On Open Nesting in Distributed Transactional Memory

Alexandru Turcu* Binoy Ravindran*

June 4, 2012

*Virginia Tech

Overview

- Introduction
  - Distributed Transactional Memory
  - Nested Transactions
- Open Nesting
- TFA with Open Nesting (TFA-ON)
  - Transactional Forwarding Algorithm
  - TFA-ON
- Experimental analysis
- Conclusions

Introduction

Distributed Transactional Memory

- Promising new model for programming distributed concurrency
  - Aims to replace distributed locks
  - Avoids their problems: distributed dead-locks, live-locks, difficult code composition
- Employs transactions
  - Successful abstraction originating in the database community
  - Provide atomicity, consistency, isolation

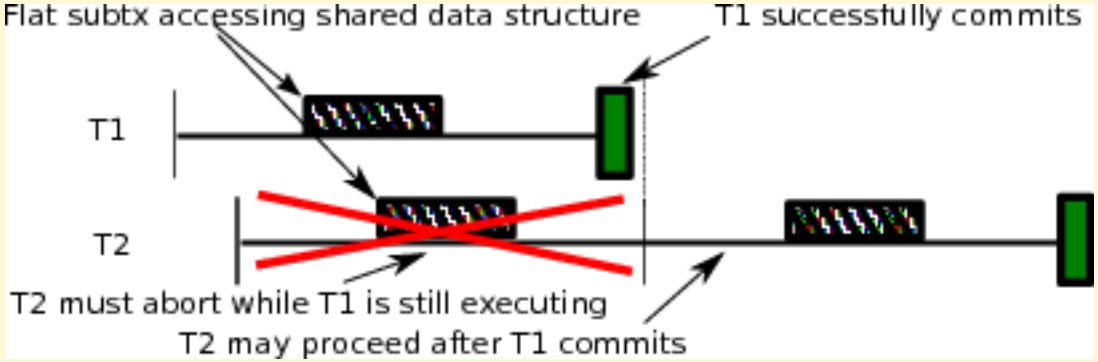
Introducing Transactions

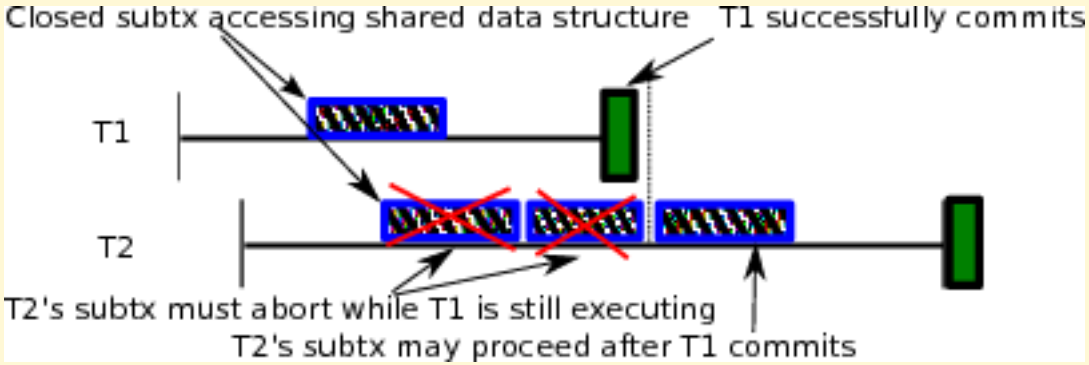
- We consider a redo-log approach:
  - Write ops are buffered to a write-set
  - Read ops first look at the write-set
  - Read ops are recorded in a read-set $\rightarrow$ used for validation (to make sure objects did not change since first seen)
- On commit, changes are propagated to the globally committed memory
- On abort, changes are discarded and the transaction is retried

Nested Transactions

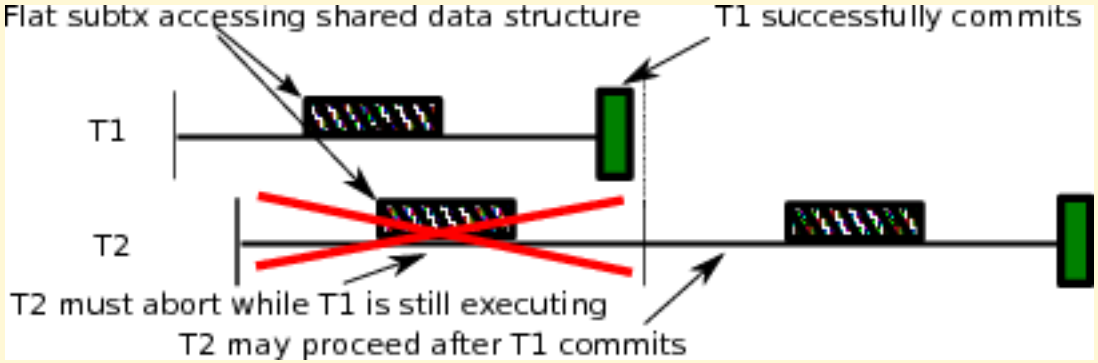
- Nesting is used to enable code composability
  - Transaction enclosed within another transaction
- Three types, based on parent/children interactions:
  - Flat nesting (monolithic transactions), conflict in child aborts parent
  - Closed nesting, children can abort independently
  - Open nesting, child releases isolation early

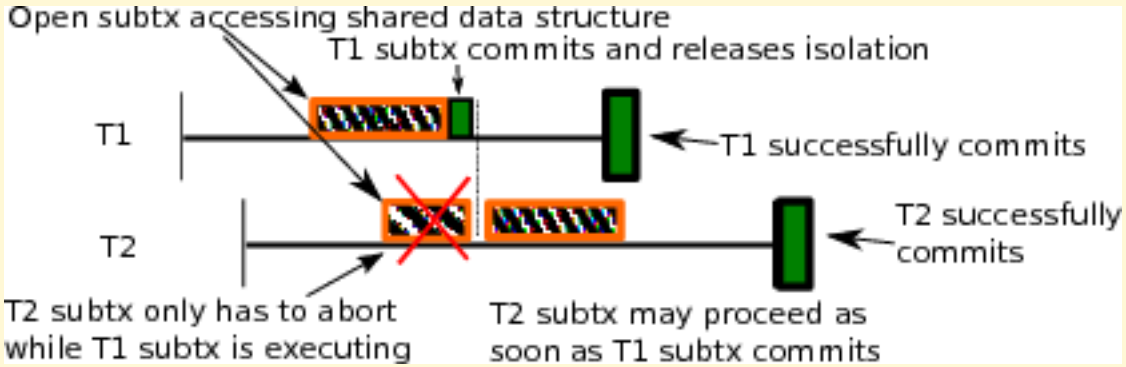
Flat vs Closed nesting





Flat vs Open nesting

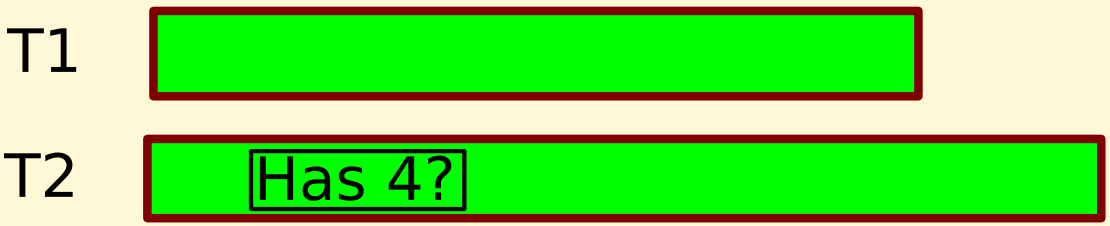




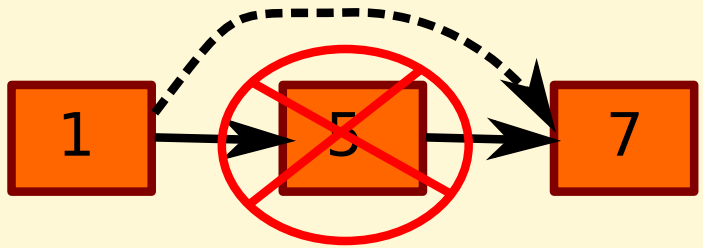
Open Nesting

Multilevel Transactions

- Why? Consider an example:
  - Set implemented using a skip-list
  - Operations accessing neighboring nodes conflict
  - For short transactions, this is OK
  - Long transactions would abort in vain.



- Why? Consider an example:
  - Set implemented using a skip-list
  - Operations accessing neighboring nodes conflict
  - For short transactions, this is OK
  - Long transactions would abort in vain.



Multilevel Transactions

- False conflicts:
  - Memory operations do conflict
  - Conceptually, no conflict
- Separate transactions into multiple levels of abstraction:
  - Make transactions at the lower (memory) level shorter, releasing isolation earlier
  - Preserve **fundamental conflicts** at the higher level
    - When two ops can not commute (e.g., *Has 5?* and *Del 5!*)
    - Detect using other mechanisms (e.g., abstract locks)

Multilevel transactions

- When parent needs to abort after child committed:
  - Revert the data structure to its original (abstract) state by applying a compensating action
  - Example: to compensate for adding 5 to a set $\rightarrow$ remove 5
- Abstract locks:
  - Do not lock physical memory locations, thus *abstract*
  - Can be used to implement mutual exclusion, R/W, etc.

Open Nesting Safety

- Because isolation is released early, maintaining correctness becomes the task of the programmer
  - Correct usage of abstract locks is paramount
  - Open nesting recommended for experts only (e.g., library devs)

TFA-ON

Transactional Forwarding Algorithm

- TFA is an existing protocol for distributed STM
  - Based around Transactional Locking II and Lamport clocks
- Provides a way to establish "happens before" relationships
  - Each node holds a node-local clock
  - Clock value affixed to all messages
  - Clock incremented on local transactions' commits
  - When a message from a node with a higher clock is received, local clock is updated

Transactional Forwarding Algorithm

- Each txn stores its starting time
- When a txn communicates with a node with a higher clock:
  - Attempt to update txn's starting time (i.e. *transactional forwarding*)
  - Must validate read-set before forwarding
    - Success $\rightarrow$ update txn starting time and continue
    - Failure $\rightarrow$ abort txn

TFA-ON

- Our contribution: Transactional Forwarding Algorithm with Open Nesting (TFA-ON), implemented in HyFlow Java DTM framework.
- Open nested transactions in TFA-ON resemble the root transactions of TFA:
  - They record the local clock when they start
  - They increment the local clock upon commit
- For closed nested sub-transactions, the closest open-nested ancestor acts as a *local root*
  - Objects in all levels underneath this *local root* are considered for validation
  - When performing transactional forwarding, the starting time of the *local root* is updated

Abstract Locks in TFA-ON

- Abstract lock acquisition:
  - With respect to the open-nested child → late acquisition (at commit time)
  - With respect to the parent transaction → early acquisition (encounter time)
- Deadlocks are possible
  - We chose to abort the whole transaction chain when an abstract lock acquisition fails.
  - This ensures all abstract locks are released, avoiding deadlocks.

Defining Transactions

```
new Atomic<Boolean>(NestingModel.OPEN) {
 @Override boolean atomically(Txn t) {
  // ...
  t.acquireAbsLock(bst, lockName);
  return true;
 }
 @Override void onAbort(Txn t) {
  // ...
  t.releaseAbsLock(bst, lockName);
 }
 @Override void onCommit(Txn t) {
  t.releaseAbsLock(bst, lockName);
 }
}.execute();
```

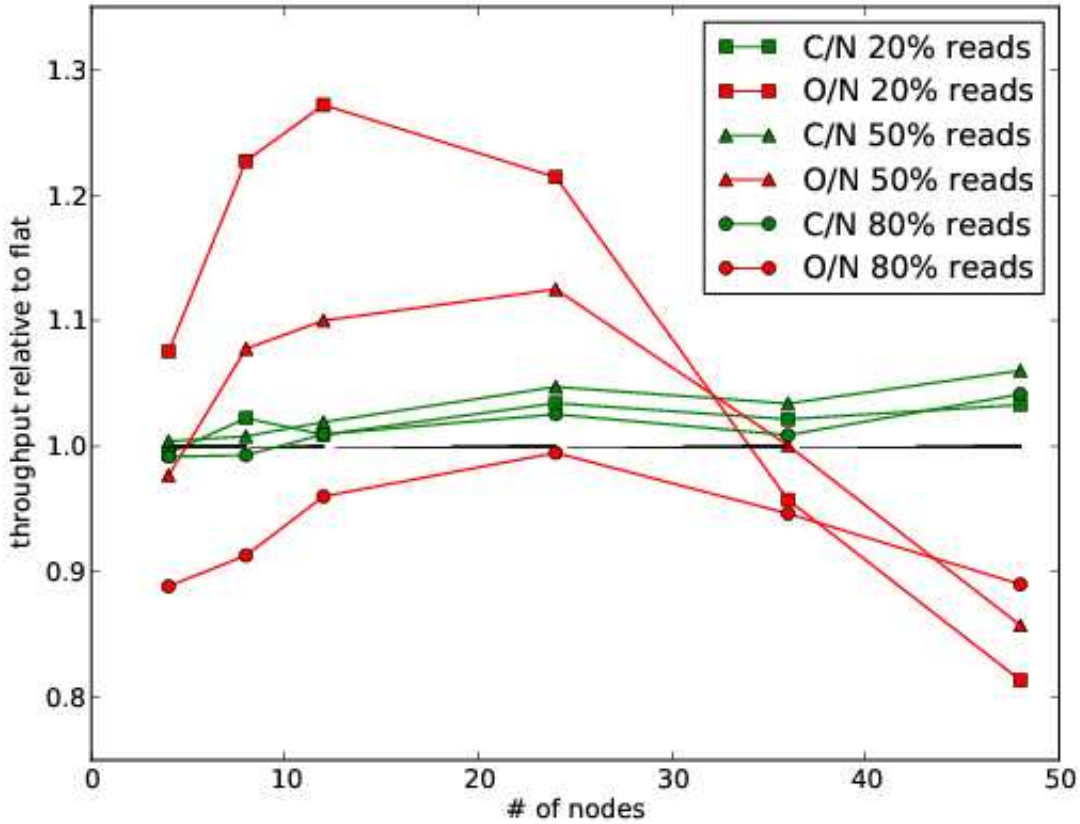
Experiments

Setup

- Micro-benchmarks
  - Three distributed data structures (skip-list, hash-table, binary search tree)
  - Enhanced counter application
- Parameters:
  - Read-only ratio (r)
  - Number of calls (c)
  - Key domain size (k)
- Measured quantities (among others):
  - Throughput
  - Committed/aborted transactions, sub-transactions, commit/compensating actions
- 48-node AMD Opteron testbed running Linux

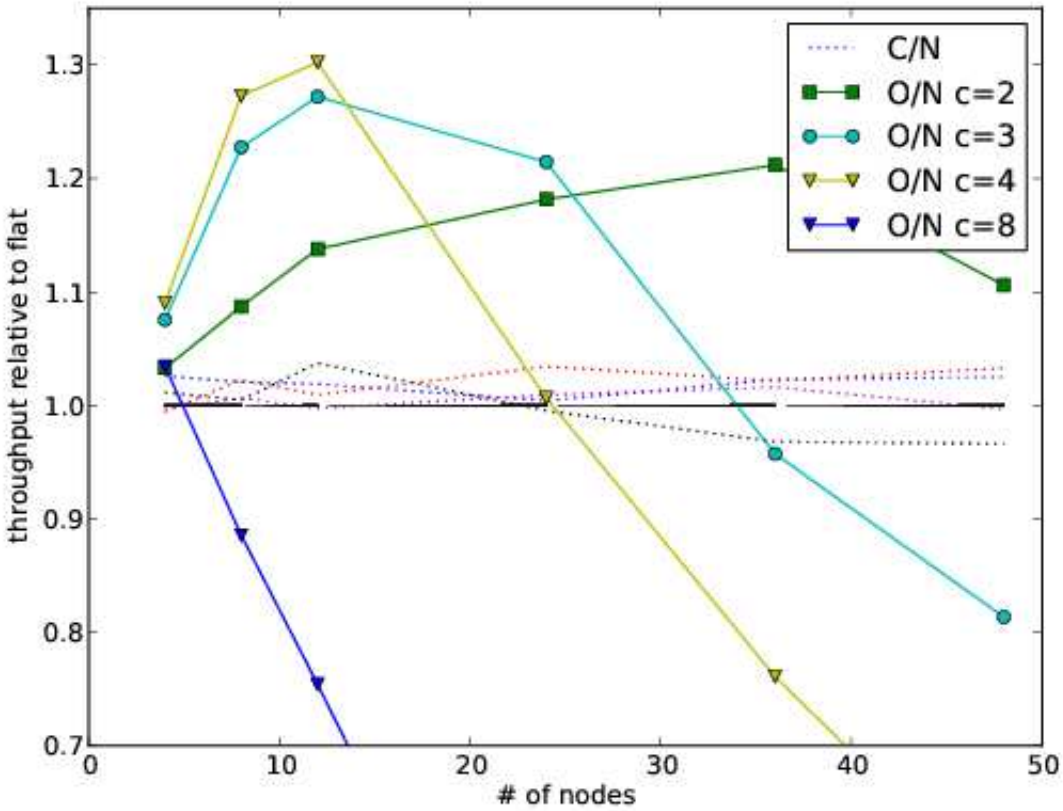
Results: Throughput

Throughput relative to flat, on skip-list, with c=3



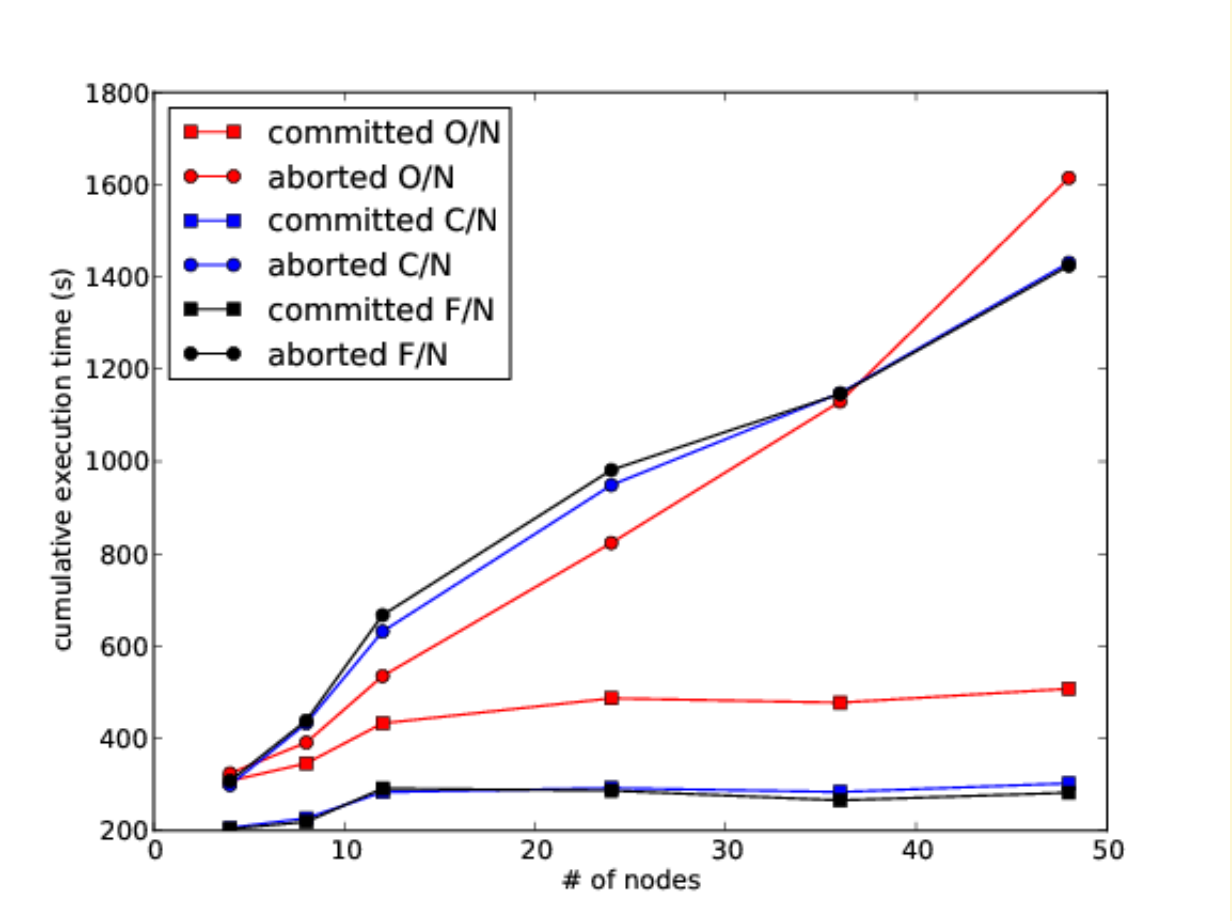
Results: Throughput

Throughput relative to flat, on skip-list, with 20% reads



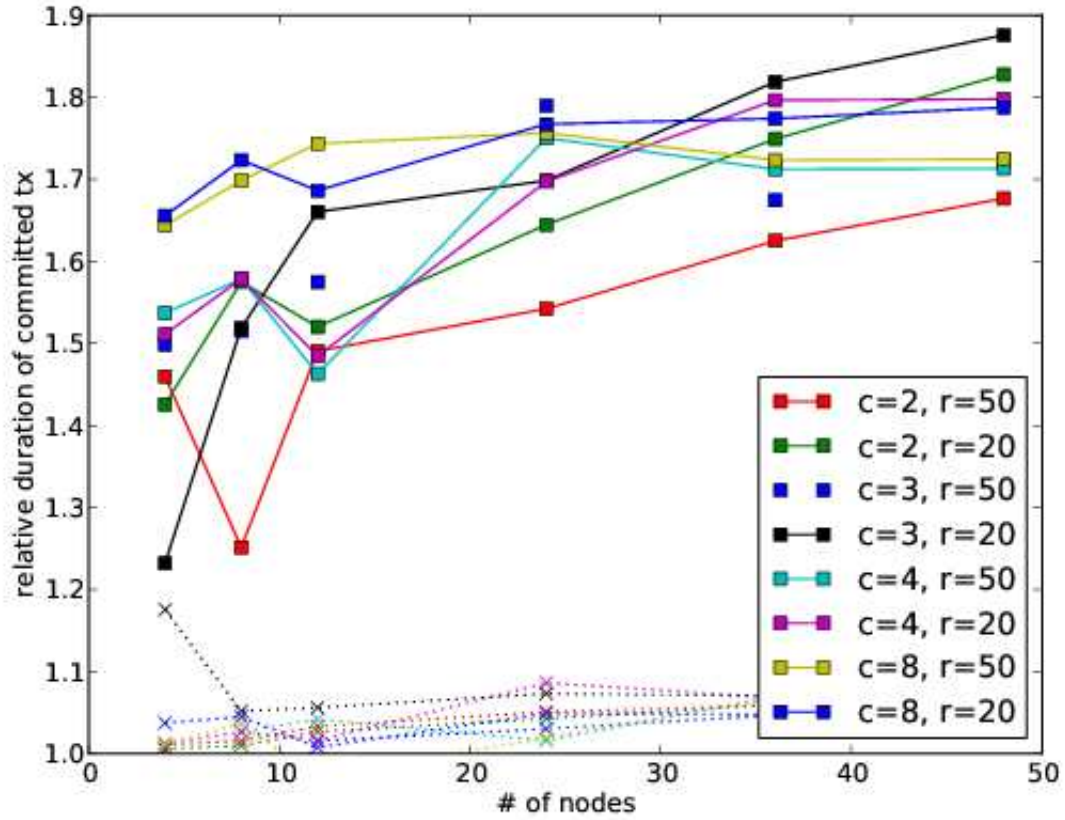
Results: abort vs. commit time

Time in committed vs aborted, hash-table, *r=20* and *c=4*



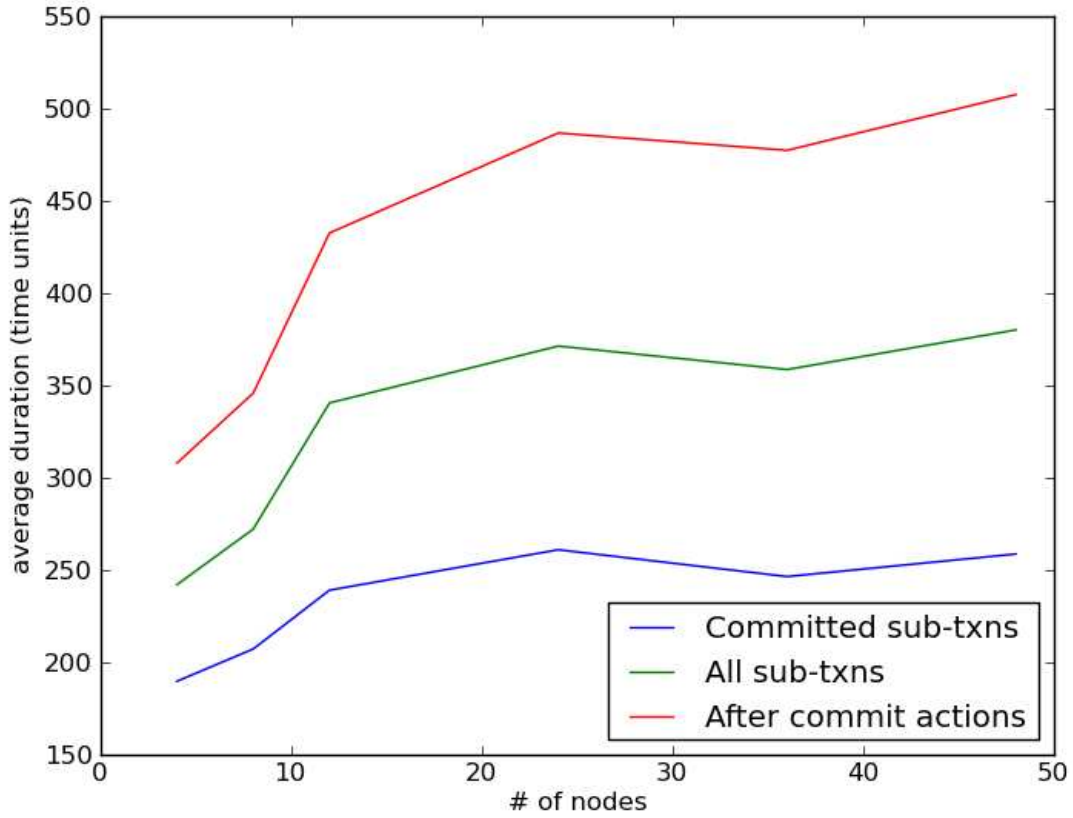
Results: commit overheads

Successful transactions relative overheads, on hash-table



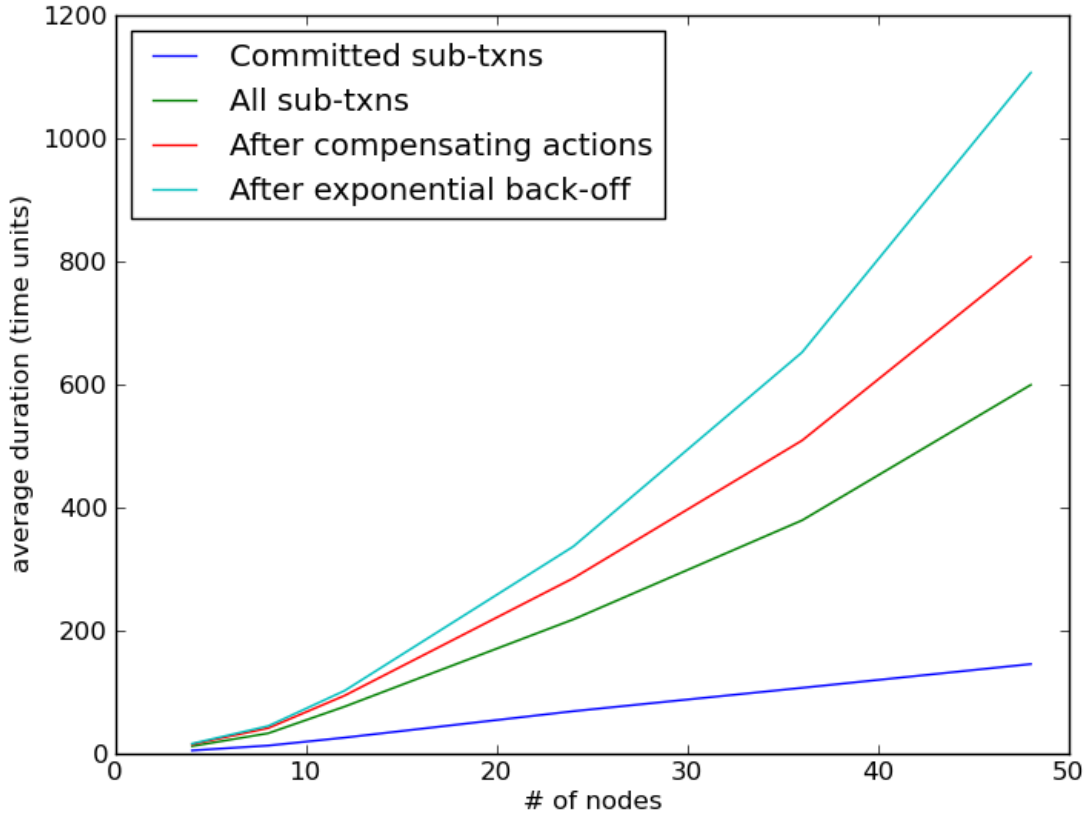
Results: breakdown

Breakdown time in successful txn, hash-table, *r=20, c=4*



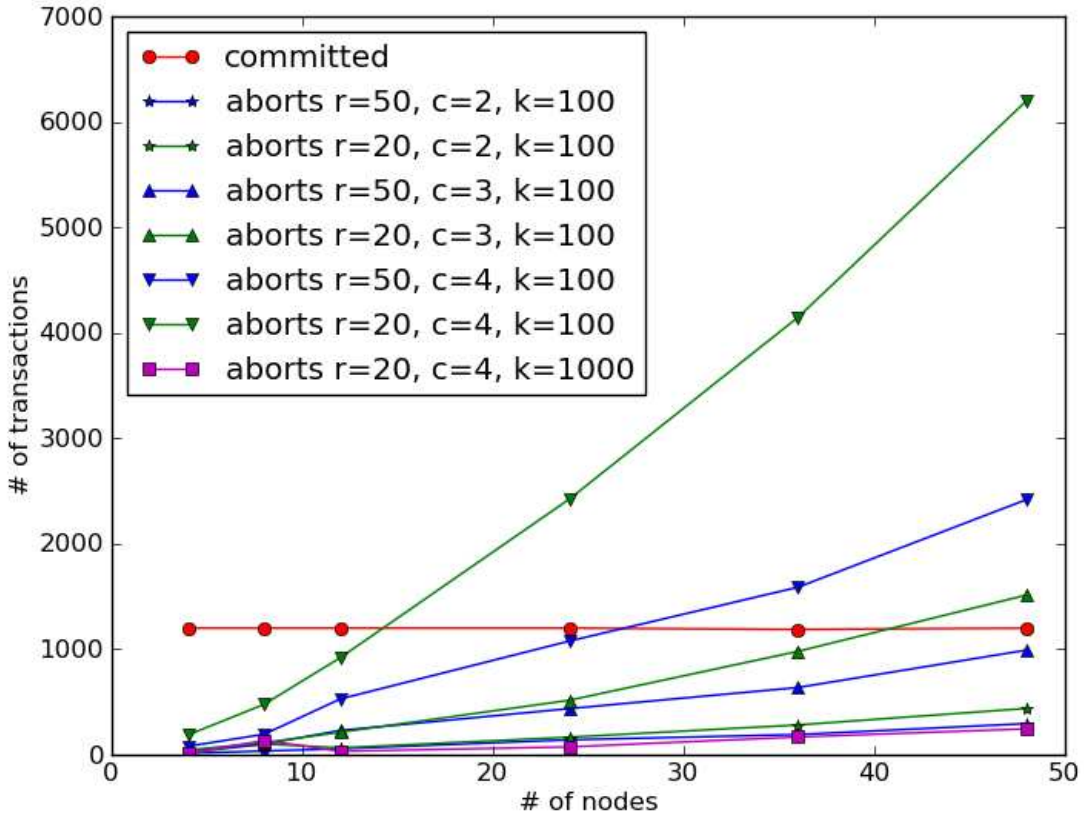
Results: breakdown

Breakdown time in failed txn, hash-table, *r=20, c=4*



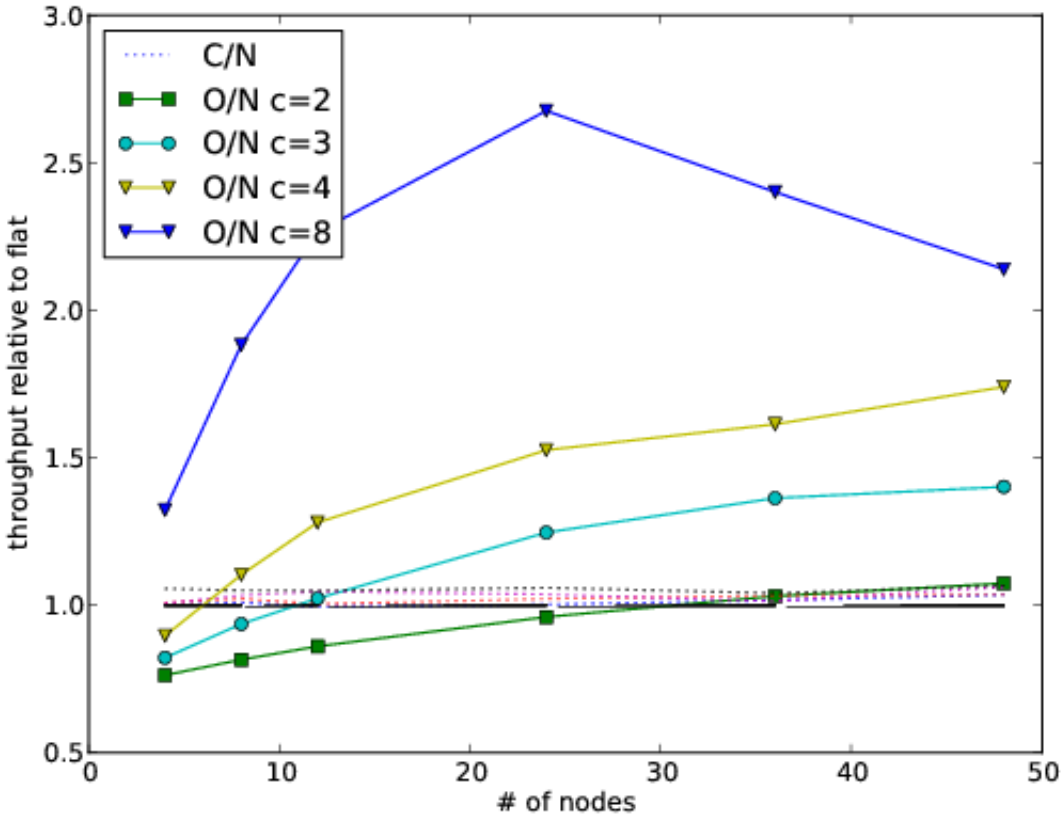
Results: number of aborts

Number of aborts vs commits, on hash-table



Results: increased key-space

Throughput, on hash-table, *r=20, k=1000*



Conclusions

Conclusions

Factors limiting performance:

- Commit overhead at low node-count
  - Significant in read-dominated workloads
- Increased fundamental conflicts at hight node-count
  - Depends on available key-space for abstract locking

Open nesting optimistically assumes the parent will commit. Contention in parent after open-nested child reduces benefit.