HyFlow: A High Performance Distributed Software Transactional Memory Framework

Mohamed M. Saad
ECE Dept., Virginia Tech
Blacksburg, VA 24061, USA
msaad@vt.edu

Binoy Ravindran
ECE Dept., Virginia Tech
Blacksburg, VA 24061, USA
binoy@vt.edu

ABSTRACT

We present *HyFlow* — a distributed software transactional memory (D-STM) framework for distributed concurrency control. HyFlow is a Java framework for D-STM, with pluggable support for directory lookup protocols, transactional synchronization and recovery mechanisms, contention management policies, cache coherence protocols, and network communication protocols. HyFlow exports a simple distributed programming model that excludes locks: using (Java 5) annotations, atomic sections are defined as transactions, in which reads and writes to shared, local and remote objects appear to take effect instantaneously. No changes are needed to the underlying virtual machine or compiler. We describe HyFlow's architecture and implementation, and report on experimental studies comparing HyFlow against competing models including Java remote method invocation (RMI) with mutual exclusion and read/write locks, distributed shared memory (DSM), and directory-based D-STM. Our studies show that HyFlow outperforms competitors by as much as 40-190% on a broad range of transactional workloads on a 72-node system, with more than 500 concurrent transactions.

Categories and Subject Descriptors

D.1.3 [**Software**]: Concurrent Programming; H.2.4 [**Systems**]: Transaction processing

General Terms

Design, Languages, Measurement, Performance

Keywords

Software Transactional Memory, Distributed systems, Dataflow, Control Flow, Framework, Java

1. INTRODUCTION

Lock-based synchronization suffers from programmability, scalability, and composability challenges [4]. Transactional memory (TM) [4] is a promising alternative to lock-based concurrency control. In addition to providing a simple programming model, TM provides performance comparable to highly concurrent, fine-grained locking, and is composable. Similar to multiprocessor TM, distributed STM (or D-STM) is an alternative to lock-based distributed concurrency control. D-STM can be supported in any of the classical distributed execution models, including a) control flow [1], where objects are immobile and transactions invoke object operations through RPCs; b) dataflow [11], where transactions are immobile, and objects are migrated to invoking transactions; and c) a hybrid model (e.g., [2]), where transactions or objects are migrated, heuristically, based on properties such as access profiles, object size, or locality. The different models have their concomitant tradeoffs.

We present HyFlow — the first ever D-STM framework implementation. HyFlow supports both dataflow, control flow, and hybrid execution models, and ensures distributed transactional properties including atomicity, consistency, and isolation. HyFlow's architecture is module-based, with well-defined APIs for further plugins. Default implementations exist for all needed modules. The framework currently includes three algorithms to support distributed memory transactions: the distributed transactional locking II (D-TL2) algorithm [8], the distributed RMI-DSTM algorithm [9], and a distributed variant of the UndoLog algorithm. A wide range of contention management policies (e.g., Karma, Aggressive, Polite, Kindergarten, Eruption) [10] are included in HyFlow. Four directory protocols [3,5,12] are implemented in HyFlow to track objects distributed over the network. HyFlow uses a voting algorithm, the dynamic two phase commitment protocol (D2PC) [6], to support control flow transactional executions. Network communication is supported using protocols including TCP, UDP, and SCTP. We also implement a suite of distributed benchmark applications in HyFlow, to evaluate D-STM algorithms and protocols.

HPDC'11, June 8–11, 2011, San Jose, California, USA.
ACM 978-1-4503-0552-5/11/06.

2. HYFLOW ARCHITECTURE

Figure 1 shows the nodal architecture of HyFlow. Five modules and a *runtime handler* form the basis of the architecture. The modules include the *Transaction Manager*, *Instrumentation Engine*, *Object Access Module*, *Transaction Validation Module*, and *Communication Manager*.

The HyFlow runtime handler represents a standalone entity that delegates application-level requests to the framework. HyFlow uses run-time instrumentation to generate transactional code, like other (multiprocessor) STM such as Deuce, yielding almost two orders of magnitude superior performance than reflection-based STM (e.g. DSTM2).

The Transaction Manager contains mechanisms for ensuring a consistent view of memory for transactions, validating memory locations, and retrying transactional code when

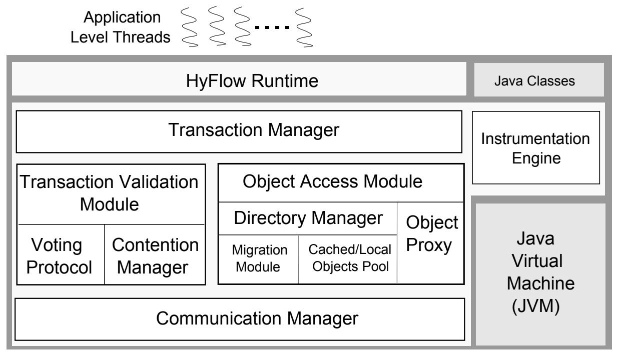

Figure 1: HyFlow Node Architecture

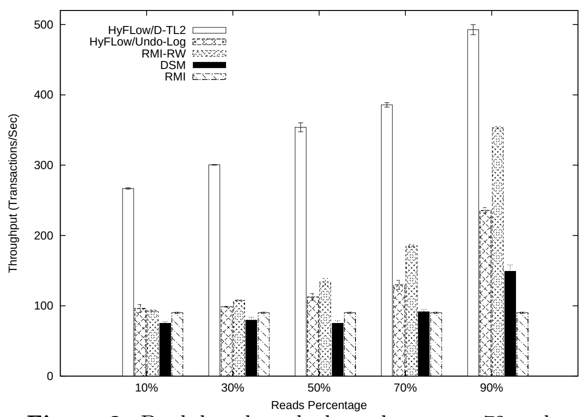

Figure 2: Bank benchmark throughput over 72 nodes.

needed. Based on the object access profile and object sizes, objects are migrated.

The Instrumentation Engine modifies class code at runtime, adds new fields, and modifies annotated methods to support transactional behavior. Further, it generates callback functions that work as “hooks” for Transaction Manager events such as onWrite, beforeWrite, beforeRead, etc.

Every node employs a Transaction Manager, which runs locally and handles local transactional code. The Transaction Manager treats remote transactions and local transactions uniformly. Thus, the distributed nature of the system is seamless at the level of transaction management.

The Object Access Module has three main tasks: 1) providing access to the object owned by the current node, 2) locating and sending access requests to remote objects, and 3) retrieving any required object meta-data (e.g., latest version number). Objects are located with their IDs using the Directory Manager, which encapsulates a directory lookup protocol (e.g. [5]). Upon object creation, the Directory Manager is notified and publishes the object to other nodes. The Migration Module decides when to move an object to another owner, or keep it locally. The purpose of doing so is to exploit object locality and reduce the overall communication traffic between nodes.

The Transaction Validation Module ensures data consistency by validating transactions upon their completion. It uses two sub-modules: 1) *Contention Manager.* This sub-module is consulted when transactional conflicts occur, toward aborting or postponing one of the conflicting transactions. However, when one of the conflicting transactions is remote, the contention policy decision is made globally based on heuristics; and 2) *Global Voting handler.* Validating control flow transactions requires a global decision across all participating nodes, such as by a voting protocol (e.g., D2PC [6]). This sub-module is responsible for collecting votes from other nodes and make a global commit decision.

Figure 2 shows the throughput of the different schemes at 10-90% read-only transactions, under increasing number of nodes, which increases contention (with all else being equal). We observe that HyFlow/D-TL2 outperforms all other distributed concurrency control models by 40-190%. Complete details of HyFlow is available in [7].

3. CONCLUSIONS

We presented HyFlow, a high performance pluggable, distributed STM that supports both dataflow and control flow distributed transactional execution. Our experiments show that the dataflow model scales well, as it permits remote objects to move toward geographically-close nodes that access them frequently, reducing communication costs. Control flow is beneficial under non-frequent object calls or calls to objects with large sizes. Our implementation shows that D-STM, in general, provides comparable performance to classical distributed concurrency control models, and exports a simpler programming interface, while avoiding dataraces, deadlocks, and livelocks. HyFlow provides a testbed for designing, implementing, and evaluating algorithms for D-STM. HyFlow is publicly available at hyflow.org.

4. REFERENCES

[1] K. Arnold, R. Scheifler, J. Waldo, B. O’Sullivan, and A. Wollrath. *Jini Specification.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[2] R. L. Bocchino, V. S. Adve, and B. L. Chamberlain. Software transactional memory for large scale clusters. In *PPoPP ’08.*

[3] M. J. Demmer and M. Herlihy. The Arrow distributed directory protocol. In *DISC ’98.*

[4] T. Harris, J. Larus, and R. Rajwar. Transactional Memory, 2nd edition. *Synthesis Lectures on Computer Architecture*, 5(1):1–263, 2010.

[5] M. Herlihy and M. P. Warres. A tale of two directories: implementing distributed shared objects in Java. In *JAVA ’99.*

[6] Y. Raz. The Dynamic Two Phase Commitment (D2PC) Protocol. In *ICDT’95.*

[7] M. M. Saad and B. Ravindran. Distributed Hybrid-Flow STM: Technical Report. Dec.’10.

[8] M. M. Saad and B. Ravindran. Distributed Transactional Locking II: Technical Report. Jan.’11.

[9] M. M. Saad and B. Ravindran. RMI-DSTM: Control Flow Distributed Software Transactional Memory: Technical Report. Feb.’11.

[10] W. N. Scherer III and M. L. Scott. Contention management in dynamic software transactional memory. In *PODC’04.*

[11] E. Tilevich and Y. Smaragdakis. J-Orchestra: Automatic Java application partitioning. In *ECOOP’02.*

[12] B. Zhang and B. Ravindran. Brief announcement: Relay: A cache-coherence protocol for distributed transactional memory. In *OPODIS’09.*