Object Based
Transactional Memory

Ali Saoud
Introduction

- Recent trends go towards object based SMT because it’s dynamic.

- Word-based STM systems are more suitable for data structures that may require concurrent access at a high level of granularity (e.g. multi-dimensional arrays).
Dynamic Software Transactional Memory (DSTM)

The locator is the main key to the design of the DSTM. Every pointer goes through a level of indirection.
Dynamic Software Transactional Memory (DSTM)

If the transaction is ACTIVE or ABORTED, the most recent valid version of the data object is the old version referenced by the locator.

If the transaction is COMMITTED, the most recent valid version of the data object is the new version referenced by the locator.
Dynamic Software Transactional Memory (DSTM)
Open For Writing

Handler

T1(open)
Open For Writing

Handler

T1(open) → Transaction

New object → Object Data

Old object → Object Data

T0: committed
A successful CAS guarantees that the current transaction is visible to the entire concurrent system. A failure in CAS implies that some other transaction has opened (acquired) the TM Object in between.
Open For Writing

Transaction
- New object
- Old object

T0: Aborted
- Object Data
- Object Data

Handler

T1(open)
Open For Writing

Handler

Transaction
  New object
  Old object

T0: aborted
  Object Data

Transaction
  New object
  Old object

T1 active
  Object Data

Copy
A transaction goes through a novel contention management protocol to decide whether to abort itself or the TM Object's current ACTIVE owner transaction.

- aggressive – always/immediately aborts conflicting transaction
- polite – adaptive back-off

contention reduced by “early release”: reference to object dropped before transaction commits and subsequent changes to the released object does not jeopardize consistency
Open For Reading

Handler

Transaction
- New object
- Old object

T0: Committed
- Object Data
- Object Data

T1 Active
- Read only list

Object
- Value
- Next
Open For Reading

Handler

Transaction
  New object
  Old object

T1 Active
  Read only
  list

Object
  Value
  Next

T0: Committed
  Object Data
  Object Data
Conflicts among transactions are detected and resolved at commit-time

Figure 5: The basic Transactional Memory Structure in FSTM
**Figure 5:** The basic Transactional Memory Structure in FSTM
Commit

• **Acquire**

  • Acquire each object in the read-write list in global total order using atomic CAS for each object
    • Abort if conflict with committed transaction detected
    • Help if conflict with uncommitted transaction detected
Commit 2

- **Read-checking**
  - Verify consistency of each object in the read-only list
    - Abort if change is detected in object held by Undecided transaction
    - If conflict detected with Read-checking transaction:
      - Help if other transaction precedes current transaction
      - Abort if current transaction precedes other transaction
  - Release acquired transactions
<table>
<thead>
<tr>
<th>Characteristic</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>STM-1</td>
</tr>
<tr>
<td>Strong/Weak Isolation</td>
<td>N/A</td>
</tr>
<tr>
<td>Granularity</td>
<td>Word</td>
</tr>
<tr>
<td>Direct/Deferred Update</td>
<td>Direct</td>
</tr>
<tr>
<td>Concurrency Control</td>
<td>Pessimistic</td>
</tr>
<tr>
<td>Synchronization</td>
<td>Lock-free</td>
</tr>
<tr>
<td>Conflict Detection</td>
<td>Early</td>
</tr>
<tr>
<td>Inconsistent Reads</td>
<td>None</td>
</tr>
<tr>
<td>Conflict Resolution</td>
<td>Helping</td>
</tr>
<tr>
<td>Nested Transactions</td>
<td>Flattened</td>
</tr>
<tr>
<td>Exceptions</td>
<td>Terminate</td>
</tr>
</tbody>
</table>