Enhancing Concurrency in Distributed Transactional Memory through Commutativity

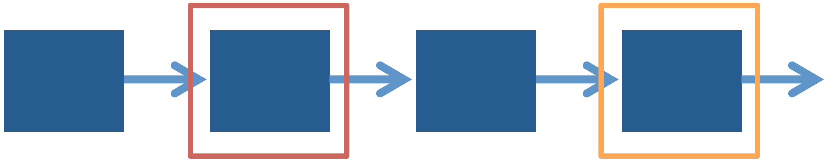
Junwhan Kim, Roberto Palmieri, Binoy Ravindran

Virginia Tech
USA

19th International European Conference on Parallel and Distributed Computing (Euro-Par 2013)

Lock-based concurrency control has serious drawbacks

- Coarse grained locking
  - Simple
  - But no concurrency



```
public boolean add(int item) {
  Node pred, curr;
  lock.lock();
  try {
    pred = head;
    curr = pred.next;
    while (curr.val < item) {
      pred = curr;
      curr = curr.next;
    }
    if (item == curr.val) {
      return false;
    } else {
      Node node = new Node(item);
      node.next = curr;
      pred.next = node;
      return true;
    }
  } finally {
    lock.unlock();
  }
}
```

Fine-grained locking is better, but…

- Excellent performance
- Poor programmability

- Lock problems don't go away!
  - Deadlocks, livelocks, lock-convoying, priority inversion,….

- Most significant difficulty – composition

```
public boolean add(int item) {
  head.lock();
  Node pred = head;
  try {
    Node curr = pred.next;
    curr.lock();
    try {
      while (curr.val < item) {
        pred.unlock();
        pred = curr;
        curr = curr.next;
        curr.lock();
      }
      if (curr.key == key) {
        return false;
      }
       Node newNode = new Node(item);
       newNode.next = curr;
       pred.next = newNode;
       return true;
     } finally {
       curr.unlock();
     }
   } finally {
     pred.unlock();
   }
}
```

Transactional memory

- Like database transactions
- ACI properties (no D)
- Easier to program
- Composable

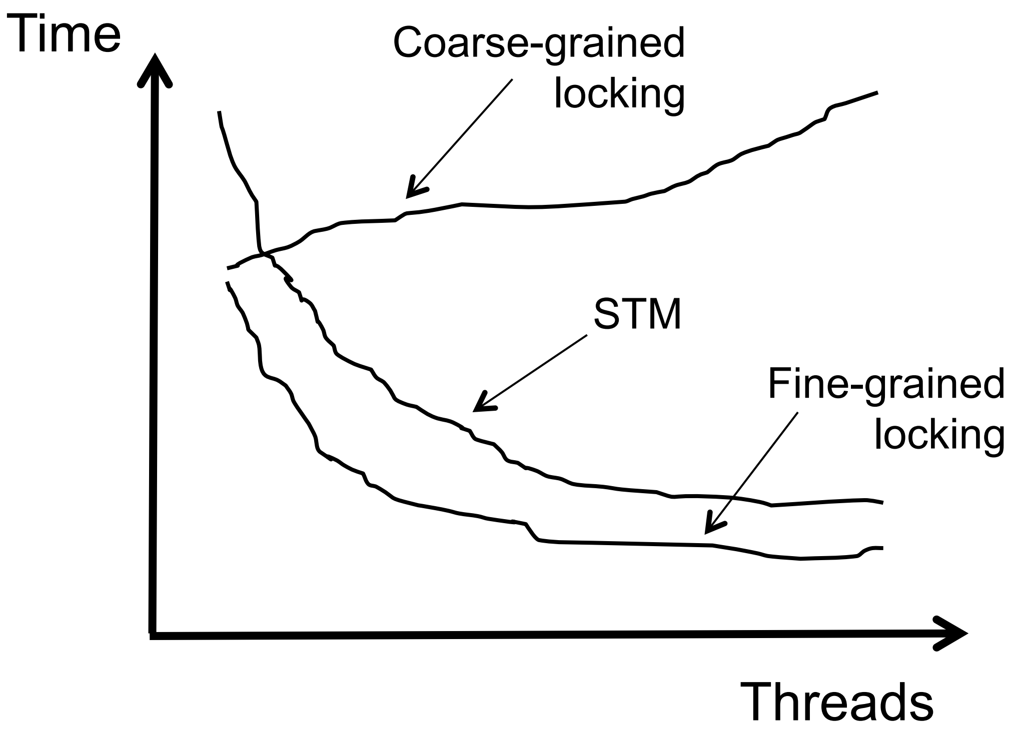
- First HTM, then STM, later HyTM

```
public boolean add(int item) {
  Node pred, curr;
  atomic {
    pred = head;
    curr = pred.next;
    while (curr.val < item) {
      pred = curr;
      curr = curr.next;
    }
    if (item == curr.val) {
      return false;
    } else {
      Node node = new Node(item);
      node.next = curr;
      pred.next = node;
      return true;
    }
}
```

M. Herlihy and J. B. Moss (1993). Transactional memory: Architectural support for lock-free data structures. *ISCA*. pp. 289–300.

N. Shavit and D. Touitou (1995). Software Transactional Memory. *PODC*. pp. 204—213.

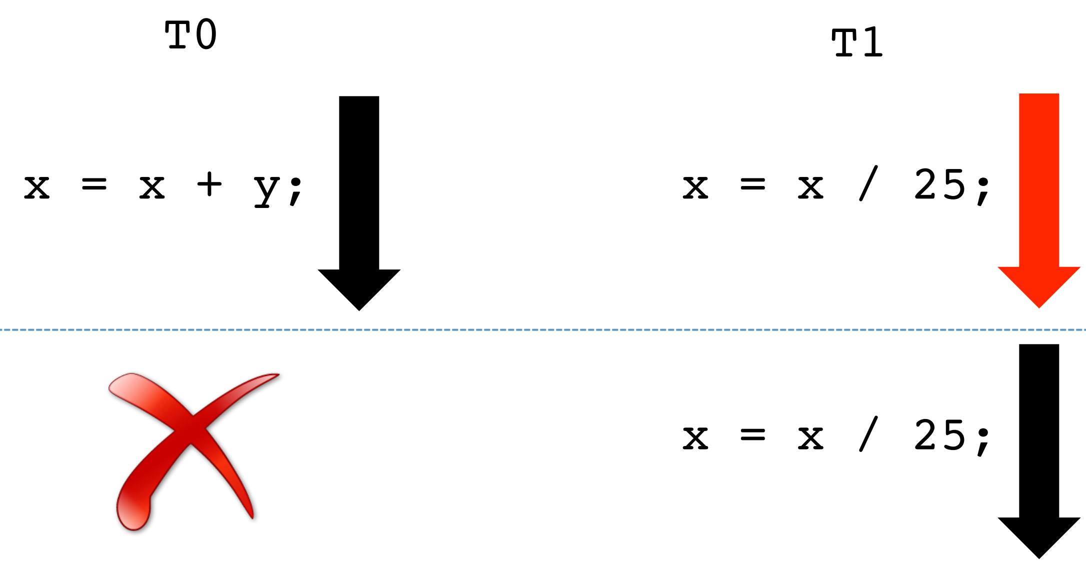
Optimistic execution yields performance gains at the simplicity of coarse-grain, but no silver bullet



- High data dependencies
- Irrevocable operations
- Interaction between transactions and non-transactions
- Conditional waiting
- ……

E.g., C/C++ Intel Run-Time System STM (B. Saha et. al. (2006). McRT-STM: A High Performance Software Transactional Memory. *ACM PPoPP*)

Contention management. Which transaction to abort?



- Contention manager
  - Can cause too many aborts, e.g., when a long running transaction conflicts with shorter transactions
  - An aborted transaction may wait too long
- Transactional scheduler’s goal: minimize conflicts (e.g., avoid repeated aborts)

Distributed TM (or DTM)

- Extends TM to distributed systems
  - Nodes interconnected using message passing links
- Execution and network models
  - Execution models
    - Data flow DTM (DISC 05)
      - Transactions are immobile
      - Objects migrate to invoking transactions
    - Control flow DTM (USENIX 12)
      - Objects are immobile
      - Transactions move from node to node
  - Herlihy's metric-space network model (DISC 05)
    - Communication delay between every pair of nodes
    - Delay depends upon node-to-node distance

1.499 ms	9.095 ms	16.613 ms	13.709 ms	15.016 ms	→ Distance
1st hop	2nd hop	3rd hop	4th hop	5th hop	

Paper’s motivation

How to boost transactions’ processing in DTM

Read/Read
No conflict

Read/Write
conflicts minimized thanks to the multi-versioning support

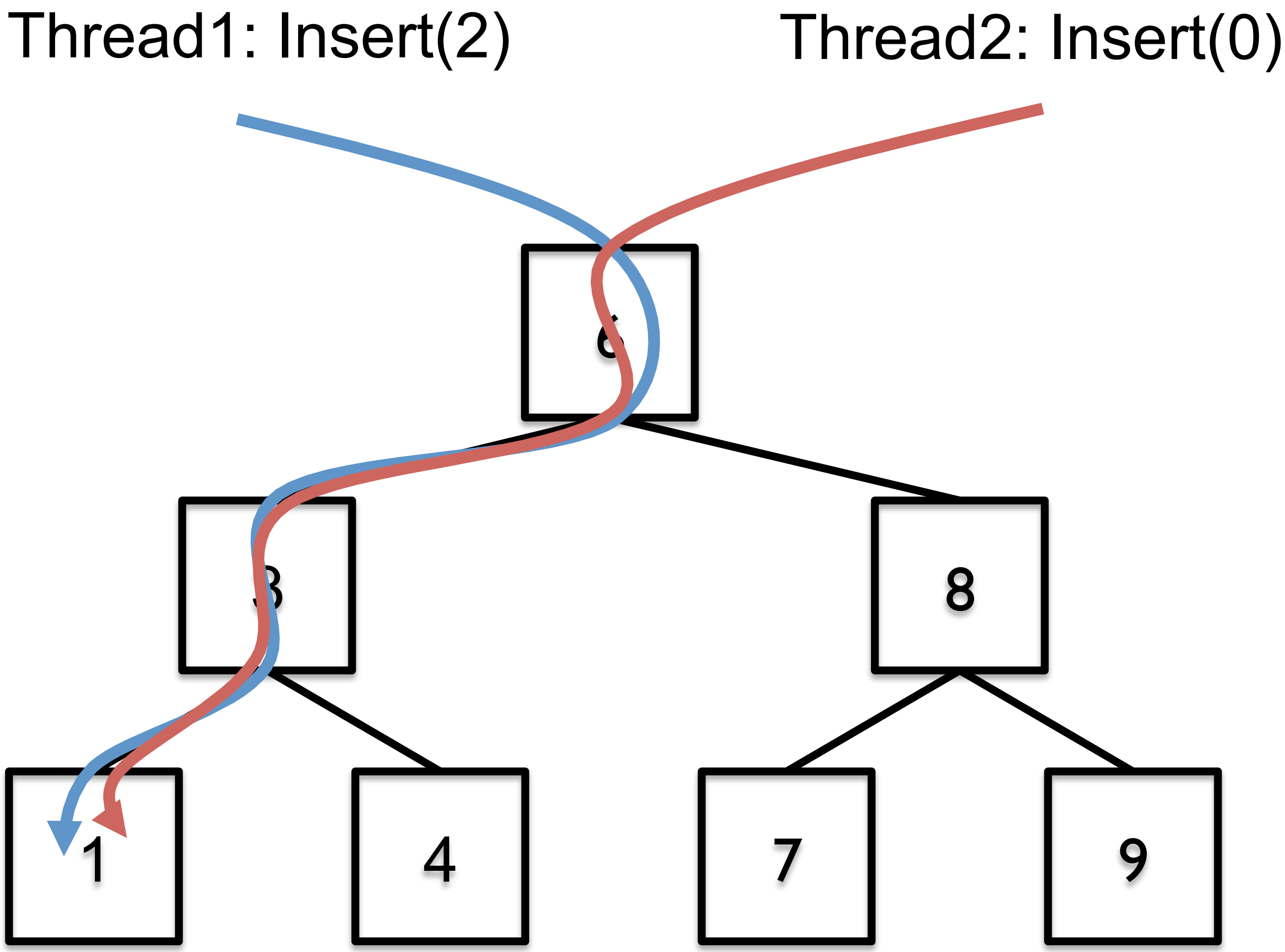
Write/Write
Conflict!

How?

- How can a concurrency control manager allow write conflicting transactions to commit concurrently without affecting isolation/consistency?

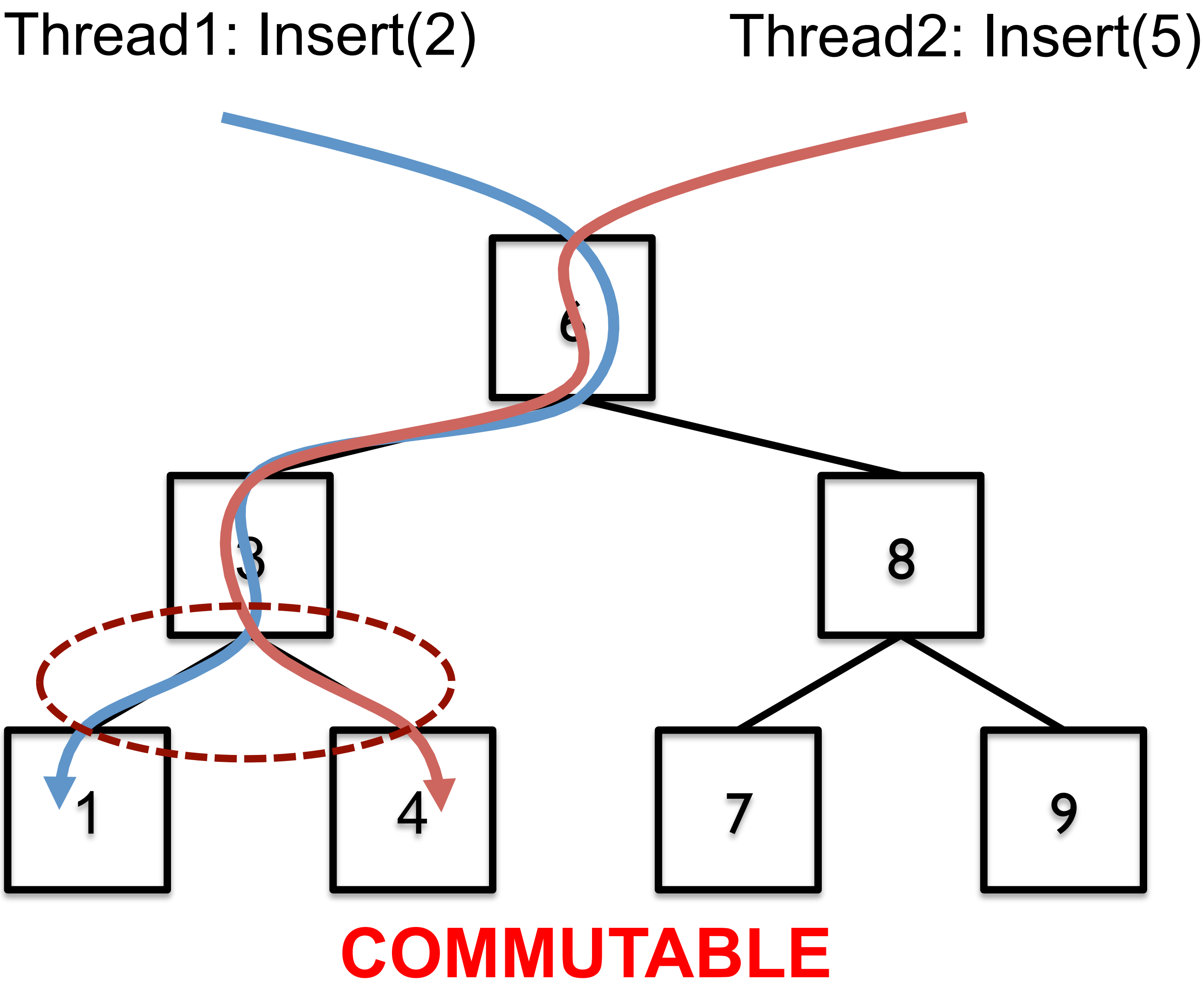
Exploiting commutable operations

Commutable operations by examples (Data Structures)



NON COMMUTABLE

Commutable operations by examples (Data Structures)



Commutable operations by examples (TPC-C)

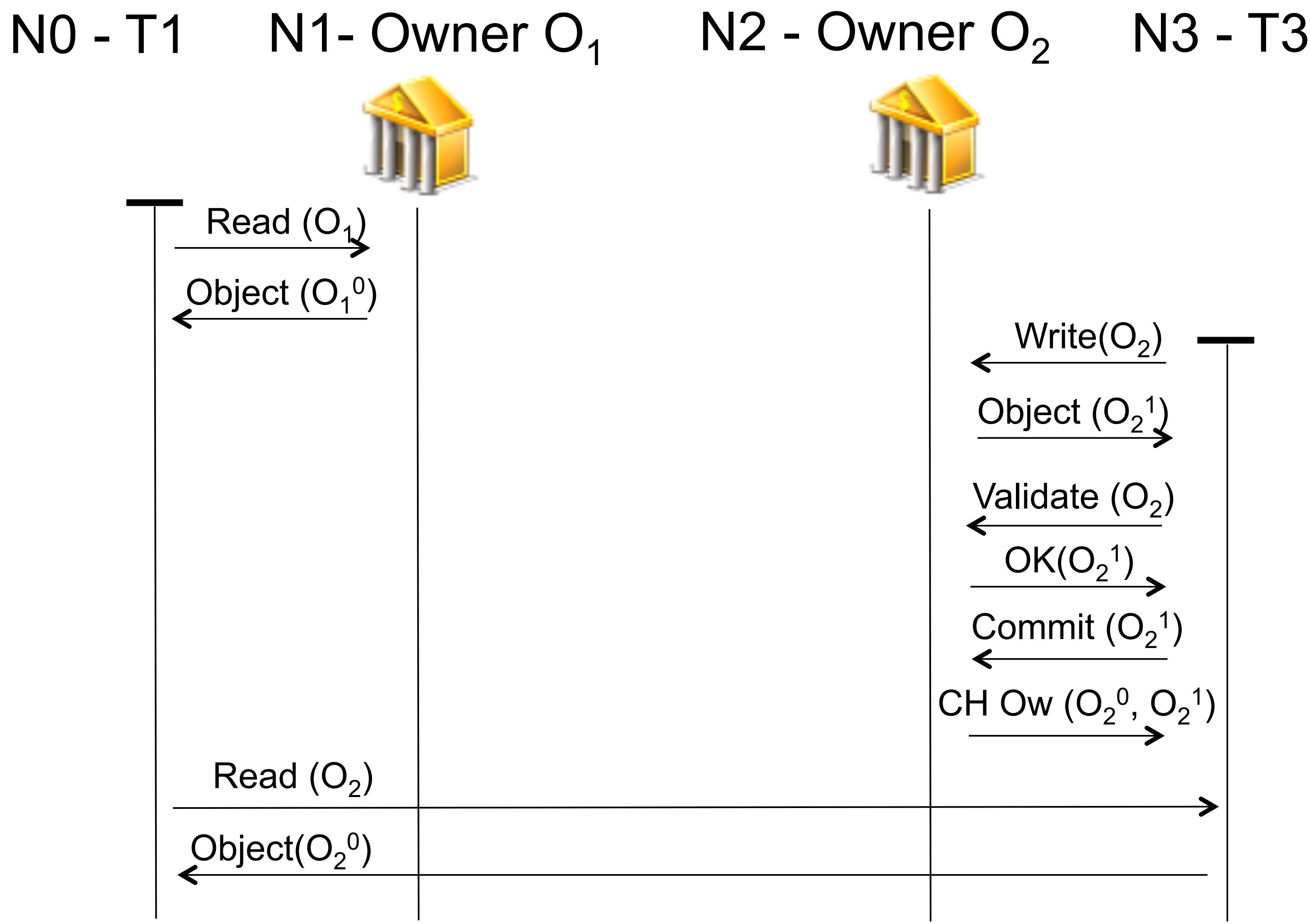
- New Order transactions:
  - Read:
    - Customer, District, Warehouse, Item, Stock
  - Write:
    - District, Stock
- Payment:
  - Read:
    - Warehouse, Customer, District
  - Write:
    - Warehouse, Customer, District

- New Order Transactions:
  - Write
    - district.D_NEXT_O_ID()
    - …
- Payment Transactions:
  - Write
    - district.D_YTD()
    - …

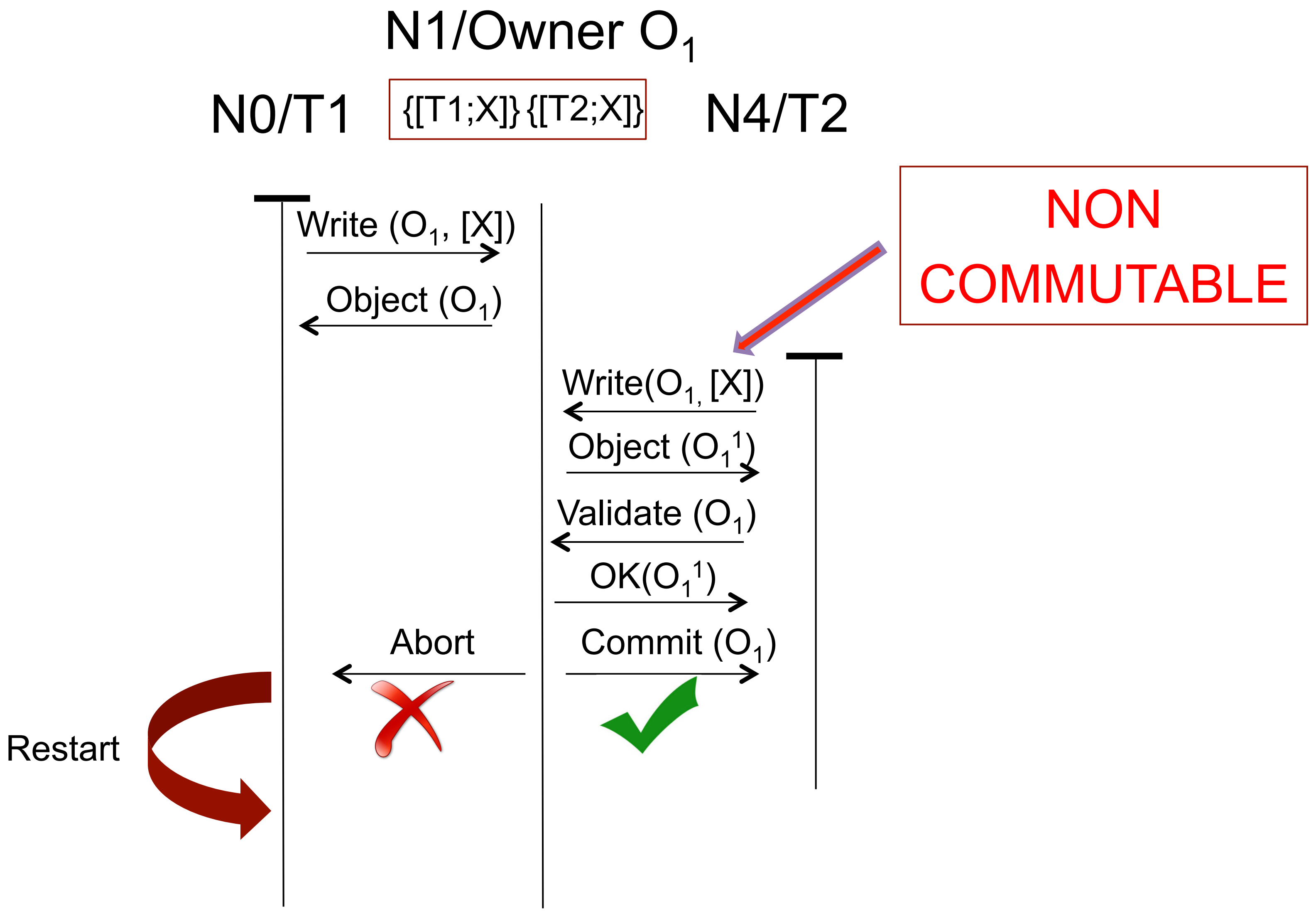
Paper's contribution

- MV-TFA, a multi-versioned version of TFA (Transactional Forwarding Algorithm) ensuring Snapshot Isolation
  - Read transactions don't abort
- CRF - Commutative Request First: a distributed transactional scheduler integrated with MV-TFA
  - CRF assumes definition of commutable rules by programmer;
  - Minimize abort rate
  - Increase concurrency
  - Increase performance
- Implementation and extensive experimental evaluation
  - Comparisons with state-of-the art DTMs
  - Experiments for the best tuning of the system

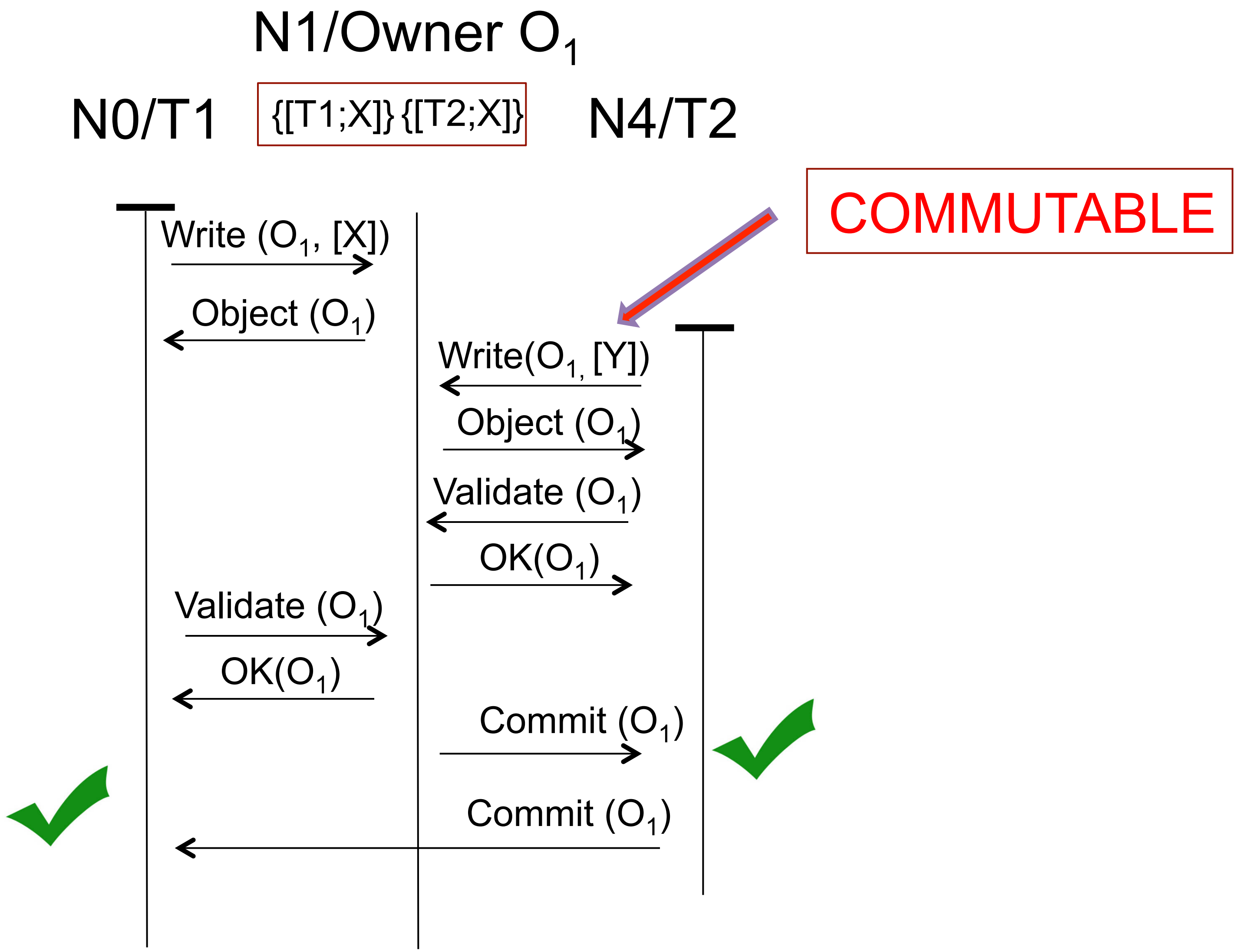
MV-TFA READ


N0 - T1
N1- Owner O1
N2 - Owner O2
N3 - T3
Read (O1)
Object (O1^0)
Write(O2)
Object (O2^1)
Validate (O2)
OK(O2^1)
Commit (O2^1)
CH Ow (O2^0, O2^1)
Read (O2)
Object(O2^0)

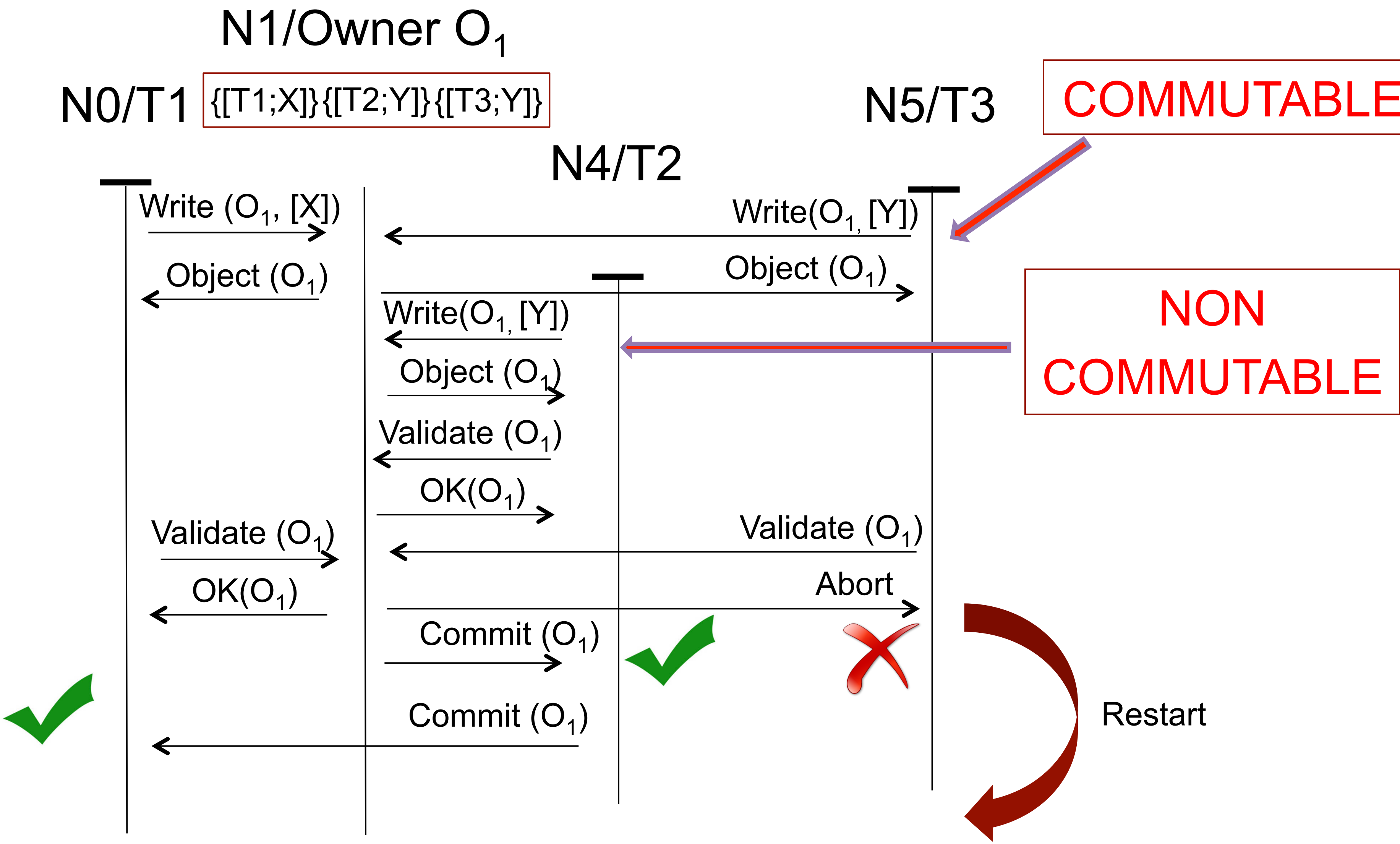
CRF – WRITE without concurrent validations



CRF – WRITE with concurrent validations

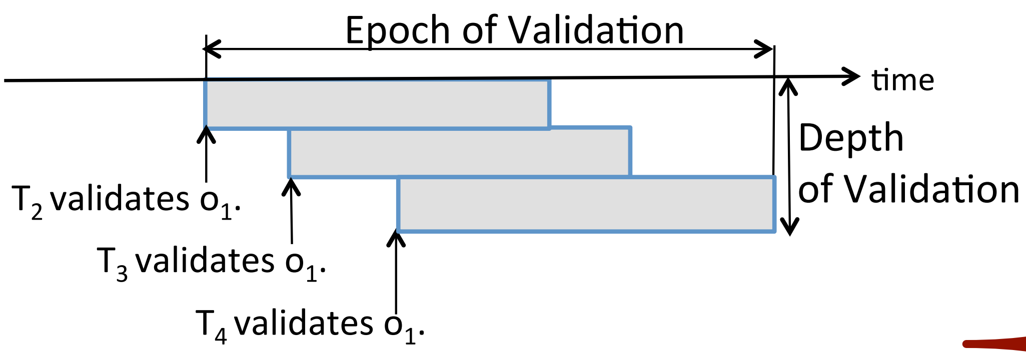


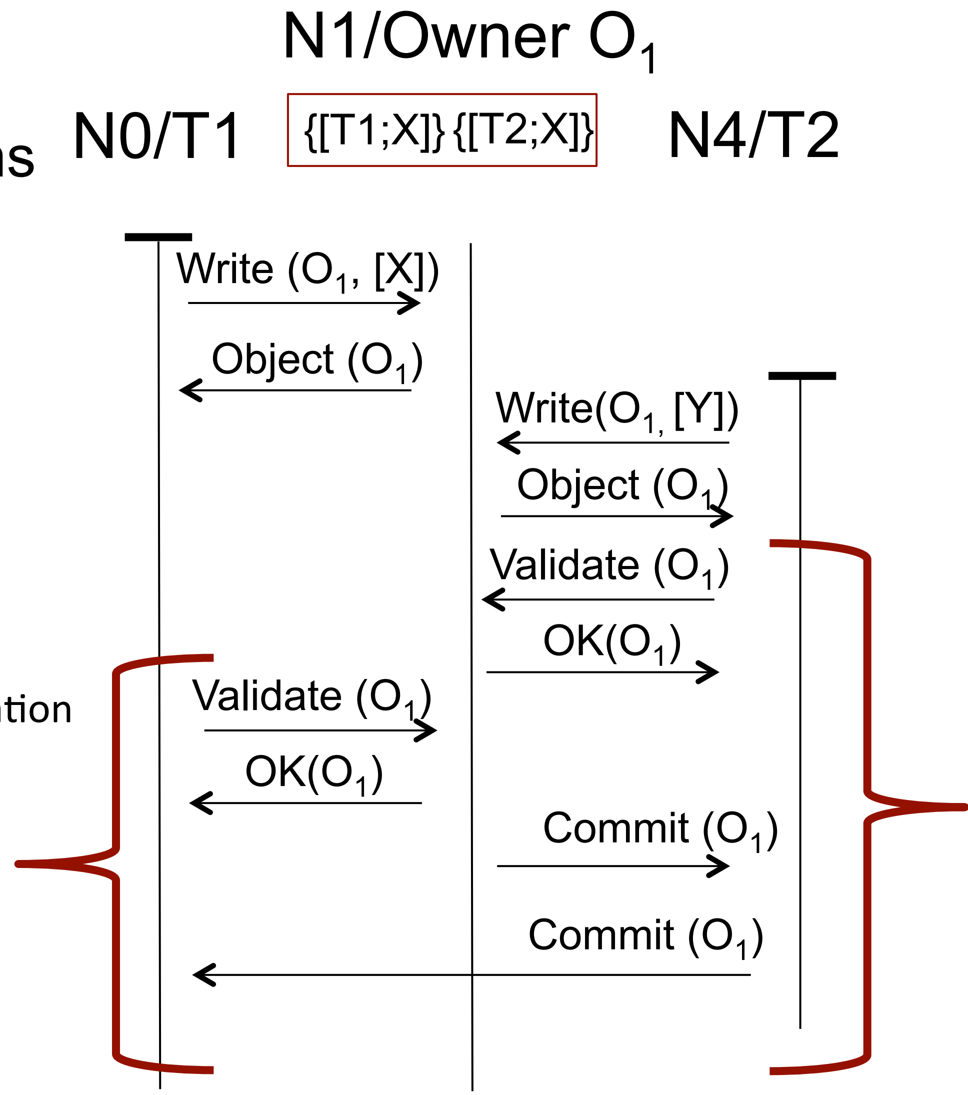
CRF – WRITE with concurrent validations



Depth of validation (MaxD)

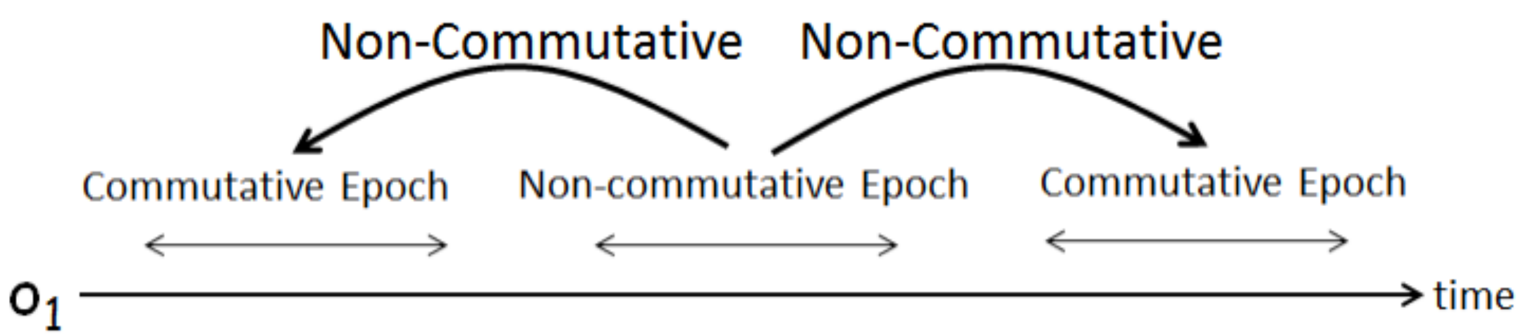
- Transaction commit phase is stretched depending on the concurrent validating transactions
- Depth of validation (MaxD): the number of transactions involved in the validation





Epochs

- CRF prioritizes commutable transactions for increasing concurrency. However adverse schedules can penalize non-commutable transactions
- CRF defines execution epochs:
  - In each epoch, commutative transactions concurrently participate in validation. In the next epoch, the non-commutative transactions stored in the scheduling queue restart and validate
  - Epoch shift is triggered when *MaxD* is reached or commitment process ends

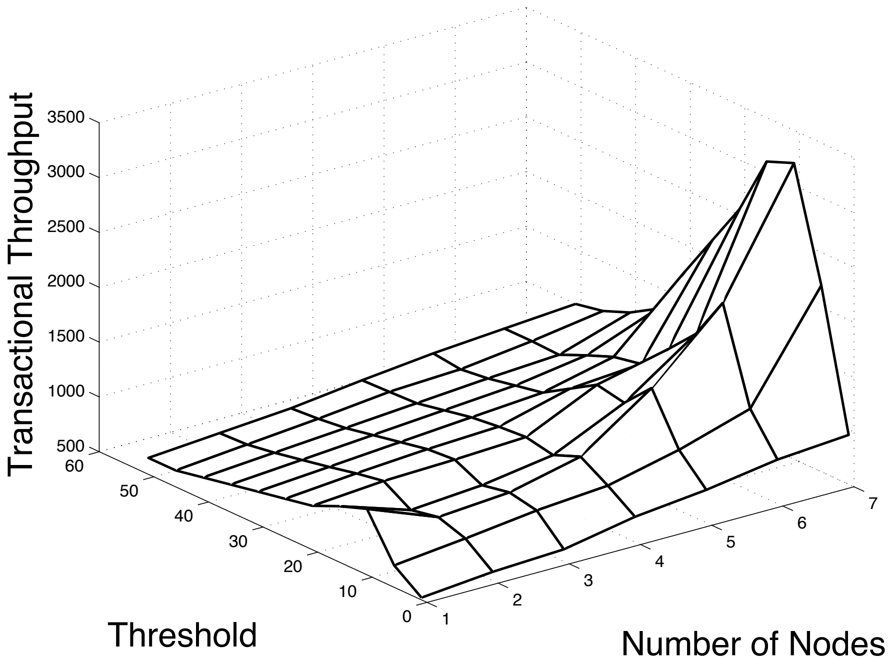


Experiments

- Test-bed:
  - Cluster of 10 nodes interconnected by a Gigabit connection
  - Each node equipped with 12 cores
  - 2 up to 120 concurrent threads in the system
- Competitors:
  - DecentSTM, MV-TFA (without scheduling)
- Benchmarks:
  - Micro Benchmarks:
    - Linked-List, Skip-list. Both implementation of Commutable Set
  - Macro Benchmarks:
    - TPC-C. Field-based commutativity

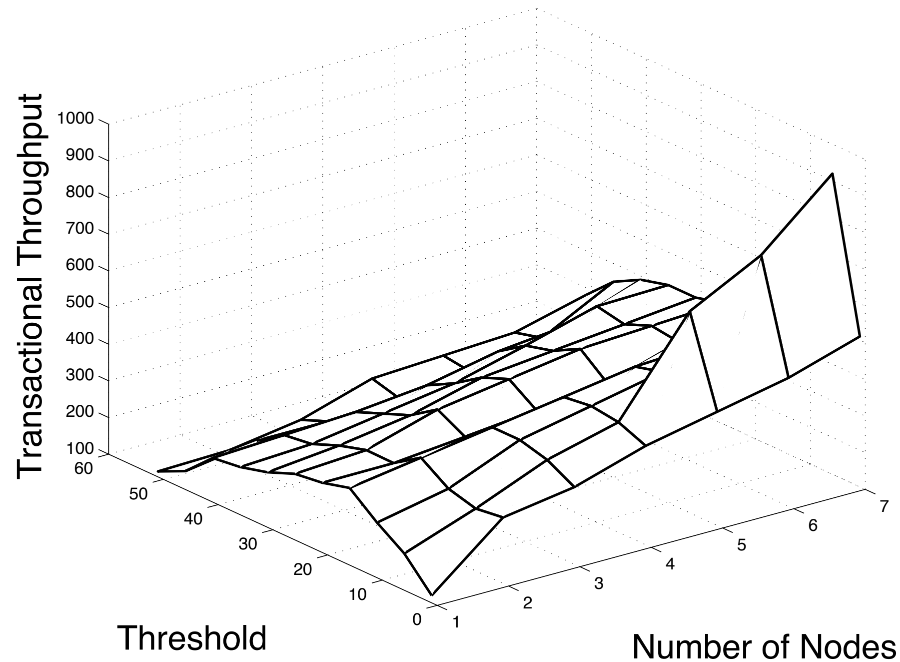
Finding depth of validation

- Run experiments measuring the performance of the system varying the depth of validation parameter



Linked List

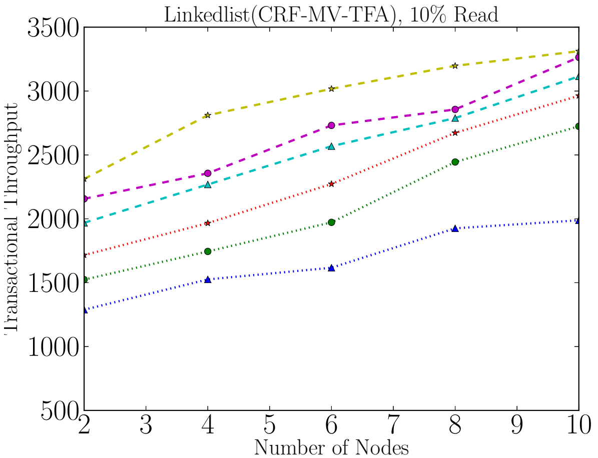
Max Depth = 10



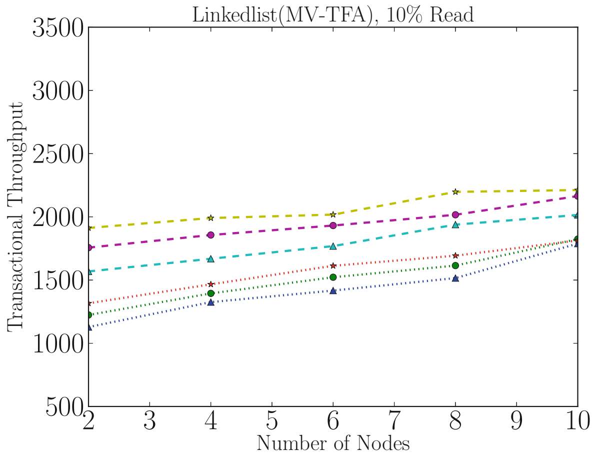
TPC-C

Max Depth = 5

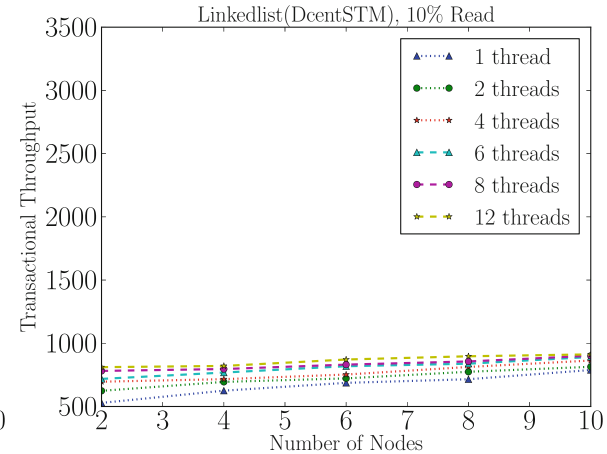
Linked List



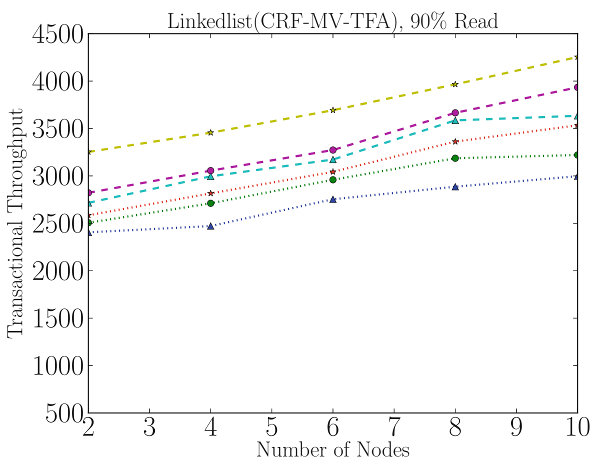
(a) CRF-MV-TFA, 10% Read



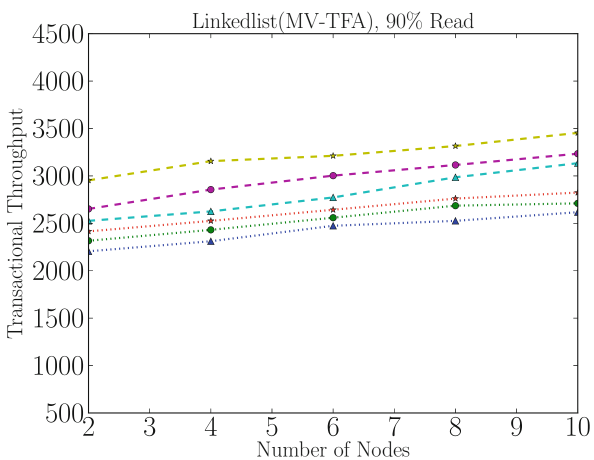
(b) MV-TFA, 10% Read



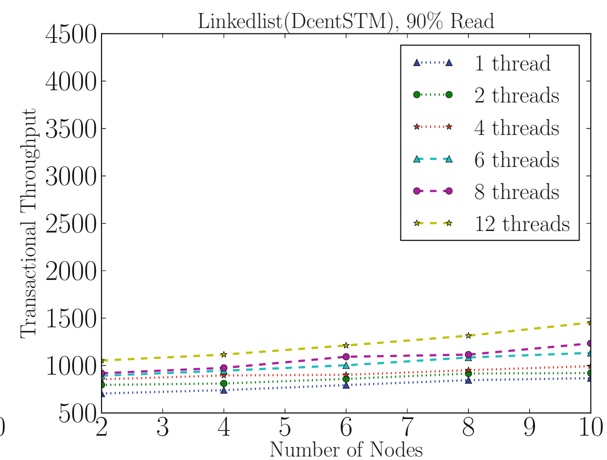
(c) DecentSTM, 10% Read



(d) CRF-MV-TFA, 90% Read

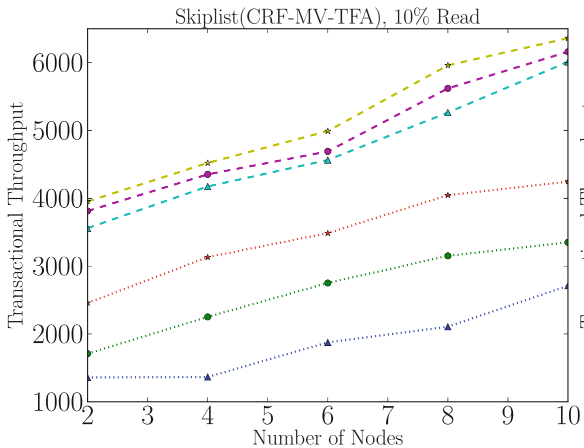


(e) MV-TFA, 90% Read

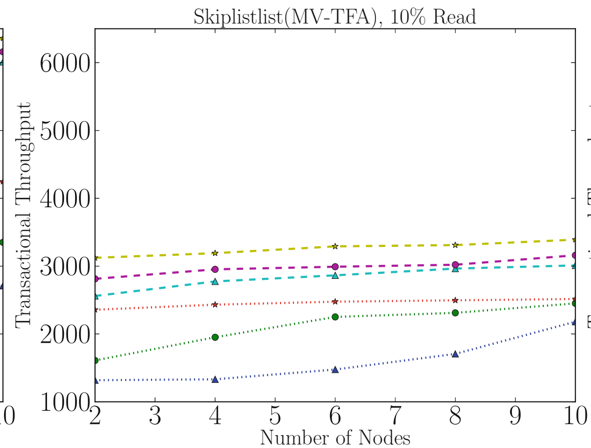


(f) DecentSTM, 90% Read

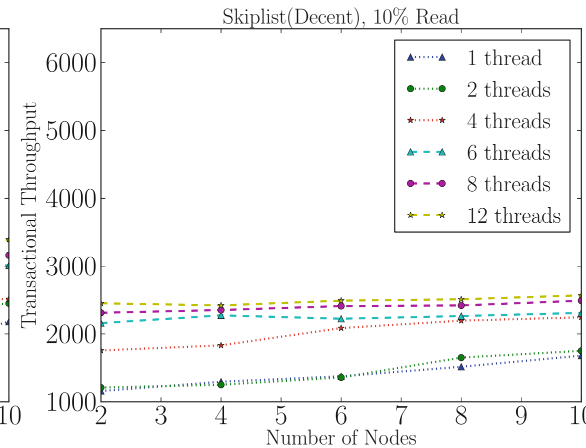
Skip List



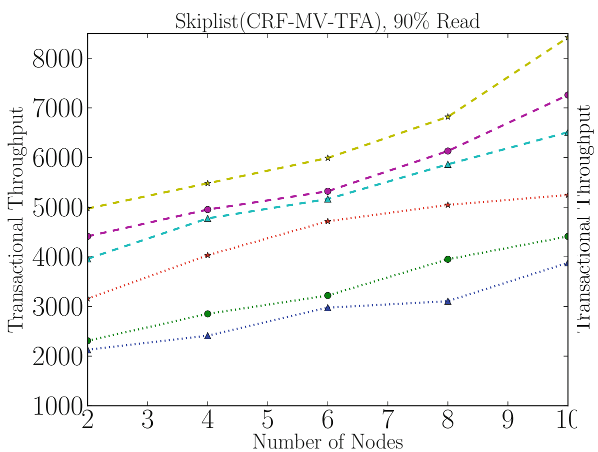
(a) CRF-MV-TFA, 10% Read



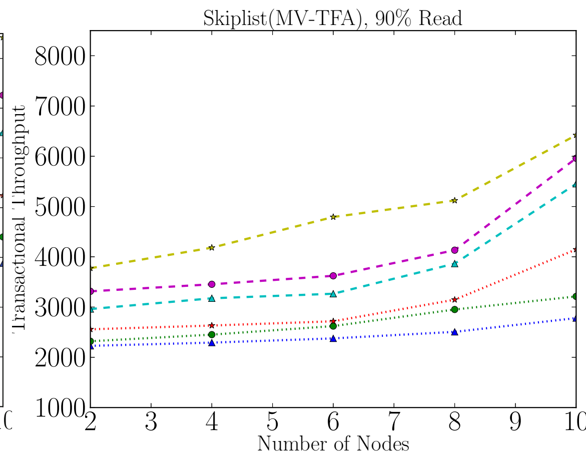
(b) MV-TFA, 10% Read



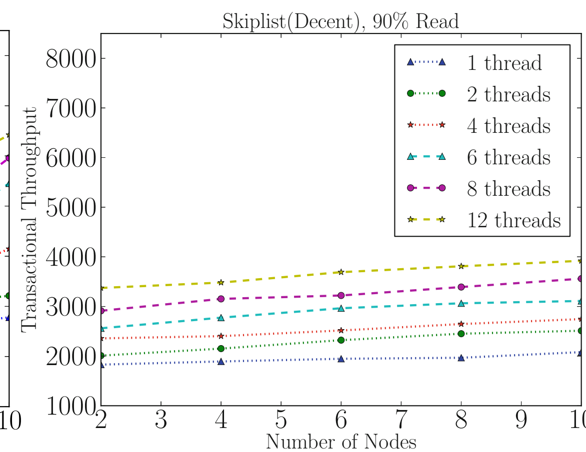
(c) DecentSTM, 10% Read



(d) CRF-MV-TFA, 90% Read



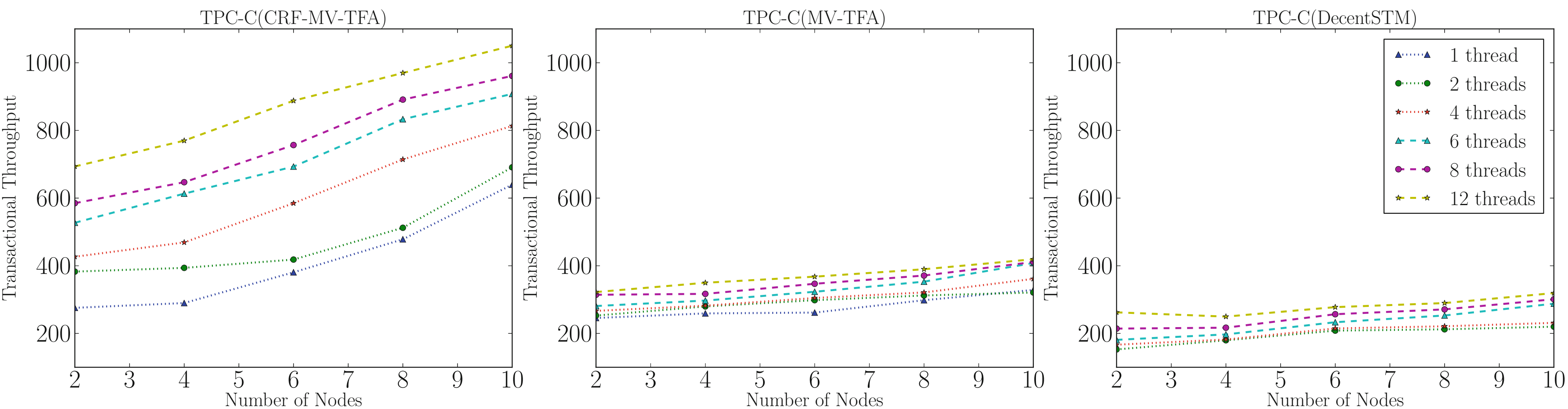
(e) MV-TFA, 90% Read



(f) DecentSTM, 90% Read

TPC-C

- % of transaction profiles as in the original specification
- # warehouse = 4 to increase the conflict probability



(a) CRF-MV-TFA (b) MV-TFA (c) DecentSTM

Thank you! Questions?

http://www.hyflow.org/

http://www.ssrg.ece.vt.edu/