A concurrent object is a data object shared by multiple threads of control within a concurrent system.

DEFINITION

BACKGROUND INFO.

- Lack of fault tolerance
- Concurrency
- Priority inversion
- Deadlocks

Important drawbacks:
- Concurrent objects suffer from several concurrent object-based implementations of classic lock-based implementations of

BACKGROUND INFO. CONT'D.

from making forward progress
- Failure of a thread can never prevent the system

synchronization algorithms
- Increased interest in nonblocking

SYSTEMS
SOFTWARE TRANSACTIONAL MEMORY
DESIGN TRADEOFFS IN MODERN
Introduction

Section 1:

DEFINITION

- Transaction

- A finite sequence of instructions (satisfying the
  linearity and atomicity properties) that is
  used to access and modify concurrent objects

OUTLINE

Section 1 – Introduction

Section 2 – DSTM

Section 3 – FSTM

Section 4 – Comparative Evaluation

Section 5 – Related Work

Section 6 – Conclusions

INTRO: SOFTWARE TRANSACTIONAL MEMORY

- STM – an approach to construct nonblocking
  objects

- Simplifies task of implementing concurrent
  objects

- Software Transactional Memory (STM) is a
  generic non-blocking synchronization
  construct that enables automatic conversion
  of correct sequential objects into correct
  concurrent objects.
Opening a TMO before (in write mode) recently modified by a committed transaction

Dynamic Software Transactional Memory

Section 2:

Purpose of Paper

This paper focuses on discussing two approaches to STM. It compares and evaluates strengths and weaknesses of the approaches.
Opening a TMXObject in write mode; currently modified by a committed transaction.

Opening a TMXObject in write mode; currently modified by a committed transaction.

Opening a TMXObject in write mode; currently modified by a committed transaction.

Opening a TMXObject in write mode; currently modified by a committed transaction.
FSWM (Fraser's Software Transactional Memory)

Section 3: Must request validity before committing
- Opened in read-only mode
- Each transaction maintains private (read-list) of objects
- How do we avoid this contention?
- Unnecessary contention between transactions
- Full acquire operation

DSFM - WHAT ABOUT READ-ONLY?

DSFM READ-ONLY CONT'D

Opening
Current version does this automatically when
read-only objects
Transaction would have to revalidate all open
Initiate loops, division by zero, etc.
May lead to unwanted behavior (addressing errors,
transaction?)
What about stale data (from a not yet committed
most recent previous transaction may still be
ACQUIRE
Example follows:
aborts/keeps its.
transaction to make updates. Then
another, it uses the conflicting transaction's
When transaction detects conflict with


Recursive Helpling

How?

(complete a transaction)
(bounded number of steps, some thread is guaranteed to
Guarantees forward progress for system (within a
Unlike DSTM, FSTM is lock-free
Named after Kier Fraser (Cambridge)
Basic Transactional Memory Structure
BASIC TRANSACTIONAL MEMORY STRUCTURE

[Diagram of basic transactional memory structure]
Unlike DSTM, multiple transactions can open the same object in write mode because each has its own shadow copy.
CAS Conflict Discovery

CAS Conflict Discovery

Example follows:
- Description: conflict is detected.
- If pointer already points to another transaction's
  transaction descriptor:
  - Each CAS wins object header's pointer to
    use atomic compare and swap (virtual)

Transaction acquires all objects it opened in

COMMIT PHASE

Helping
- If conflict is detected, the transaction uses recursive
  recursive
- OPEN by another transaction, will be detected

COMMITTED
- OPEN is not visible to other transactions
- Transaction opens object headers while in this

UNDECIDED
Comparative Evaluation

This section does the following:
- Highlights the impact on performance of various concurrent data structures
- Highlights design tradeoffs between the two approaches
- Simpler than previous approaches
- Both DSM and FSTM have significant overhead

READ-CHECKING STATE

Recursive HELPING (for validdating)
- Also, commit will be in READ-
- If not, commit will be aborted
global total order (thread/transaction id)
Only proceeds if commitment proceeds in in

COMMITTED
After successful validation, switch to
- If pointing to another transaction, recursively help
- If not, abort
- Object as when handle was created
- Very object header still points to version of
- Validates objects in read-only list

Section 4:
Object Acquire Semantics

FSTM

Transaction acquires exclusive access only in

commit phase

Eager acquire

Transaction acquires visible to potential competitors

lazy acquire

Later in its lifetime (lazy acquire)

Red-black tree

More Complex Benchmark

3 variants of a list-based set

A stack

Simple Benchmarks

TESTING BENCHMARKS

Six test runs

Vary number of worker threads between 1-48

Measured total throughput over 10 seconds

Condition

Keeping this range small increases probability of

255

Repeatedly/Randomly insert/delete Integers 0-

In list and Red-black tree benchmarks

TESTING BENCHMARKS CONT...
FS TM outperforms due to lazy acquire

Eager acquire

Embraces earlier detection & resolution of conflicts

Enables conflict

May cause transactions to waste computational resources on doomed transactions before detecting a conflict

Tends to reduce amount of transactions identified as conflict

Contracts

Avoids application semantics allowing both transactions to commit, lazy acquire may result in higher concurrency

Object Acquire Semantics - Performance Results

Object Acquire Semantics - Performance Results

Read & Write Performance Results

Object Acquire Semantics - Performance Results

Object Acquire Semantics - Performance Results

Number of Contention Instances

# of Threads

Contention Instances, sec

Contention Instances, sec

0

50+*

50+*

0

4

8

12

16

20
- Insert/Release
- Insert
- Stack

Testing Benchmarks

Performance

- Bookkeeping and Indirection
- Lazy acquire requires extra bookkeeping
- Faster in DSTM
- Transactions with large number of writes may be
- Object handles into descriptor chains
- Indirection eliminates overhead of inserting

- Bookkeeping and Indirection
- Slower transactions (particularly if most are read-only)
- Reads/Writes
- Extra indirection may result in slower
- FSITM object header points directly to object data
- FSITM points to locator which points to
- DSTM has extra level of indirection (Fig. 1.8.2)
- DSTM outperformed again
- Difference remained more or less constant
- Object acquire semantics - Performance results
Initial Performance Results

- Due to FSTM extra bookkeeping in write mode
- DSTM outperforms stack pointer
- Stack illustrates very high contention (lock)

Successful transactions are serialized
- Objects from beginning of list
- Every insertion/deletion opens (write mode) all
  Initial mainlist sorted list

Initial Performance Results

Stack Performance Results

- FSTM - FSTM.NoRef
- FSTM - FSTM.NoRef2
- FSTM - FSTM.NoRef3
- DSTM's indirection is to blame.
- FSM more than a factor of 2 better.

**Bookkeeping and Indirection**

**Performance Results**

- Or upgrade to white mode.
- Then either released (moving on to next node in list).
- All others opened temporarily in read mode.
- Only if object to be modified is opened in white.

**InferRedeal - variant of Inset**

**Bookkeeping and Indirection**

**Performance Results**

- Sorting overhead, and extra cases.
- FSM suffers extra bookkeeping overhead.
- Higher throughput for DSTM.
Inconsistentues is left to the programmer.

The exception handler validates the transaction that caused the exception.

On a memory access violation, exception

transaction validation - FSTM

Invisible reads & lazy acquire (FSTM) may allow transaction to enter inconsistent state during execution.

This may lead to memory access violations.

Initialize loops & arithmetic faults, etc.

DSTM - Performs incremental validation at open. Time

FSTM - Proposes mechanism based on exception

Handing (catch) problems when they arise rather than prevent them.

Some concurrency issues. The less that most objects are read-only, introduces changed objects in write mode only if object needs to be similar to Insettleupgradefree.

Transaction validation

Transaction validation
Conclusions

Related Work

Conclusions

Both systems incur significant overhead for read-
write mode. DSM tends to do better for transactions in
practical, object-based STM systems.

Paper evaluated tradeoffs in design of

Related Work

Bookkeeping overhead in read mode

Concurrency — optimization to DSM to reduce

Recursive help

Avoids cache invalidation that might result from

Hashes & Fraser — novel stealing mechanism

Transactions acquire records before updating

Records

Hashes shared memory words into ownership

Harris & Fraser — word-based STM

Transaction Validation Performance Results

(At least frequency of incremental validation

Application-specific reasoning to eliminate need

Results suggest potential benefits from using

Incremental validation introduces dramatic

Overhead in both systems