On Transactional Memory Concurrency Control in Distributed Real-Time Programs

Sachin Hirve, Aaron Lindsay, Roberto Palmieri, Binoy Ravindran
Department of Electrical and Computer Engineering, Virginia Tech, Virginia, USA
{hsachin, robertop, binoy}@vt.edu, aaron@aclindsay.com

http://ssrg.ece.vt.edu

Systems Software Research Group

VirginiaTech Invent the Future®

Applying Concurrency Control in Distributed Real-time Programs

Solution to commit transactions as per task priorities

Distributed embedded software is inherently concurrent, as they monitor and control concurrent physical processes. Often, their computations need to concurrently access (i.e., read/write) shared data objects, which must be properly coordinated so that consistency properties (e.g., linearizability, serializability) can be ensured. Furthermore, they must satisfy application time constraints. The usual way for managing concurrency of different processes in a system is using locks, which inherently suffers from programmability, scalability, and composability challenges.

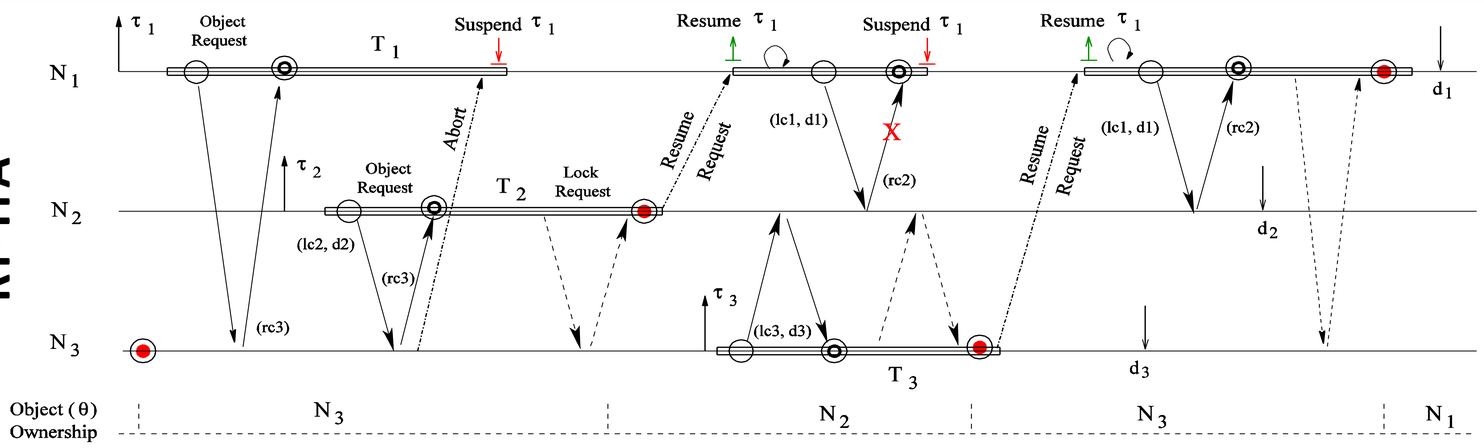
MAIN CHALLENGE

- **Honor task deadlines for transactions.**
- **Bound distributed transactional retries.**

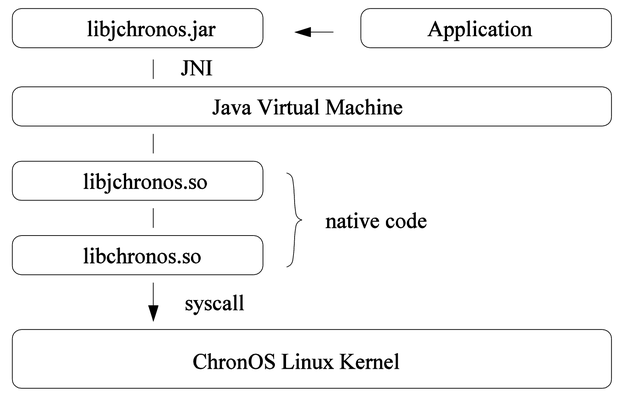
- ❖ We present RT-TFA (Real-Time Transaction Forwarding Algorithm), a real-time distributed transactional memory.
- ❖ RT-TFA transparently handles object relocation and versioning using an asynchronous clock-based validation technique.
- ❖ RT-TFA supports data-flow model i.e. transaction are fixed on invoking node and objects migrate to nodes.
- ❖ Transactions carry deadline of subsuming tasks.
- ❖ RT-TFA resolves transactional contention using task time constraints.
- ❖ We assume a bounded clock drift using clock synchronization.

Main components of the proposed solution

Transaction execution under RT-TFA



System Architecture



Concurrency Control

- ❖ RT-TFA extends TFA (Transaction Forwarding Algorithm) to support transactions that execute under time constraints.
- ❖ Transactions inherit deadlines of their parent tasks.
- ❖ Objects are acquired at encounter time and object request carry deadline to remote node.
- ❖ Transactions are early-aborted if conflicts are detected at object access time.
- ❖ Locks are acquired at commit time and transactions resolve conflicts using the deadlines of subsuming tasks before getting locks over objects.

System Architecture

- ❖ The implementation consists of a stack of ChronOS Real-Time Linux kernel, JChronOS (a Java interface library), JVM, RT-TFA and application.
- ❖ ChronOS supports various scheduling algorithms (EDF, RMA, GEDF, DASA etc.).
- ❖ Time constraints are expressed using scheduling segments in a thread.
- ❖ Scheduling segments occur at regular intervals and have deadlines.
- ❖ JChronOS library extends scheduling interface of ChronOS for Java programs.

Experimental Evaluation

Configuration

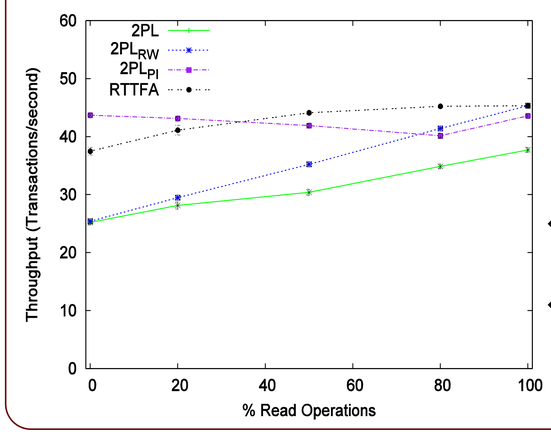
- ❖ Private cluster of 14-nodes (AMD Opteron processor, 1.9GHz).
- ❖ Each node runs a set of periodic tasks constituting 70 distributed tasks.
- ❖ Implementation of concurrency control in Hyflow Java DTM framework.
- ❖ Benchmark: Bank
- ❖ Parameters tested: effects of variation in % of read transactions on throughput and deadline satisfaction ratio (DSR) of distributed tasks.

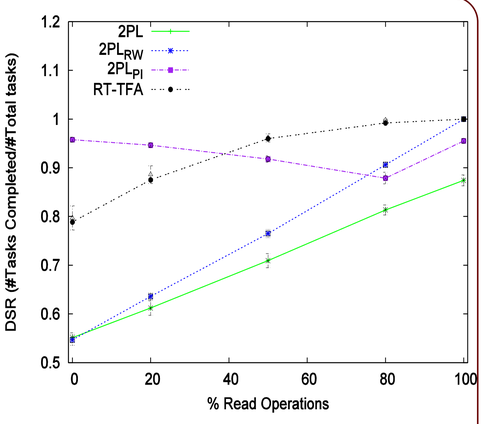
Results and Discussion

Bank Benchmark:

- ❖ A monetary application, which maintains a set of accounts distributed over bank branches and contains two transactions (*transfer* and *total balance*).
- ❖ Compared with three variations of classic two-phase-locking protocol.

- ❖ RT-TFA underachieves compared to 2PL-PI for write-intensive loads due to higher abort rate and retries.
- ❖ RT-TFA yields comparable DSR to 2PL-PI based locking protocol and out perform others by as much as 43%.





- ❖ RT-TFA does lazy locking and acquires locks at the commit time.
- ❖ 2PL serializes objects by acquiring locks before entering critical section, therefore suffers higher priority inversion costs.

Finally…

Our results revealed that RT-TFA yields comparable or better deadline satisfaction ratios to 2PL-based locking protocols. Additionally it allows programmers to reap benefits of DTM's programmability and composability properties.

www.ssrg.ece.vt.edu

ACKNOWLEDGMENTS

This work is supported in part by US National Science Foundation under grants CNS 0915895, CNS 1116190, CNS 1130180, and CNS 1217385.