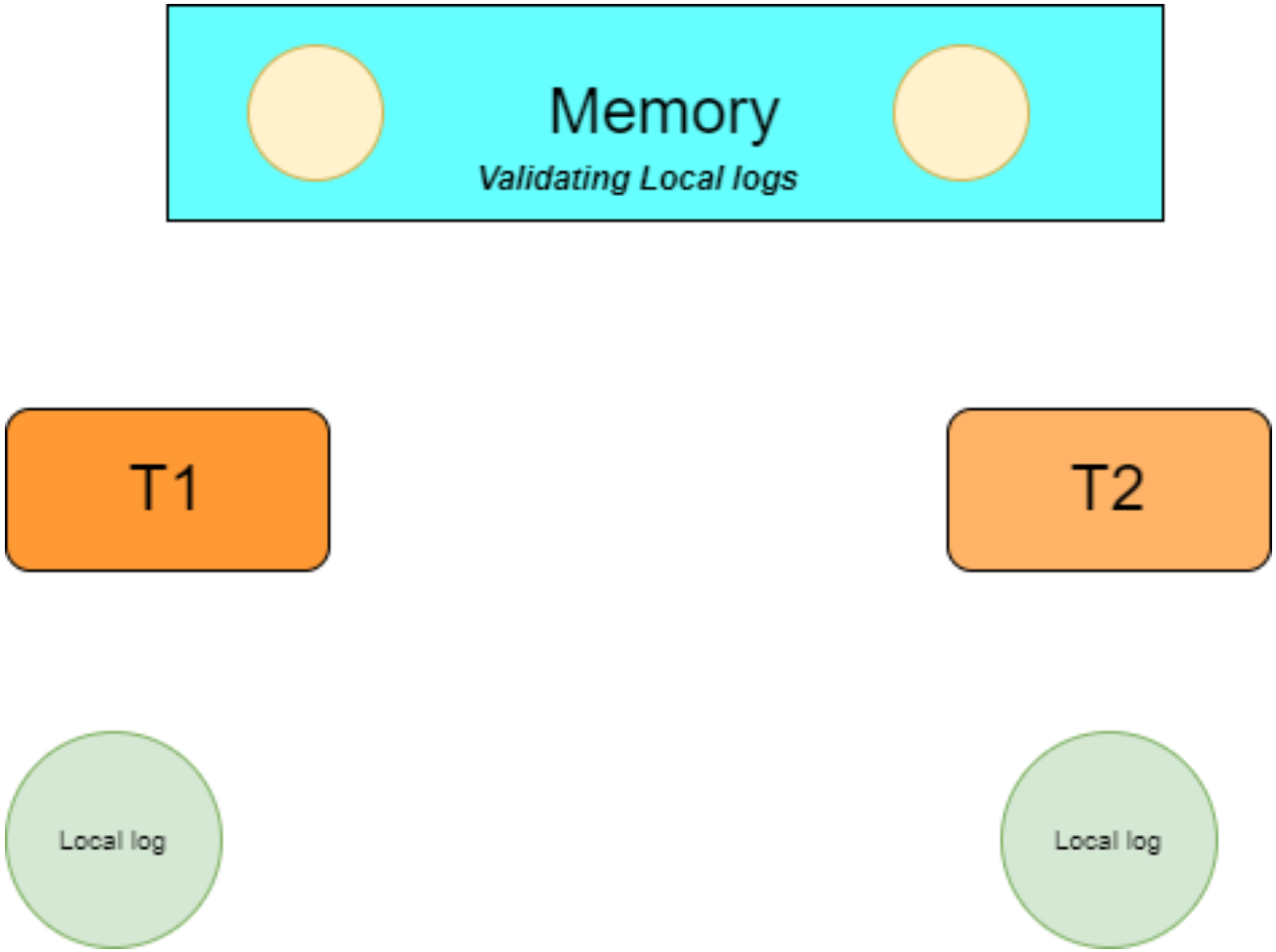


Figure: Validating local Buffers

Illustration Cont'd..

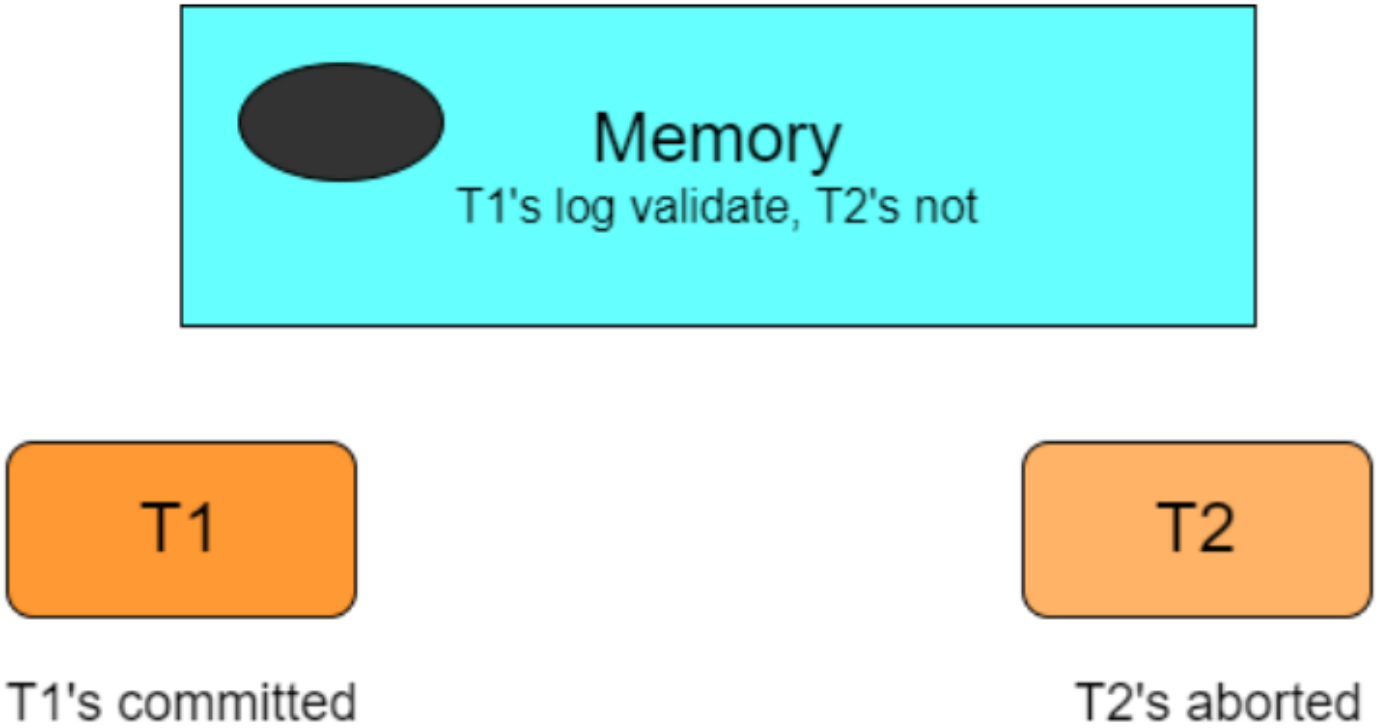


Figure: Buffers validated

Nesting of Transactions

A transaction is Nested

- If it invokes other transactions.

Nesting of Transactions

A transaction is Nested

- If it invokes other transactions.

Why Nesting?

- To aid software composition.

Nesting of Transactions

A transaction is Nested

- If it invokes other transactions.

Why Nesting?

- To aid software composition.
- To reduce the vulnerability period of long running transaction.

The Challenge: Correctness (Opacity)

Opacity: The Gold Standard for STM

- **Serializability:** Transactions appear to execute in some serial order.
- **Abort Consistency:** Even aborted transactions must see a consistent state.

The Gap in Literature

Existing nested STM protocols **lack a theoretical correctness guaranteeing opacity.**

Correctness of Close Nested STMs

Issues with Nested Transactions

- In Nested Transactions
  - Aborted transaction can have committed sub-transactions
  - And committed transactions can have aborted sub-transactions

[a] "Correctness of Concurrent Executions of Closed Nested Transactions in Transactional Memory Systems" , Sathya Peri and K.Vidyasankar. Theoretical Computer Science, 496:125-153, 2013

Correctness of Close Nested STMs

Issues with Nested Transactions

- In Nested Transactions
  - Aborted transaction can have committed sub-transactions
  - And committed transactions can have aborted sub-transactions
- To extend Opacity's definition to nested transactions,
  - Need some mechanism to treat aborted transaction
  - Need to verify membership in polynomial time

[a] "Correctness of Concurrent Executions of Closed Nested Transactions in Transactional Memory Systems" , Sathya Peri and K.Vidyasankar. Theoretical Computer Science, 496:125-153, 2013

Correctness of Close Nested STMs

Issues with Nested Transactions

- In Nested Transactions
  - Aborted transaction can have committed sub-transactions
  - And committed transactions can have aborted sub-transactions
- To extend Opacity's definition to nested transactions,
  - Need some mechanism to treat aborted transaction
  - Need to verify membership in polynomial time
- **Peri et al formally extended opacity to nested transactions and identified conflict-preserving schedules i.e (CP-CNO) that maintain opacity under closed nesting.** [a]

[a] "Correctness of Concurrent Executions of Closed Nested Transactions in Transactional Memory Systems" , Sathya Peri and K.Vidyasankar. Theoretical Computer Science, 496:125-153, 2013

Our Contribution: NestedBTO

A novel STM protocol for **Closed Nested Transactions**.

Key Features

1. **Formal Opacity Guarantee:** Proven correct for arbitrary nesting depth.
2. **Parallel Execution:** Child transactions can run concurrently.
3. **Timestamp-Based:** Uses lexicographic transaction IDs for conflict detection.

System Models

Transactional Semantics

- We presume a system of *n* asynchronous process/threads, p_1, p_2, ... p_n which will access a collection of *transactional objects* via atomic transactions through *thread pool*.

Transactional Semantics

- We presume a system of *n* asynchronous process/threads, p_1, p_2, ... p_n which will access a collection of *transactional objects* via atomic transactions through *thread pool*.
- Each transaction will have a unique identifier.

Transactional Semantics

- We presume a system of *n* asynchronous process/threads, p_1, p_2, ... p_n which will access a collection of *transactional objects* via atomic transactions through *thread pool*.
- Each transaction will have a unique identifier.
- Each transaction can invoke multiple subtransactions and any number of memory operations.

Transactional Semantics

- We presume a system of *n* asynchronous process/threads, p_1, p_2, ... p_n which will access a collection of *transactional objects* via atomic transactions through *thread pool*.
- Each transaction will have a unique identifier.
- Each transaction can invoke multiple subtransactions and any number of memory operations.
- Each transaction t_i will be assigned a unique timestamp ts(t_i).
  - t_0 will be the root transaction.
  - The subtransactions of the transaction t_i will have ids like t_{i1}, t_{i2} etc..

Transactional Semantics

- We presume a system of *n* asynchronous process/threads, p_1, p_2, ... p_n which will access a collection of *transactional objects* via atomic transactions through *thread pool*.
- Each transaction will have a unique identifier.
- Each transaction can invoke multiple subtransactions and any number of memory operations.
- Each transaction t_i will be assigned a unique timestamp ts(t_i).
  - t_0 will be the root transaction.
  - The subtransactions of the transaction t_i will have ids like t_{i1}, t_{i2} etc..

Basic Algorithm

Data Structures

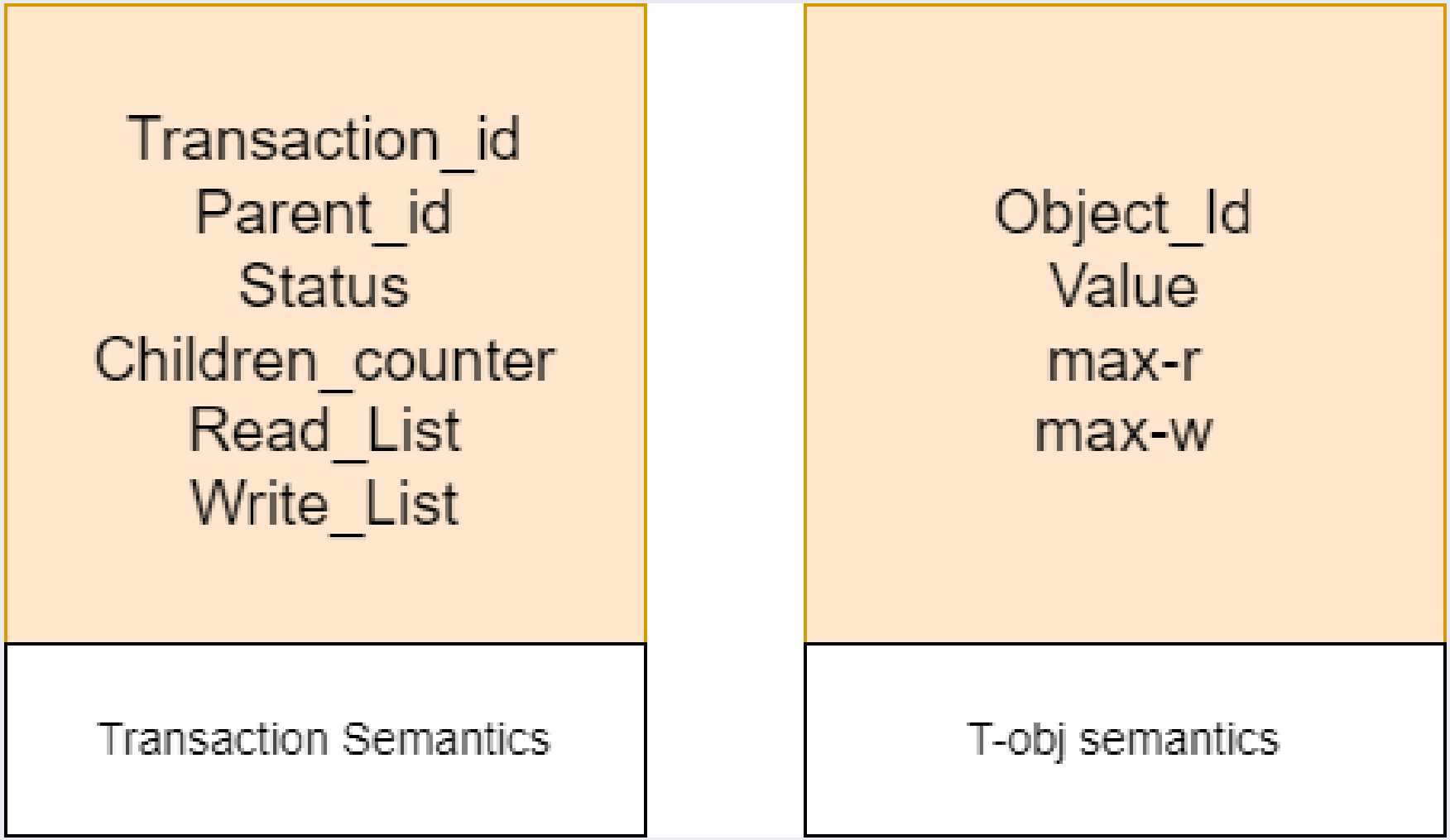


Figure: Transaction and T-obj Structures

Read Rule

- Each transaction(sub) will first search required data objects in it's write-list and read-list.

Read Rule

- Each transaction(sub) will first search required data objects in it's write-list and read-list.
- If not found, the transaction will traverse up and look for data items in its ancestors' buffer.

Read Rule

- Each transaction(sub) will first search required data objects in it's write-list and read-list.
- If not found, the transaction will traverse up and look for data items in its ancestors' buffer.
- After validation, the data items will be traversed down along the same path, filling the buffers of the transactions.

Read Rule Illustration

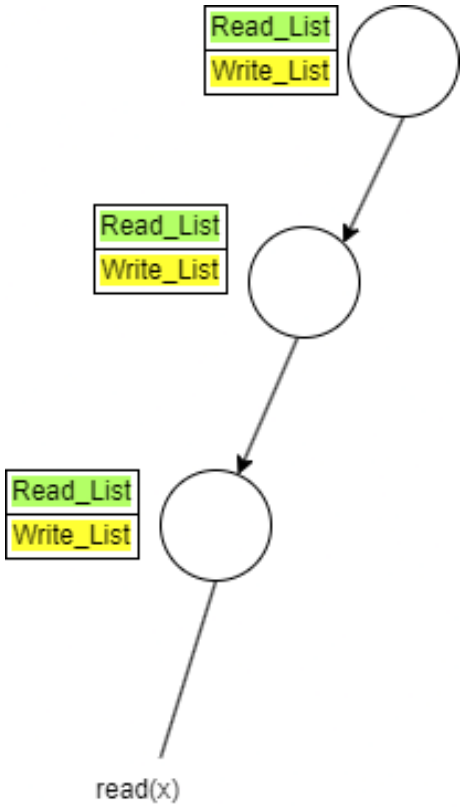


Figure: Working of Read Operation

Read Rule Illustration

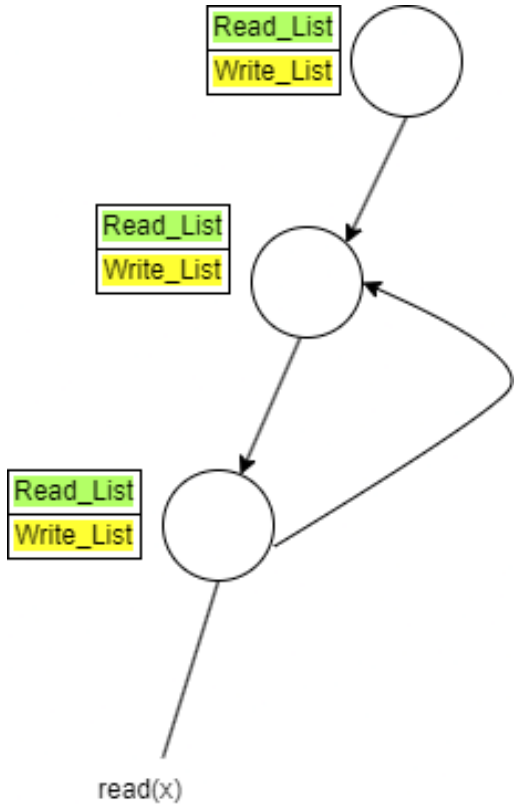


Figure: Working of Read Operation

Read Rule Illustration

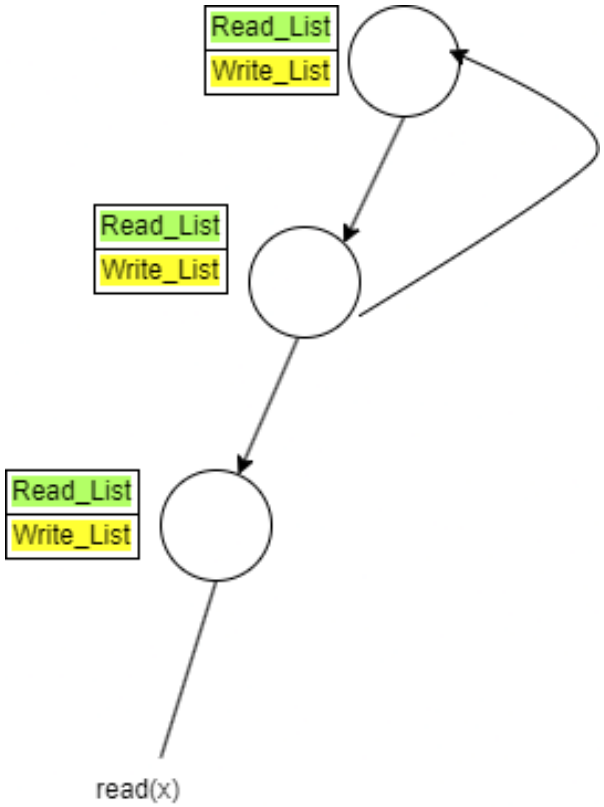


Figure: Working of Read Operation

Write Rule

- Deferred Write Approach.

Write Rule

- Deferred Write Approach.
- Each transaction will perform updates on its local buffer.

Write Rule

- Deferred Write Approach.
- Each transaction will perform updates on its local buffer.
- On successful completion, the updates made by the transaction will get reflected in its parent's buffer.

Write Rule Illustration

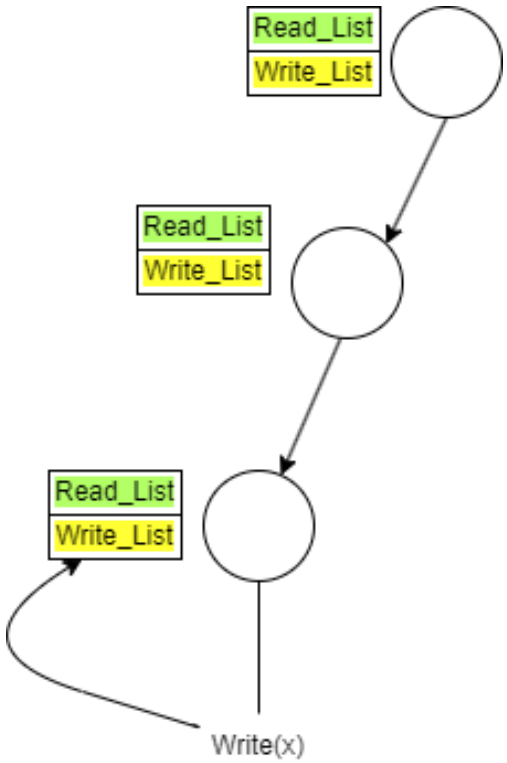


Figure: Working of Write Operation

TryC rule

- Operation p_i(x) is compared to *max-q(x)* for each conflicting q:
  - If ts(t_i) $<$ *max-q(x)* for some q then abort t_i.

TryC rule

- Operation p_i(x) is compared to *max-q(x)* for each conflicting q:
  - If ts(t_i) $<$ *max-q(x)* for some q then abort t_i.
  - If t_i is a parent transaction, then all its descendent will also be aborted.

TryC rule

- Operation p_i(x) is compared to *max-q(x)* for each conflicting q:
  - If ts(t_i) $<$ *max-q(x)* for some q then abort t_i.
  - If t_i is a parent transaction, then all its descendent will also be aborted.
  - Else schedule p_i(x) for execution and set *max-p(x)* to ts(t_i).

TryC Rule Illustration

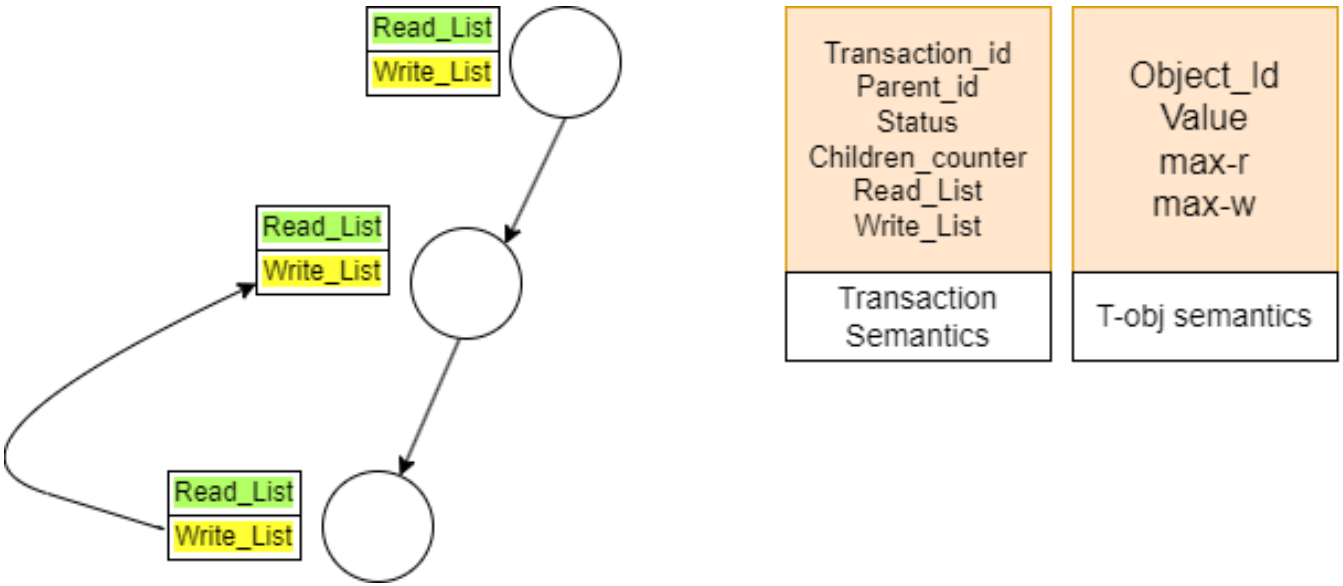


Figure: Working of TryC Operation

Correctness Guarantee

Theorem

The history generated by the NestedBTO algorithm satisfies ***Conflict-preserving Closed Nested Opacity (CP-CNO)****.*

This provides a formal, rigorous proof of correctness.

An Example

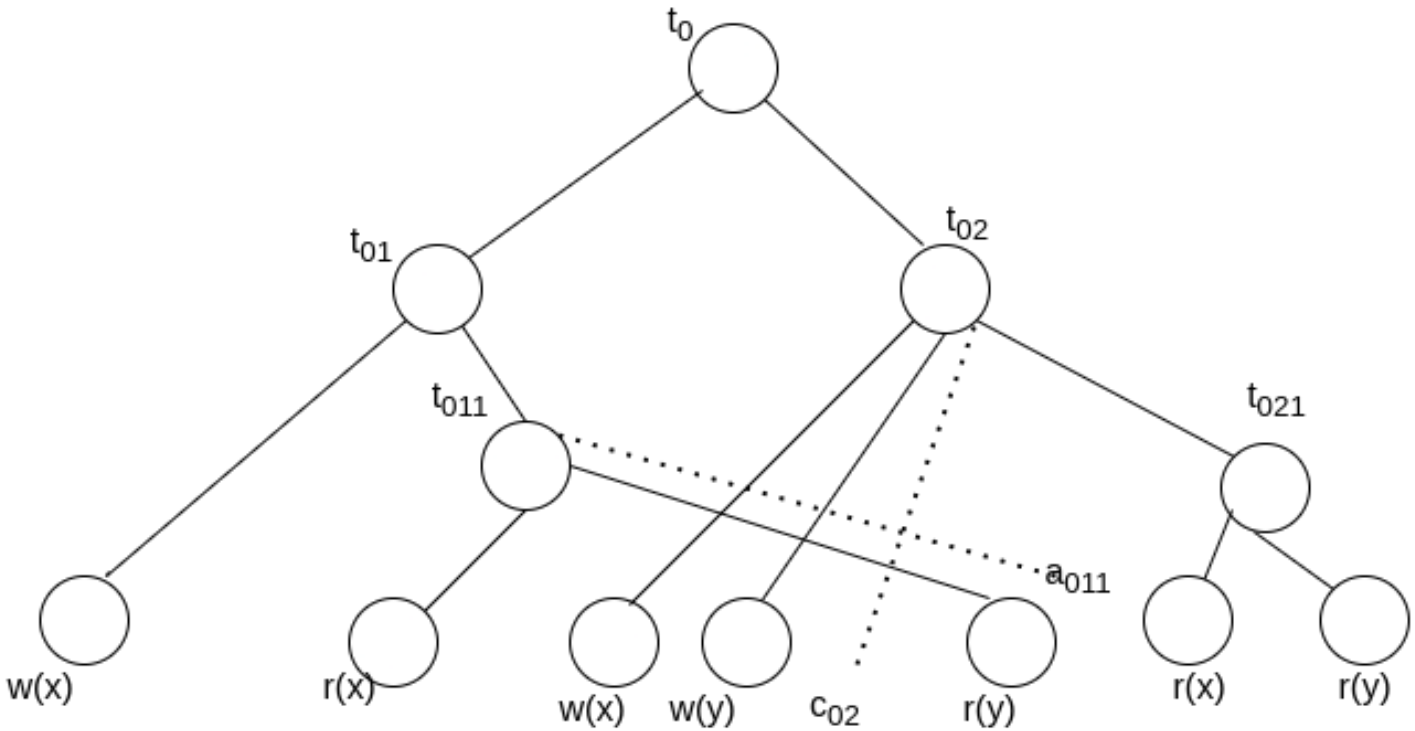


Figure: An Example

Experimental Evaluation

Evaluation setup

Platform

- AMD EPYC 7452 64 physical cores and 128 logical cores.
- Compiler - g++ 11.3.0 with POSIX MultiThreaded library support.

Goal

Compare **NestedBTO** against state-of-the-art **NesTM**[a].

[a] Baek, Woongki, et al. "Implementing and evaluating nested parallel transactions in software transactional memory." Proceedings of the twenty-second annual ACM symposium on Parallelism in algorithms and architectures.

Benchmarks

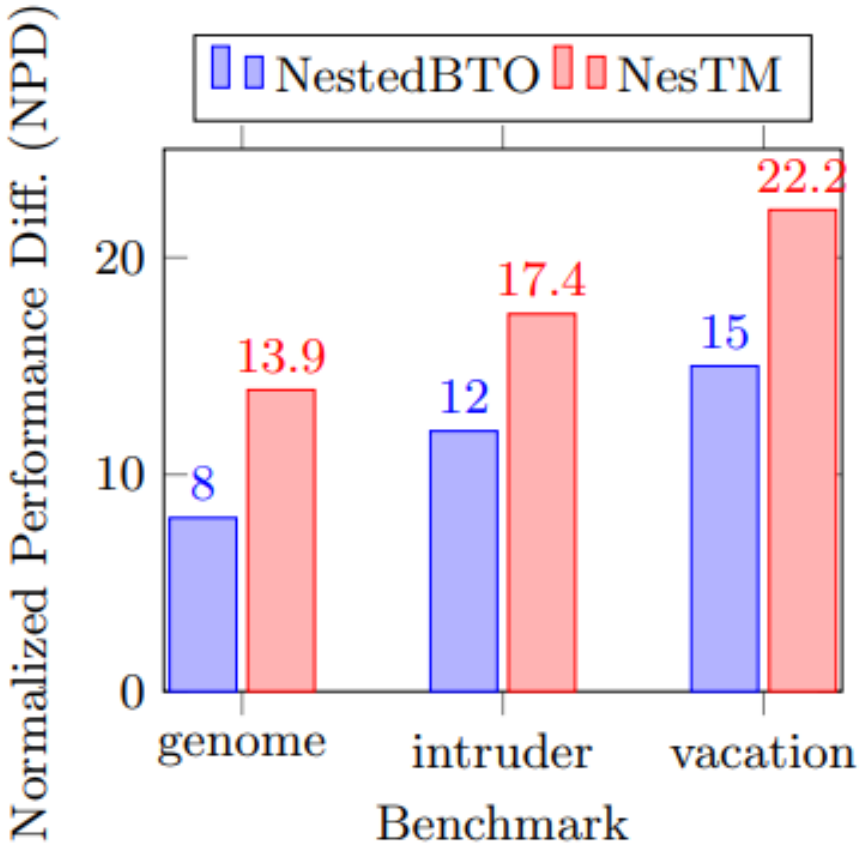
- **STAMP Suite:** Application-level workloads (Vacation, Intruder, etc.)
- **Microbenchmarks:** Concurrent Red-Black Tree, Hash Table.

Metrics

Throughput, Abort Rate.

Results: STAMP Benchmark

- **Throughput:** NestedBTO shows 5-10% better performance.
- **Scalability:** Superior at higher nesting levels.
- **Abort Rate:** Reduces aborts by 25-30% in high-contention (e.g., intruder).



STAMP Throughput Comparison

Reason: Timestamp-based conflict resolution avoids lock contention.

Results: Red-Black Tree Benchmark

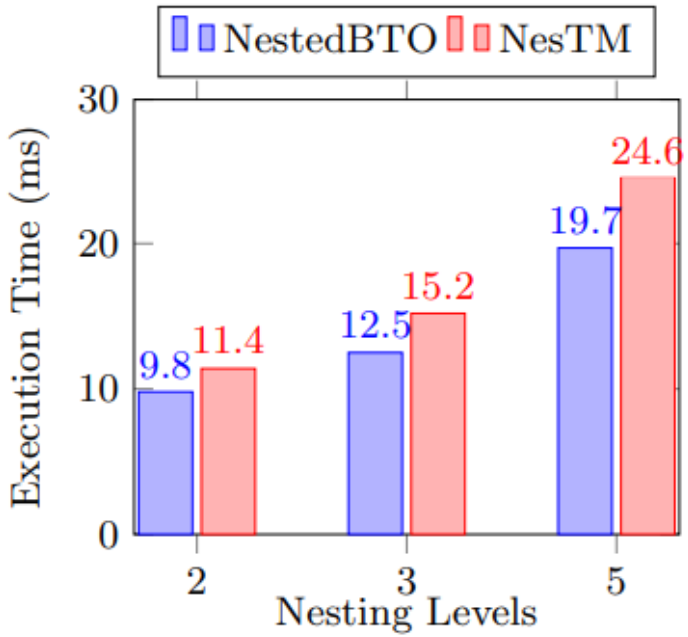


Figure: Throughput Comparison

Throughput: 16-25% faster as nesting depth increases.

NestedBTO's timestamp-based conflict detection minimizes synchronization

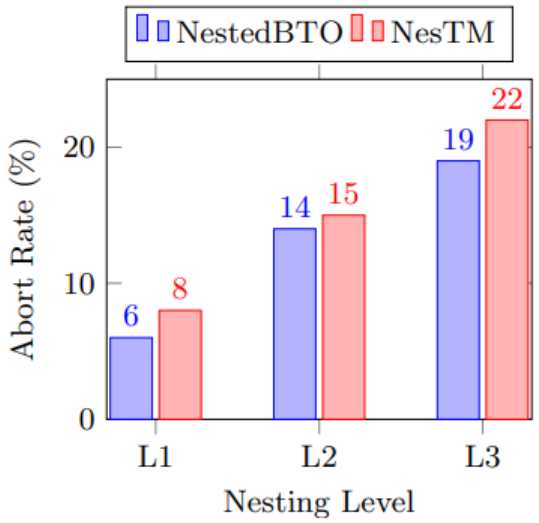


Figure: Abort Rate Comparison

Abort Rate: Reduces cascading aborts by 20-30% for heavily nested transactions.

Deterministic timestamp ordering reduces unnecessary aborts.

Results: Hash Table Benchmark

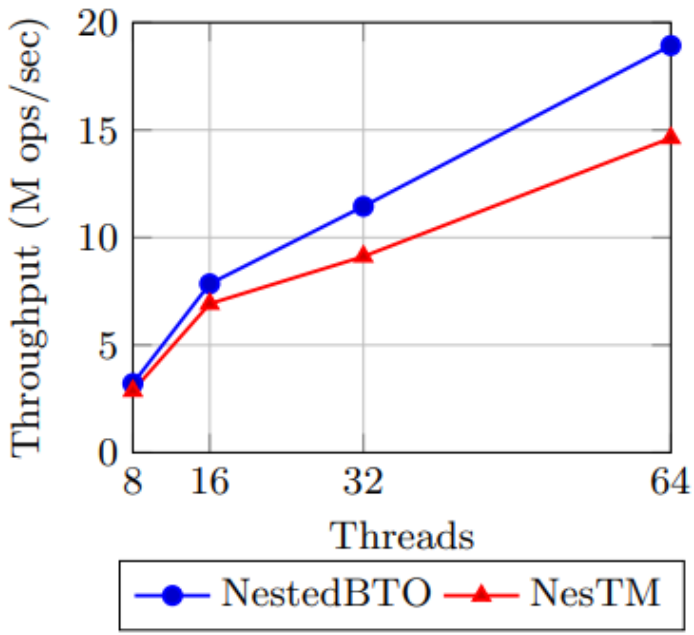


Figure: Throughput Comparison

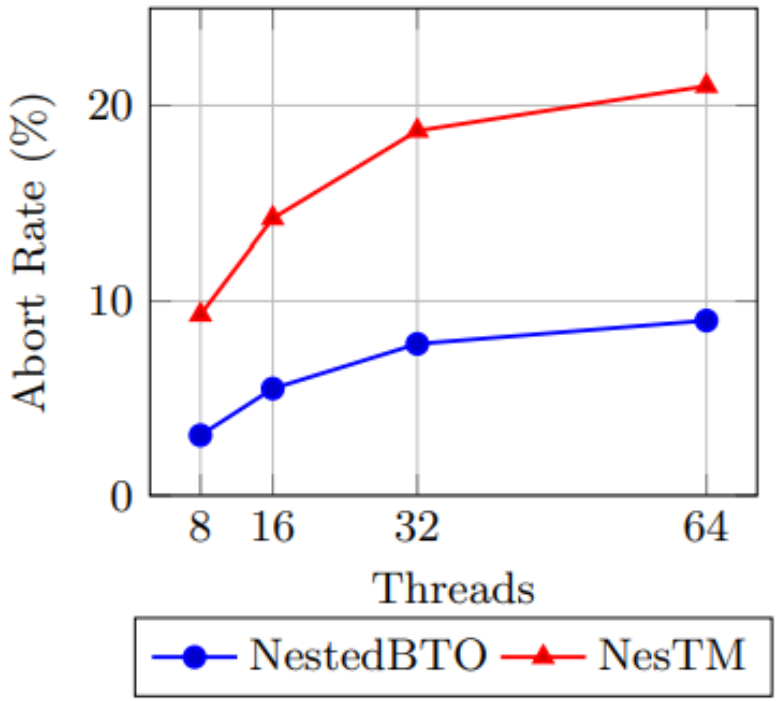


Figure: Abort Rate Comparison

Throughput: Up to 1.29× higher throughput at 64 threads.

NestedBTO's timestamp-based commit protocol outperforms NesTM's locking mechanism.

Abort Rate: 9% vs 21% at high contention.

Deterministic timestamp ordering reduces aborts compared to NesTM's lock-induced stalls.

Conclusion & Future Works

Conclusion

- **NestedBTO** is a high-performance STM protocol for closed nested transactions.
- Its core innovation is a timestamp-based protocol that **formally guarantees opacity**.
- It enables parallel execution of child transactions, improving performance and scalability.
- Extensive evaluation shows significant improvements over the state-of-the-art.

Future Work

- **Multi-versioning:** Allow multiple versions of objects to increase concurrency.

Future Work

- **Multi-versioning:** Allow multiple versions of objects to increase concurrency.
- **Distributed STM:** Explore extending the protocol to distributed systems.

Thank you!

Backup Slides

Opacity

- A History H is a opaque if there exists a legal sequential History S such that
  1. H is valid
  2. the operations of H and S are the same
  3. S respects the real time order of H

Conflict Notion

- Two data operations on the same data items are conflicting:
  - if one of them is a write operation
- Checking for Opacity is similar to checking for View Serializability in databases
  - can not be done efficiently
- Conflicts can be used to efficiently test Conflict Serializability, a subclass of View Serializability
  - All known database transaction schedulers (2PL, Optimistic execution etc) are a subclass of Conflict Serializability
- Similarly conflicts can be used to efficiently check subclass of Opacity

External Reads

- For a transaction t_x, we define
  - External-read: A read operation belonging to t_x or a descendent of t_x, whose lastWrite is external to t_x
- Extending this concept to simple - memory reads, a simple-memory read is an external read of itself

New Conflict Notion: OptConf

- OptConf
  - Is defined between commit-writes and external-reads of peer transactions
  - Similar to traditional conflicts, w - r, r - w and w –w OptConfs are defined
- Using OptConf, we define:
  - Subclass of conflict Preserving Closed Nested Opacity or CP-CNO
  - Membership can be verified in polynomial time

Correctness of Close Nested STMs

Issues with Nested Transactions

- In Nested Transactions
  - Aborted transaction can have committed sub-transactions
  - And committed transactions can have aborted sub-transactions

Correctness of Close Nested STMs

Issues with Nested Transactions

- In Nested Transactions
  - Aborted transaction can have committed sub-transactions
  - And committed transactions can have aborted sub-transactions
- To extend Opacity's definition to nested transactions,
  - Need some mechanism to treat aborted transaction
  - Need to verify membership in polynomial time

Correctness of Close Nested STMs

Issues with Nested Transactions

- In Nested Transactions
  - Aborted transaction can have committed sub-transactions
  - And committed transactions can have aborted sub-transactions
- To extend Opacity's definition to nested transactions,
  - Need some mechanism to treat aborted transaction
  - Need to verify membership in polynomial time
- **So far, there are no nested STM implementations that satisfy Opacity correctness**

Challenges in Nesting

- Providing Correct Synchronisation between threads

Challenges in Nesting

- Providing Correct Synchronisation between threads
- Dealing with Conflicts between transactions at different levels of the nesting hierarchy

Challenges in Nesting

- Providing Correct Synchronisation between threads
- Dealing with Conflicts between transactions at different levels of the nesting hierarchy
- Managing the state of the transactions within the nesting hierarchy

Challenges in Nesting

- Providing Correct Synchronisation between threads
- Dealing with Conflicts between transactions at different levels of the nesting hierarchy
- Managing the state of the transactions within the nesting hierarchy
- Maintaining consistency between Nested transactions is harder

Challenges in Nesting

- Providing Correct Synchronisation between threads
- Dealing with Conflicts between transactions at different levels of the nesting hierarchy
- Managing the state of the transactions within the nesting hierarchy
- Maintaining consistency between Nested transactions is harder
- Scalability Issue

Challenges in Nesting

- Providing Correct Synchronisation between threads
- Dealing with Conflicts between transactions at different levels of the nesting hierarchy
- Managing the state of the transactions within the nesting hierarchy
- Maintaining consistency between Nested transactions is harder
- Scalability Issue
- Rollback Issue

Related Work on Nested STM

- CWSTM, proposed by Agrawal et al., based on eager conflict detection and work-stealing approach, did not provide complete design implementation

Related Work on Nested STM

- CWSTM, proposed by Agrawal et al., based on eager conflict detection and work-stealing approach, did not provide complete design implementation
- PNSTM, proposed by Barreto et al., is motivated by CWSTM, but limits the maximum number of concurrent transaction

Related Work on Nested STM

- CWSTM, proposed by Agrawal et al., based on eager conflict detection and work-stealing approach, did not provide complete design implementation
- PNSTM, proposed by Barreto et al., is motivated by CWSTM, but limits the maximum number of concurrent transaction
- NeSTM, proposed by Baek et al., an extension of McRT-STM to provide nesting, defined its own locks, expected user to understand the structure of the lock

Related Work on Nested STM

- CWSTM, proposed by Agrawal et al., based on eager conflict detection and work-stealing approach, did not provide complete design implementation
- PNSTM, proposed by Barreto et al., is motivated by CWSTM, but limits the maximum number of concurrent transaction
- NeSTM, proposed by Baek et al., an extension of McRT-STM to provide nesting, defined its own locks, expected user to understand the structure of the lock
- JVSTM, proposed by dieges et al., an extension of the original non-nested JVSTM, used Vboxes

Related Work on Nested STM

- CWSTM, proposed by Agrawal et al., based on eager conflict detection and work-stealing approach, did not provide complete design implementation
- PNSTM, proposed by Barreto et al., is motivated by CWSTM, but limits the maximum number of concurrent transaction
- NeSTM, proposed by Baek et al., an extension of McRT-STM to provide nesting, defined its own locks, expected user to understand the structure of the lock
- JVSTM, proposed by dieges et al., an extension of the original non-nested JVSTM, used Vboxes
- CP-CNO, A correctness criteria for closed nested transaction system