Hyflow2: A High-Performance Distributed Transactional Memory Framework in Scala

Alexandru Turcu, Binoy Ravindran, Roberto Palmieri*
talex@vt.edu, binoy@vt.edu, robertop@vt.edu

Electrical and Computer Engineering Department
Virginia Tech (USA)

PPPJ 2013

September 11, 2013

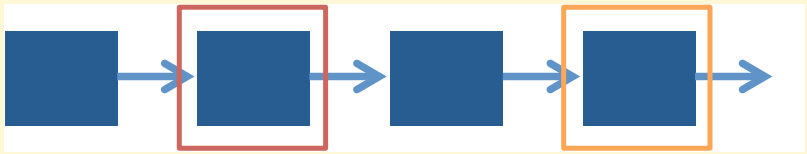
Overview

- Introduction
- Hyflow2 API
- Nested Transactions
- Java Compatibility
- Implementation
- Conclusions

Introduction

Lock-based concurrency control has serious drawbacks

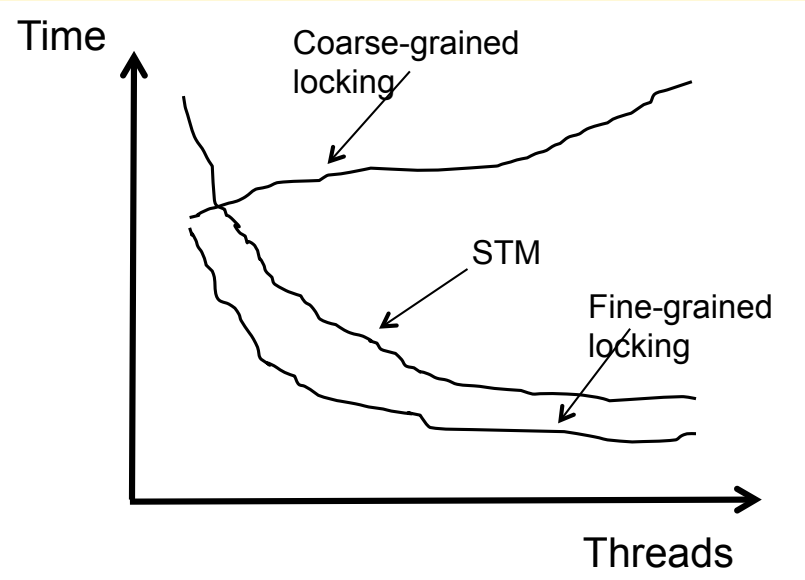
- Coarse-grained locking
  - Simple.
  - But no concurrency.



- Fine-grained locking
  - Excellent performance.
  - Poor programmability.
  - No composition.
  - Lock problems dont go away! Deadlocks, livelocks, lock-convoying, priority inversion, ...

Software Transactional Memory

- Like database transactions
- ACI properties (no D)
- Easier to program
- Composable
- First HTM, then STM, later HyTM



Distributed Concurrency

- Shared memory
  - Ensure safety using distributed locks.
  - Locks have problems: deadlocks, race conditions.
  - Difficult to debug (especially distributed).
- Traditional database transactions
  - Heavy-weight, slow, complex to set up.
  - Cumbersome to program (SQL).
- Actor model (message passing)
  - Promising, but not a silver bullet.
- Novel approaches:
  - *NewSQL*, main-memory DB, txn storage.
  - Distributed Transactional Memory (DTM)

Distributed Transactional Memory

- Promising new model for programming distributed concurrency
  - Abstracts away lock usage.
  - Programming concurrency using a TM library is easy: atomic blocks.

```
atomic {
    if (acc1.balance < amount) {
        abort()
    } else {
        acc1.balance -= amount
        acc2.balance += amount
    }
}
```

Introducing DTM Transactions

- Successful abstraction originating in the database community.
- Provides failure atomicity, consistency, isolation (possibly even durability).
- Code generally not sand-boxed: TM opacity vs. DB serializability.
- Two main approaches: redo-log and undo-log.

Transactional Forwarding Algorithm

- TFA is an existing protocol for distributed STM:
  - Transactional Locking II
  - Lamport clocks
- Early Validation
  - Validate read-set upon remote object access.
  - Txns are not allowed to become invalid.
  - Provides opacity.
- Objects migrate on commit (data-flow model)

Hyflow2

Motivation

- HyFlow is our previous DTM framework. It was hard to maintain:
  - Poorly designed component interfaces encouraged hard-coded links between modules.
  - Annotation based API required bytecode rewriting, slowing down development of new features.
- Bytecode rewriting makes centralized STM fast.
  - However in DTM, local execution costs are small in comparison to the cost of distribution.
  - Supporting bytecode rewriting is misplaced effort.

Hyflow2

- Our second generation Distributed Transactional Memory framework.
- Written in Scala, for the JVM (Java, Scala, etc.)
- Library based approach. Most features run on stock JVM, without byte-code rewriting.

API

- Clean API, based on ScalaSTM

```
class Acc extends HObj {
    val amount = field(0)
}
object Bank {
    def transfer(src: String, dst: String,
                 v: Int) {
        atomic { implicit txn =>
            val a1 = Hyflow.dir.open[Acc](src)
            val a2 = Hyflow.dir.open[Acc](dst)
            a1.amount() = a1.amount() - v
            a2.amount() = a2.amount() + v
        }
    }
}
```

API

- Refs are containers for transactional data.
  - Get value using ref.get() or ref.apply()
  - Set value using ref.set() or ref.update()
  - Scala syntactic sugar for the latter. E.g. ctr() = ctr() + 1 is ctr.update(ctr.apply() + 1)
- The **atomic** "keyword" is just a method call:
  - Call method atomic.apply()
  - Pass as parameter an anonymous function (the transaction body).
- Directory manager for *opening* objects.

Nested Transactions and Checkpointing

Nested Transactions

- Nesting is used to enable code composability
  - Transaction enclosed within another transaction
- Three types, based on parent/children interactions:
  - Flat nesting: monolithic transactions
  - Closed nesting: children can abort independently
  - Open nesting: child releases isolation early
- Closed nesting is a solution for implementing transactions' partial abort

Nested Transactions API

- Nested atomic blocks just work.
  - Runtime chooses model (flat or closed) based on configuration.
- Open nesting must be requested explicitly.

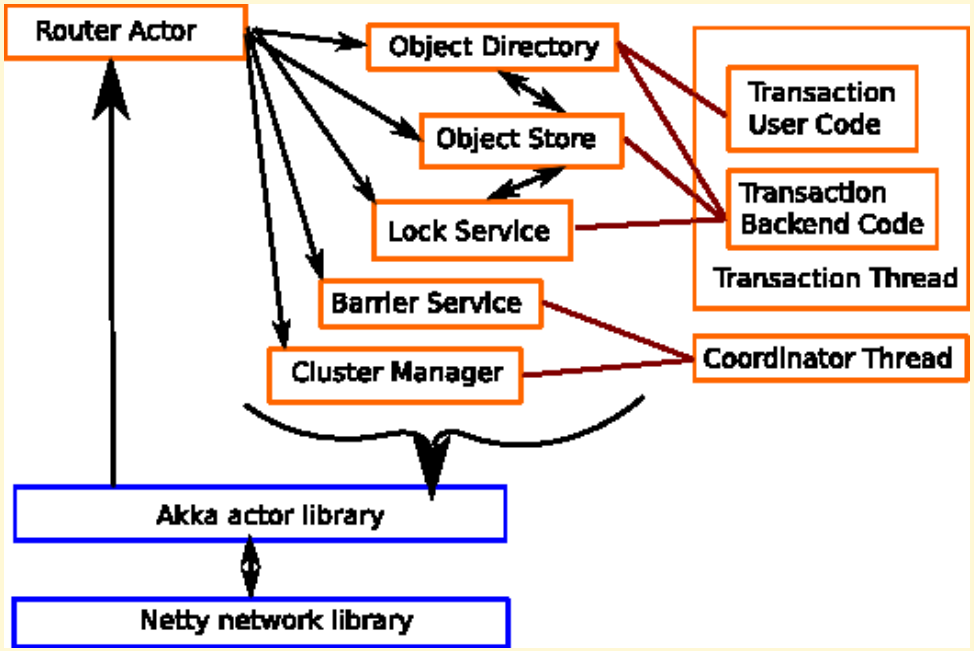
```
atomic.open { implicit txn =>
        val ctr = Hyflow.dir.open[Counter]("id")
        ctr.value() += 1
} onAbort { implicit txn =>
        val ctr = Hyflow.dir.open[Counter]("id")
        ctr.value() -= 1
} onCommit { implicit txn =>
        //...
}
```

Checkpointing

- Fine-grain partial abort mechanism
  - Transaction state is saved each time a new shared object is accessed.
  - In case a conflict happens, transaction can rollback to any previously saved checkpoint.
  - Checkpointing identifies the invalid object and restart the transaction just before the first access of that object.

Implementation

Architecture



- Based on the actor model (message passing).
- High-performance, actively maintained libraries:
  - Akka (actor library)
  - Netty (asynchronous networking)
  - Kryo (serialization).

Transaction Checkpointing

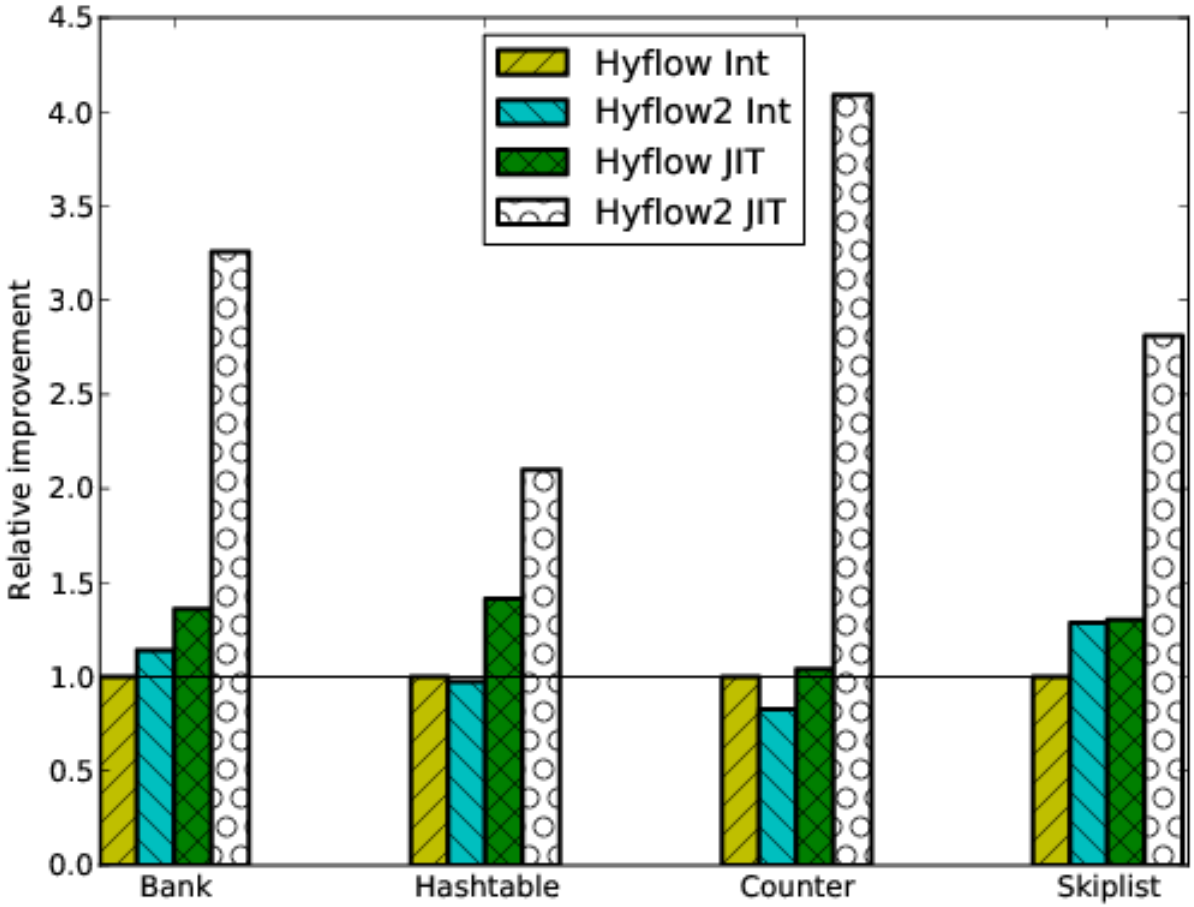
- Uses **continuations**, a mechanism for controlling program flow.
  - Similar to *getcontext*/*setcontext* in C.
  - Not available in stock JVM, needs patched JVM from the DaVinci VM Project.
  - Library-based approach possible (e.g., JavaFlow, NightWolf), but was discarded due to low performance and Scala compatibility issues.

Evaluation

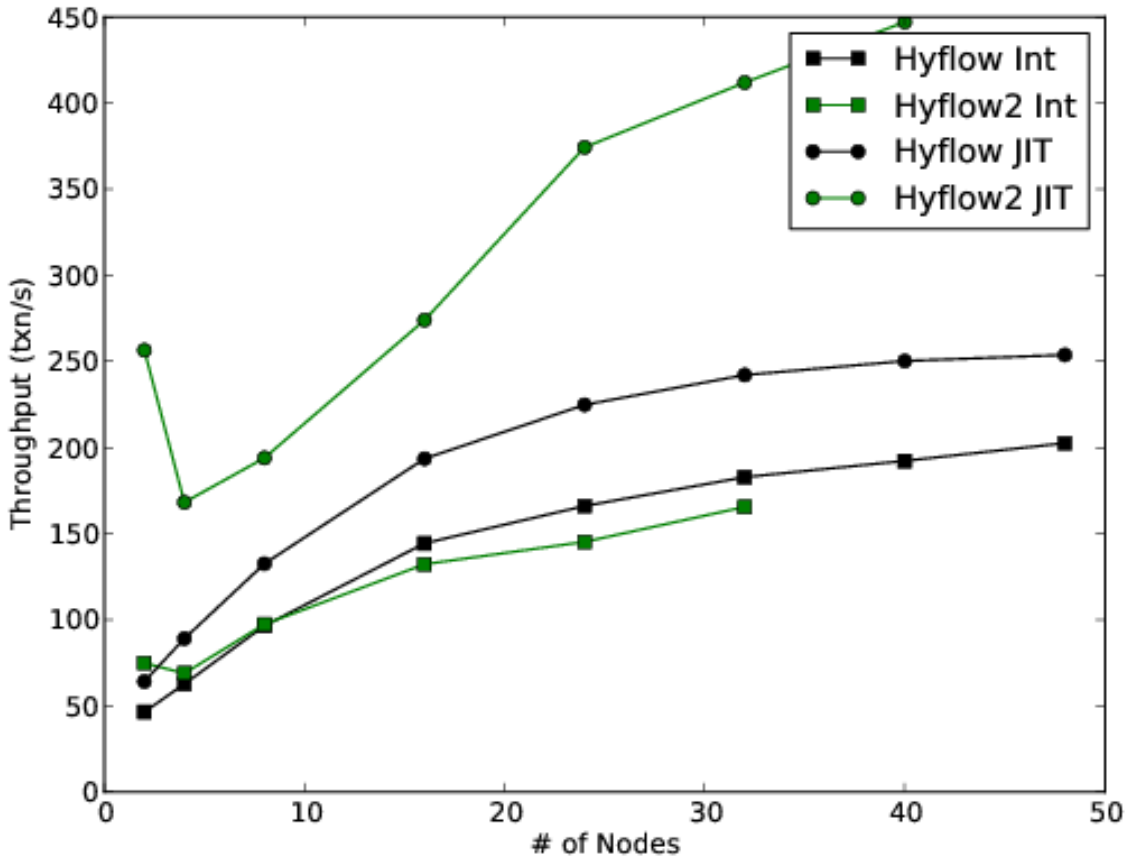
Configuration

- One benchmark (bank) and three micro-benchmarks (enhanced counter, skip list, hash table), configured with high contention.
- Competitor is the original Hyflow (which implements the same algorithm, TFA).
- Up to 48 nodes.

Results



- Bank, 80% reads, absolute



Conclusions

- We introduced Hyflow2, a high performance DTM framework written in Scala.
- Supports for nesting and checkpointing.
- Modular to allow for rapid prototyping.
- Hyflow2 available at: www.hyflow.com

- Systems Software Research Group: www.ssrg.ece.vt.edu

Systems Software Research Group

- Thank you! Questions?