SNAKE

CONTROL FLOW DISTRIBUTED SOFTWARE TRANSACTIONAL MEMORY

Mohamed Saad and Binoy Ravindran

Electrical & Computer Engineering Department
Virginia Tech

SSS'11 Grenoble, France

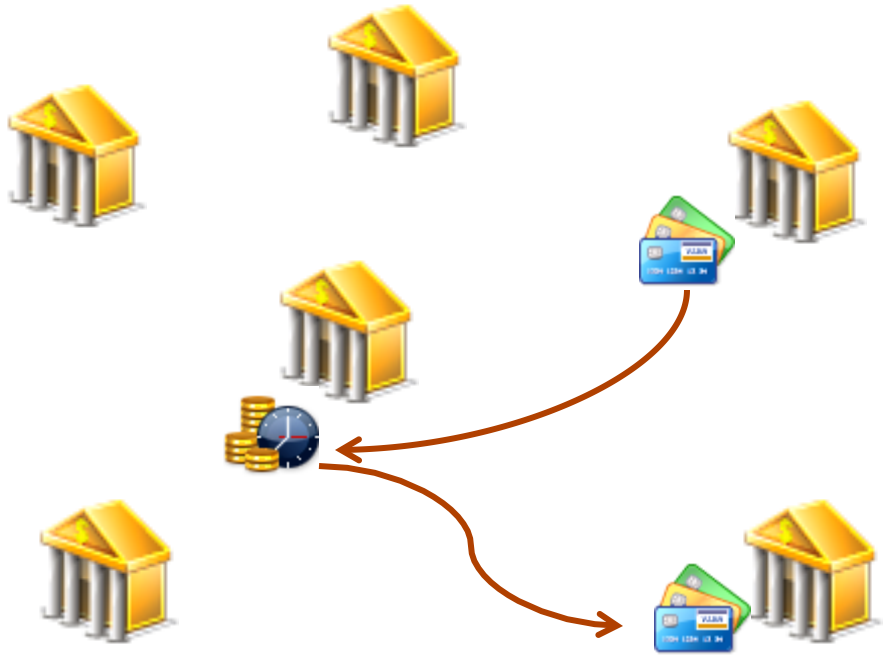
Distributed Atomicity

System is deployed on a set of distributed nodes with message passing links

An operation (or set of operations) appears to the rest of the system to occur instantaneously

Example (Money Transfer):

.....
.....
from = from - amount
to = to + amount
.....
.....



Distributed Atomicity

Locking – traditional approach

Locks attached to objects

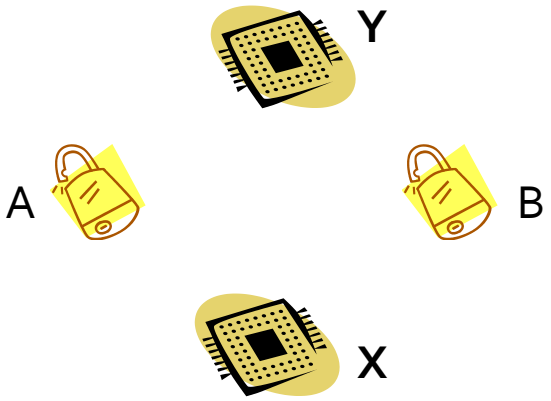
APIs for remote access locks

Drawbacks

- Distributed deadlock
- Distributed livelock
- Starvation
- Priority inversion
- Composability
- Scalability

Example (Money Transfer):

```
.....
.....
account1.lock()
account2.lock()
from = from - amount
to = to + amount
account1.unlock()
account2.unlock()
.....
.....
```



Distributed Software Transactional Memory

Transactional Memory (TM)

Simplifies concurrency control by allowing a group of instructions to execute **atomically** using additional primitives (e.g., transaction_begin & transaction_end)

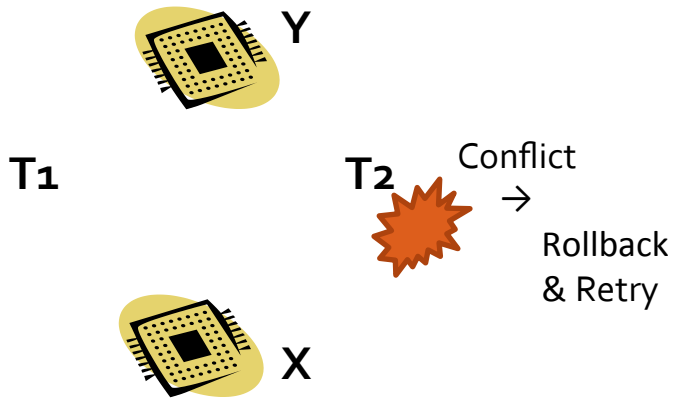
Distributed TM

Generalization of TM to distributed environments

Not a silver bullet

Example (Money Transfer):

```
.....
.....
transaction_begin
from = from - amount
to = to + amount
 transaction_end
.....
.....
```



(D)STM Mechanisms

Versioning

```
.....
transaction_begin
from = from - amount
.....
 transaction_end
.....
```

Where to store new *from* until commit?

Eager: store new at original location; old in an *undo log*

Lazy: store new in a transaction-local *write-buffer*

Conflict detection

To	T1
.....	.....
transaction_begin	**transaction_begin**
from = from - amount	from = from + amount
.....	.....
transaction_end	**transaction_end**
.....	.....

How to detect conflict between To and T1?

Record read and write locations in read and write sets

Conflict if one's read or write set intersects with the other's write set

(D)STM Mechanisms

Contention management

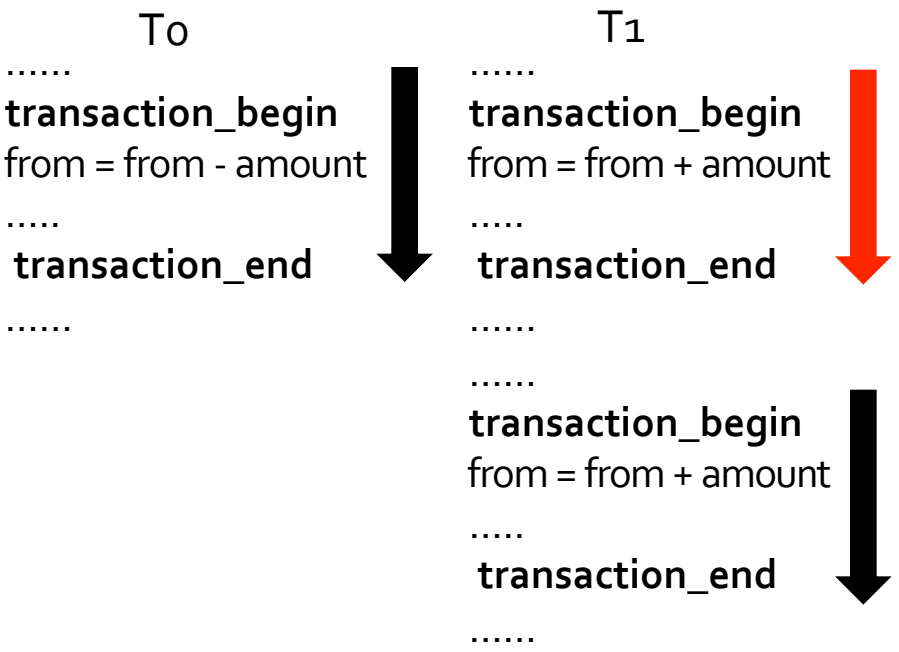
Which transaction to abort/retry?

Backoff

Priority

Karma

.....



STM Implementations

Hardware Transactional Memory

Modifications in processors, cache and bus protocols
e.g., unbounded HTM [11], TCC, ...

Software Transactional Memory

Software runtime library, programming language support
Minimal hardware support (e.g., CAS, LL/SC)
e.g., RSTM, DSTM, Deuce, ESTM, ..

Hybrid Transactional Memory

Exploits HTM support to achieve hardware performance for transactions that do not exceed HTM's limitations, and STM otherwise
e.g., LogTM [16], HyTM, ...

Distributed Software Transactional Memory (D-STM)

Extends STM to work in distributed environments
e.g., Cluster-STM [5], D^{2}STM [7], DiSTM [14], ...

Snake

- D-STM implementation exploiting control-flow execution model (immobile objects and mobile transactions)
- Extends Java Remote Method Invocation (RMI) architecture
- Uses annotations and code generation (using run-time instrumentation) to support atomicity/remote access
- No recompilation, or changes to underlying virtual machine
- Objects versions used to track object state

Programming Model

Annotation-based

@Remote

@Atomic

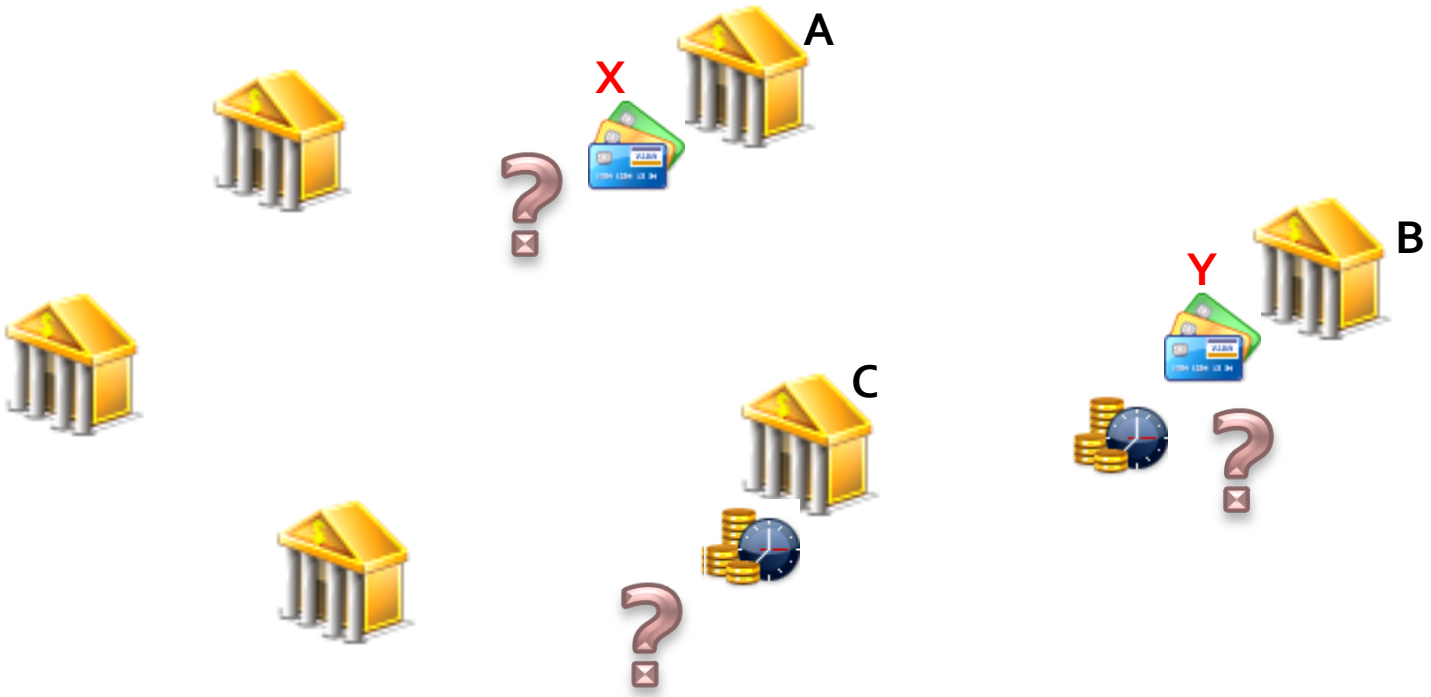
(Inspired by Deuce STM)

```
Class BankAccount{

    @Remote
    public void withdraw(int amount){
        this.amount -= amount;
    }

    @Remote
    public void deposit(int amount){
        this.amount += amount;
    }

    @Atomic
    public static void   transfer(
            BankAccount  acc1,
            BankAccount  acc2,
            int amount){
        .....
    }
}
```

Control Flow

Immobile objects, mobile transactions

Distributed commit needed for commit/abort decision



Algorithms

Transactions move between nodes, while objects are immobile

Each node has a portion of a transaction's read and write sets

Transaction metadata are detached from the transaction context

Distributed validation at commit using a voting mechanism

Default is D2PC [18]

Algorithms

Undo Log (Eager/Pess.)

On Write

If (owned) resolve
set owned by me
Backup and change in master copy

On Read

If (owned) resolve
Read value and version

Try Commit

Validate reads (version < current)

On Commit

Increment owned versions
Release owned

On Rollback

Undo changes for owned
Release owned

Write Buffer (Lazy/Opt.)

On Write

Change in private copy

On Read

If (in write-set) read local value
else read master copy value
Read version

Try Commit

Acquire ownership of write-set
Validate reads (version < current)

On Commit

Write values to main copy
Increment owned versions
Release owned

On Rollback

Discard local changes

Distributed Contention Management

- Contention managers can be classified into categories:
  - **Incremental** builds up priorities of transactions during transaction execution
    - E.g., Karma, Eruption, Polka
  - **Progressive** ensures system-wide progress (i.e., at least one transaction will proceed to commit)
    - E.g., Kindergarten, Priority, Timestamp, Polite
  - **Non-Progressive** assumes that conflicting transactions will eventually complete (livelocks can occur)
    - E.g., Backoff, Aggressive

Distributed Contention Management

- CM behavior under control flow D-STM
  - **Incremental.** Transactions can have different priorities at each node, as a transaction builds its priority during its execution over multiple nodes → livelocks
  - **Non-Progressive.** Works for non-distributed TM, however, aborts without progress guarantees is costly in distributed environments
  - **Progressive.** Most appropriate for control flow
  - Empirical evidence

Evaluation

120 nodes, 1.9 GHz each, 0.5~1 ms network delay

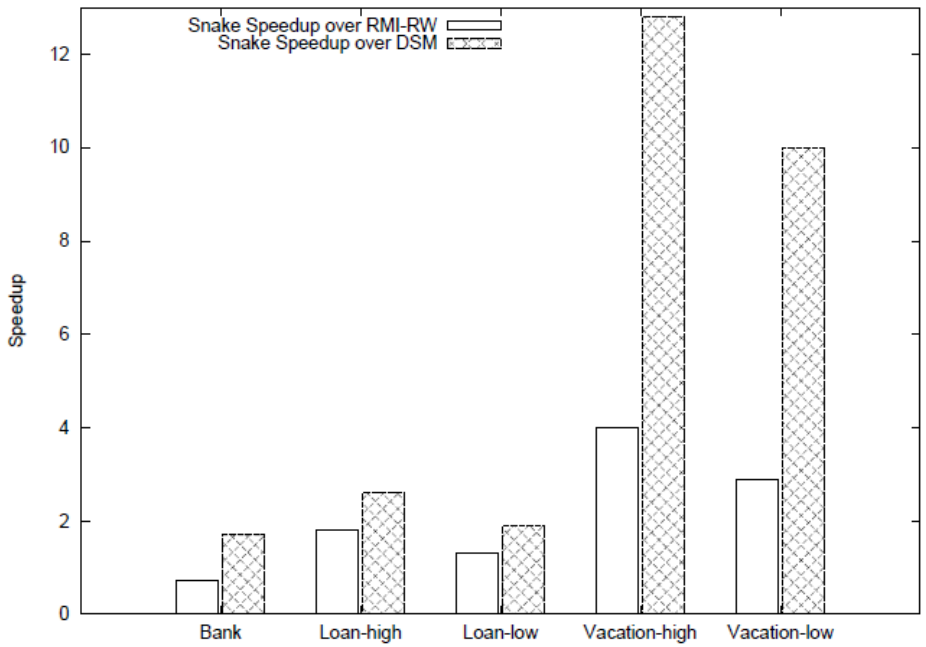
8 threads per node (~1000s of concurrent transactions)

50-200 sequential transactions

4 million transactions

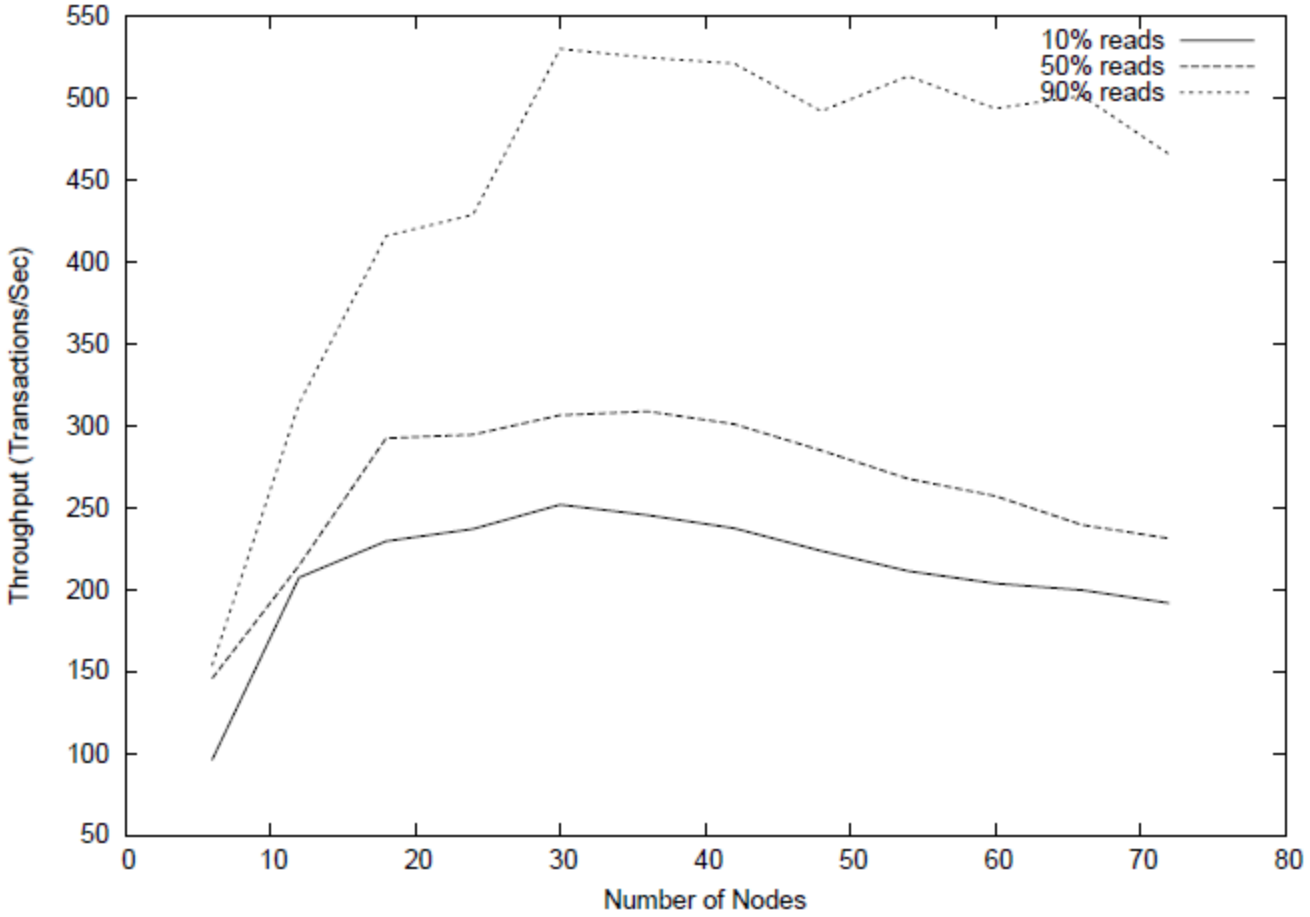
5% confidence interval (variance)

Use 4 distributed benchmarks: Bank, Loan, P2P Search Agent, Vacation



Evaluation

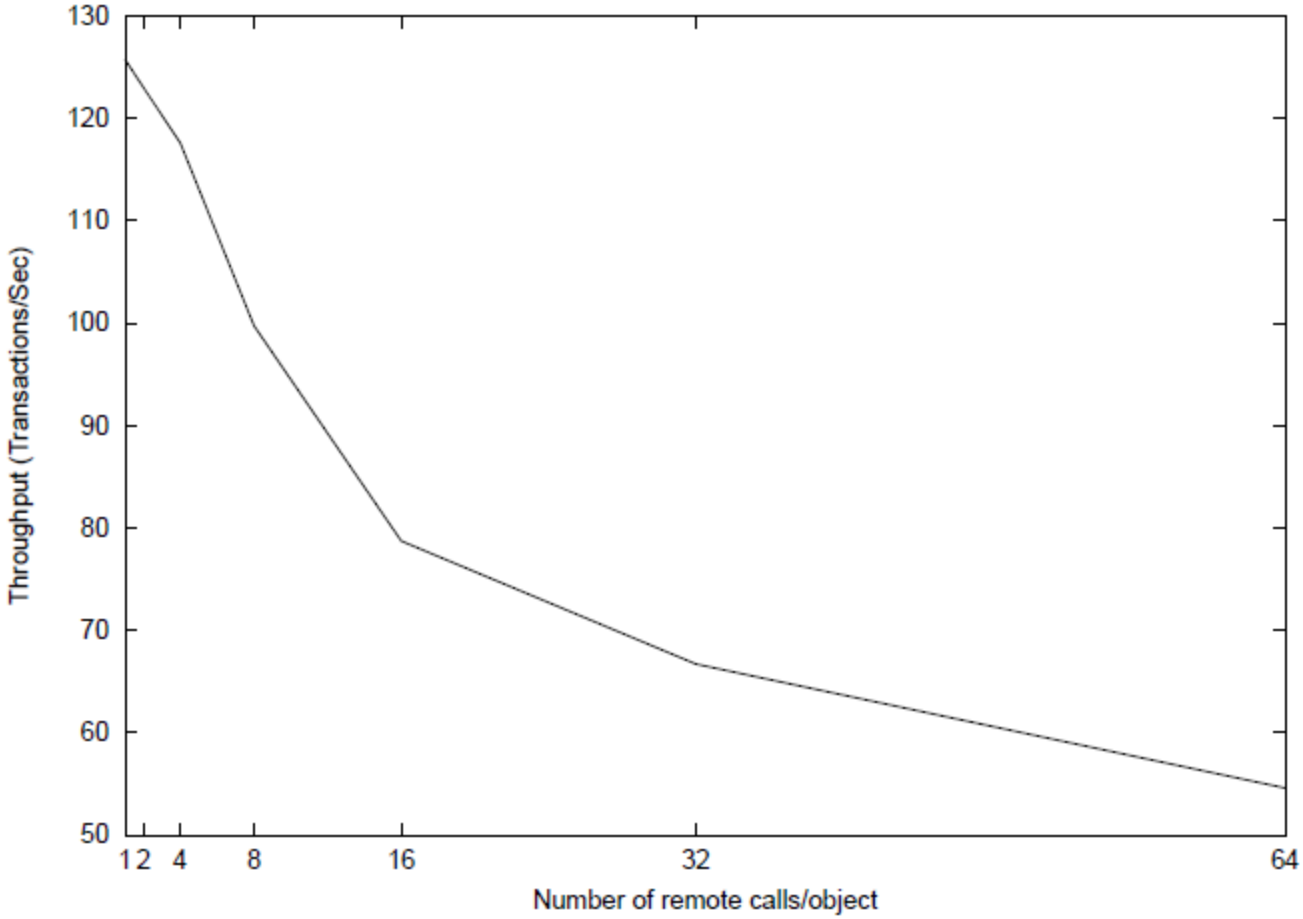
P2P Benchmark



Evaluation

Locality (Dataflow vs. Control-flow)

Bank Benchmark



Conclusions

- Snake DSTM, a control-flow D-STM
  - Transactional meta-data is detached; uses distributed commit
- Outperforms other distributed concurrency control models (for models and benchmarks studied)

- Control flow is beneficial under

  non-frequent object calls, or

  when objects must be immobile due to object state dependencies, object sizes, or security restrictions

Future work

- Production application case studies
- Mechanisms for (distributed) transactional nesting
- Techniques and mechanisms for multi-version control flow D-STM
- ….

- Snake implementation is available at hyflow.org