On Partial Aborts and Reducing Validation Costs in Fault-tolerant Distributed Transactional Memory

Presented by Aditya Dhoke

09/04/2013

Committee Members:

Binoy Ravindran, *Co-Chair*

Eli Tilevich, *Co-Chair*

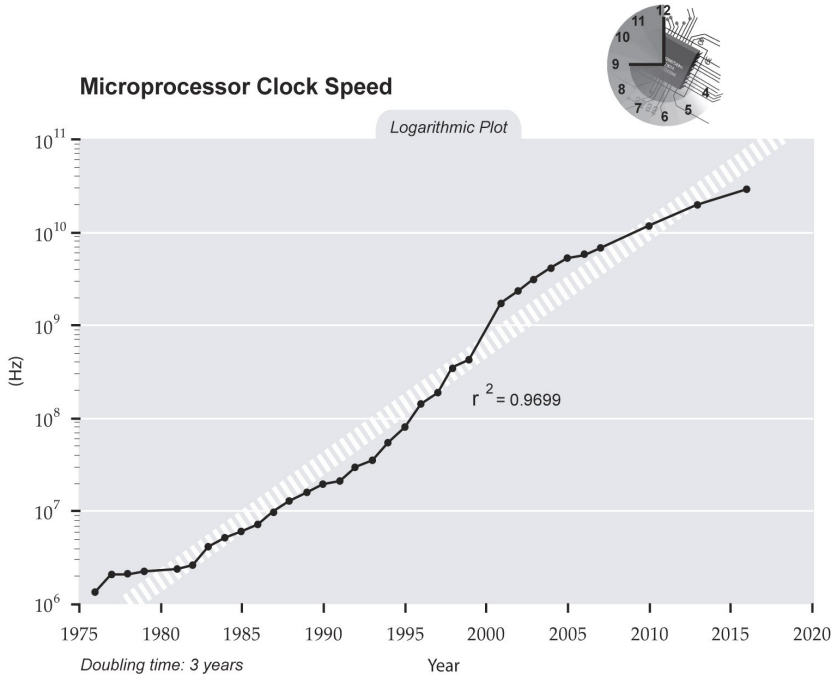
Wu-chun Feng

Thesis Contribution

- Implemented Java-based quorum replication framework, **QR-DTM**
- We present protocols for supporting partial aborts in fault-tolerant DTM, **QR-CN** and **QR-CHK**.
- **QR-ACN**, a framework for automating closed nesting in DTM
- We present three protocols for reducing validation cost in DTM **QR-ON**, **QR-OON**, and **QR-ER**.

Concurrency

- CPU clock speeds are increasing
- Speedup limited by sequential code
- Parallelize applications
- Hardware capability



Multiprocessor programming is tough!!!

Lock-based Concurrency Control

- Coarse grained locking
- Programming simple
- No concurrency
- Performance similar to serial execution

```
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();
  }
}
```

Lock-based Concurrency Control

- Fine grained locking
- Better parallelism
- Difficult to program
- Problems
  - Deadlocks
  - Livelocks
  - Priority inversion
  - Not Composable

```
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 (TM)

- Similar to database transactions
- Atomicity, Consistency, Isolation
- Easy to program
- Composability

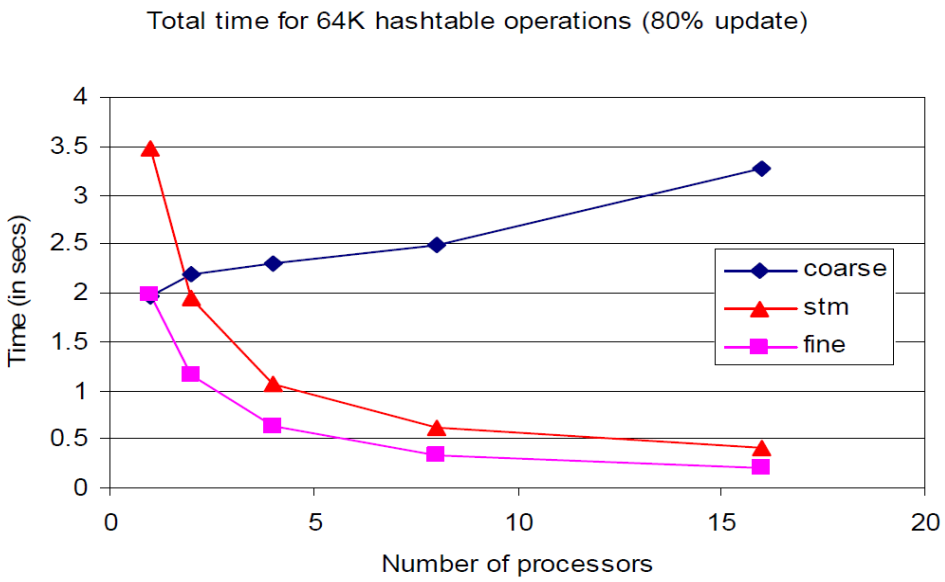
```
public boolean add(int item) {
  Node pred, curr;
  atomic {
    pred = head;
    curr = pred.next;
    while (curr.val < item) {
      pred = curr;
      curr = curr.next;
    }
    if (item == curr.val) {
      return false;
    } else {
      Node node = new Node(item);
      node.next = curr;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

How does TM work?

- Optimistic execution
- Transactions log changes to shared objects in read-set and write-set
- Validate objects to detect read/write & write/write conflicts
- Two transactions conflict, one of them is aborted, other is committed.
- Aborted transaction roll-back the changes and restarts

TM Performance

- Comparable to fine-grained locking



McRT-STM [61]

TM is Gaining Traction

- Hardware TM
  - Oracle, AMD and Intel have released hardware with HTM support
- Software TM
  - GCC - Language extension for STM support
  - Intel – C++ compiler with STM support
- Hybrid TM
  - STM + best-effort HTM

Distributed Transactional Memory (DTM)

- Extension of TM in distributed systems
- Classification based on system architecture :
  - Cache Coherent (cc) DTM – metric space communication
  - Cluster DTM – local and remote cluster
- Classification based on execution model :
  - Data Flow: Transactions immobile, objects migrate
  - Control Flow: Objects immobile, transactions invoke RPC

Distributed Transactional Memory (DTM)

- Durability by persistence in databases
- DTM has replication strategies
  - Partial Replication
  - Full Replication
- Synchronization among replicas
  - Atomic Broadcast – Non-scalable
  - Quorum-based replication uses Multicast

We consider cc DTM with full replication, quorum-based replication

Partial Transactional Abort

- Traditional TM's conservative approach (Flat nesting)
  - Conflict in later part, earlier part is conflict-free
  - Still rollback entire transaction !!!
  - Incur computation cost and remote calls

- Partially rollback till conflict-free and resume execution
- Suited for replicated systems, where operations are costly

Problem Definition

- What application workload will benefit from partial abort, as compared to flat nesting?
- What is the potential performance improvement or degradation due to partial abort?
- Which parameters of a transaction will affect partial abort's performance?
- How should the transaction code be transformed to obtain maximum benefits from partial abort?

In context of fault-tolerant DTM

Thesis Solutions: Partial Rollback

- Closed Nesting (QR-CN)
  - Transaction consists of multiple inner closed nested transactions
  - Inner transactions commit locally
  - Abort independently of outer transaction
- Checkpointing (QR-CHK)
  - Checkpoints created by saving transactional execution state
  - Partially rollback to resolve conflict and resume execution
- Automated Nesting (QR-ACN)
  - Dynamically determine contention
  - Compose closed nested transactions

Reducing Validation Costs

- False conflict
  - Independent high-level operations, conflict at low-level
  - High-level: Add element to set, Low-level: Add object to sorted list

- Performance degradation especially in fault-tolerant DTM
- Reduce validation cost approach to resolve false conflicts
  - Commit sub-transactions to expose partial changes
  - Selectively drop read-set objects

Problem Definition

- What is the performance improvement that can be obtained by reducing the validation cost?
- Which approach has the least performance degradation with increasing number of operations within a transaction?
- What applications are most suited for what validation cost reduction approaches?

Thesis Solution: Reducing Validation Costs

- Open Nesting (QR-ON)
  - Inner transactions commit globally
  - Objects released, not validated during commit
- Optimistic Open Nesting (QR-OON)
  - Commit phase cost, make non-blocking commit
  - Next transaction executes speculatively
- Early Release (QR-ER)
  - Release objects that do not affect transaction semantics
  - Suited for transactional data structures

Thesis Contribution

- Evaluation of **QR-CN** and **QR-CHK**. QR-CN improves throughput by 53% over flat nesting.

  "On Closed Nesting and Checkpointing in Fault-tolerant DTM", *IPDPS 2013*

- **QR-ACN**, an automated closed nesting framework, improves performance by 51% over flat nesting

  "*Automated Closed Nested Transactions in DTM"* (To be submitted in CGO 2014)

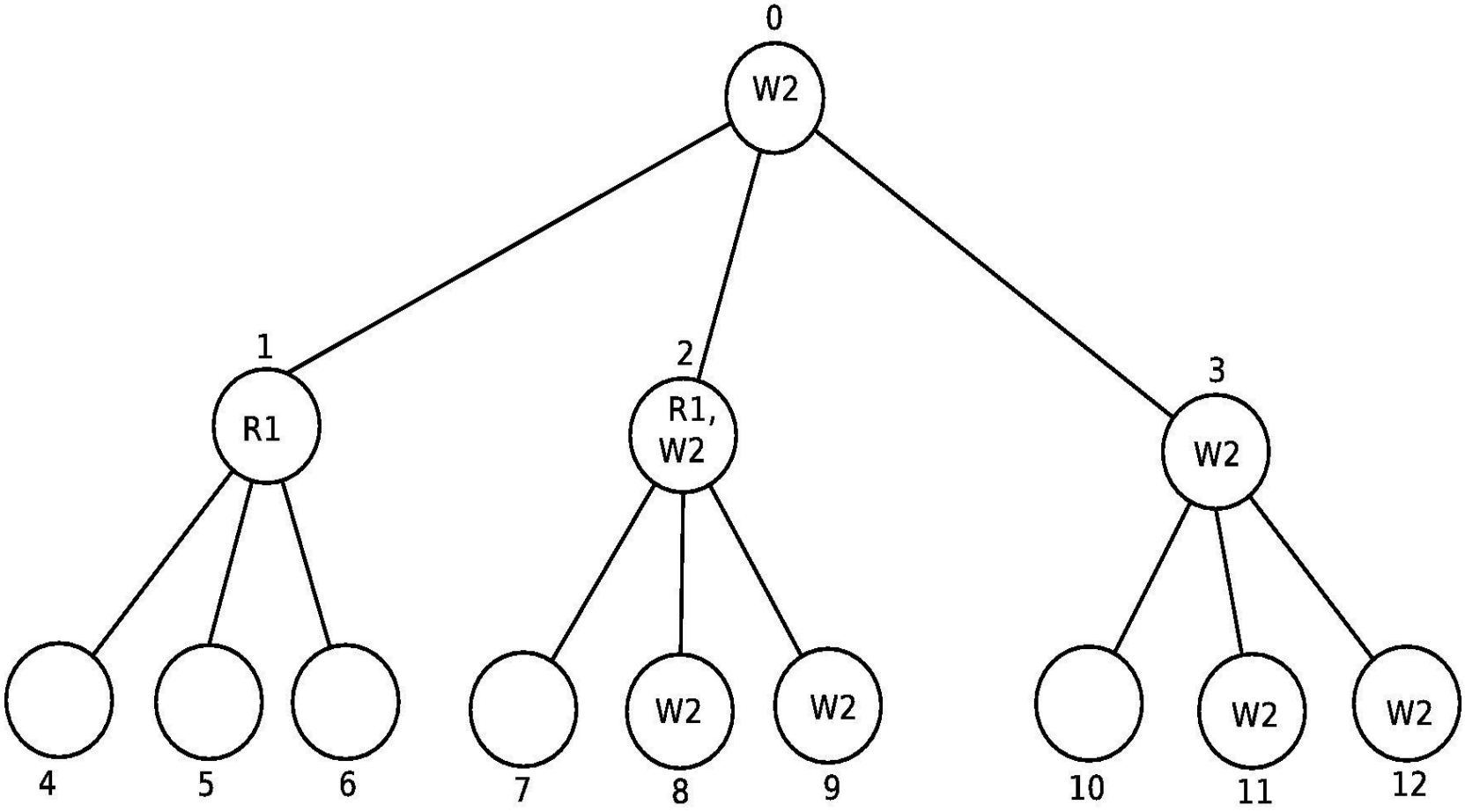
- Evaluation of **QR-ON**, **QR-OON**, and **QR-ER** show QR-ER outperforms QR-ON and QR-OON by up to 10x

  "*On Reducing Validation Costs in DTM"* (To be submitted in IPDPS 2014)

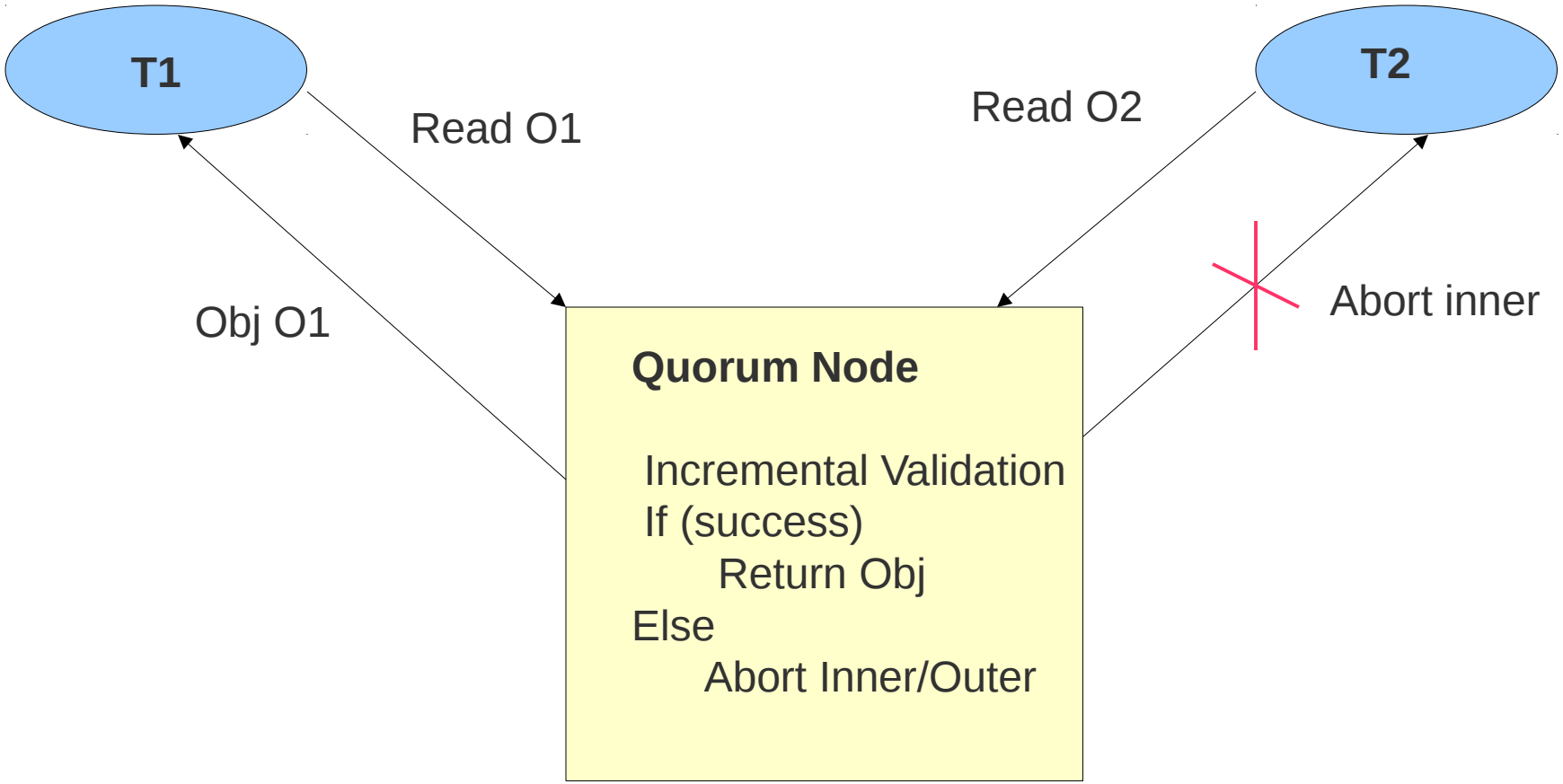
Quroum-based Replication (QR-DTM)

- Logical Ternary Tree
- Read quorum : Majority at a level ---> read/write requests
- Write quorum : Majority at all levels ---> commit requests
- Read and write quorum always intersect

Quorum Nodes in QR-DTM



QR-CN: Closed Nesting in QR-DTM



QR-CN: Commit Operation

- Inner transaction commit :
  - Merge read and write set with outer transaction
  - Incremental validation ensures that data-set is valid at commit time

- Outer transaction commit:
  - Commit using write quorum

QR-CHK: Checkpointing in QR-DTM

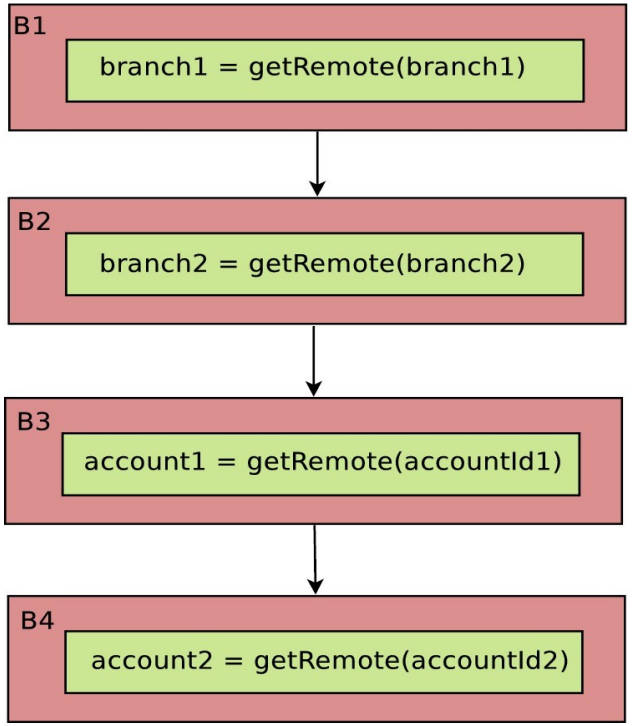
- Transaction (client node) creates checkpoint locally for every read
- Remote node :
  - Validates the data-set
  - Records the checkpoint ID for each read
  - On conflict
  - Finds checkpoint ID that has all its objects valid
  - Transaction rolls back to ID and resumes

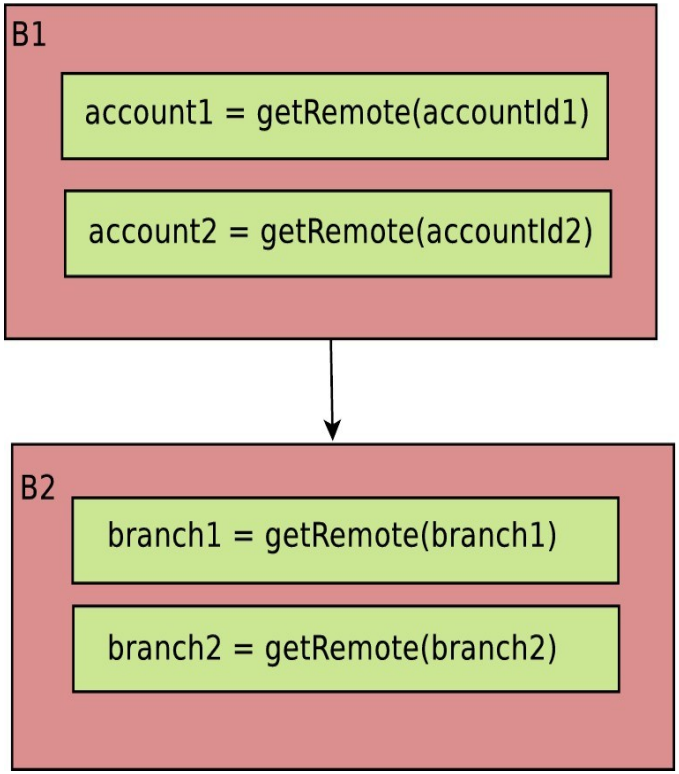
QR-ACN: Automated Closed Nesting in QR-DTM

- Easy programmability in TM
- Performance Improvement from Closed Nesting
- Automation can achieve both!

- Closed nesting effective when transactions access high contention objects later in execution
- Determine the contention of objects
- Move high contention objects towards commit

QR-ACN: Code for Bank Transaction

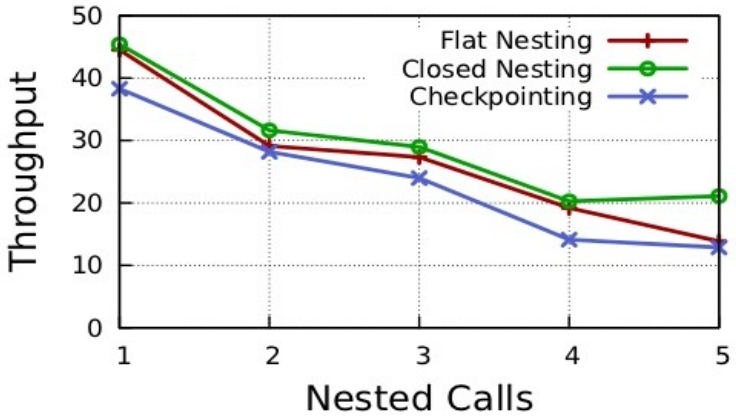


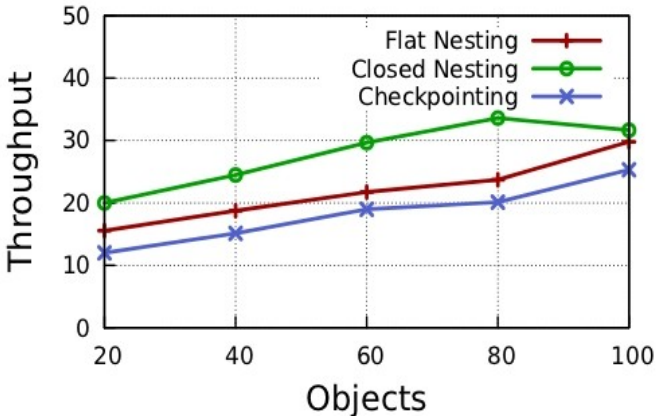


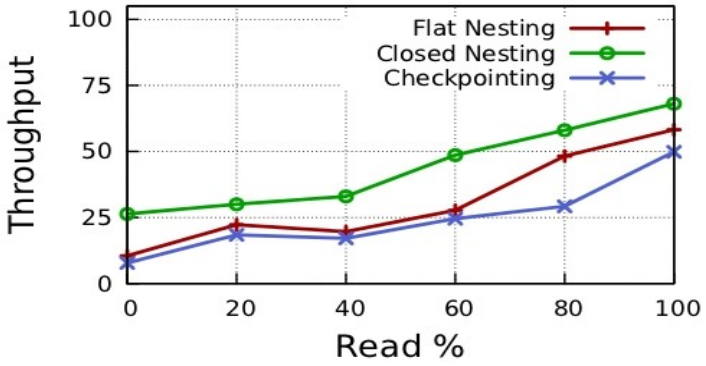
Experimental Evaluation

- Benchmarks
  - Bank, Hashmap, RBTree, SkipList, Vacation (STAMP), TPC-C
- Experimental Setup
  - Each node is running AMD Opteron processor on Linux 10.04
  - Each node assigned same read and write quorum
  - Testbed consisted of 40 quorum nodes
  - Up to 30 clients

Evaluation of Partial Abort Protocols

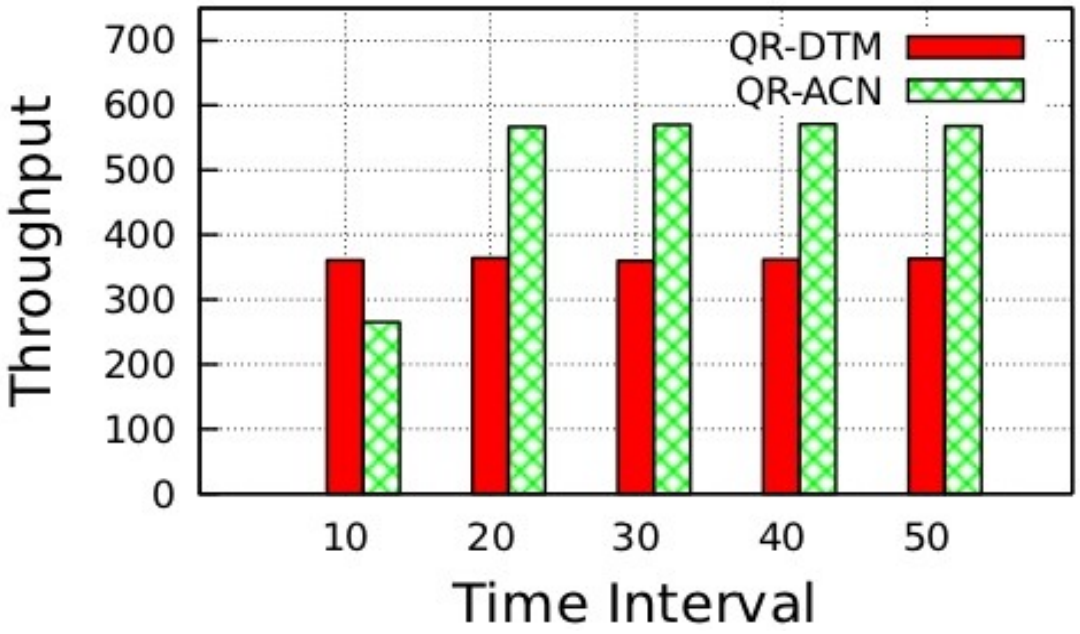






Bank Benchmark

TPC-C: QR-ACN versus QR-DTM



% Throughput Improvement for Payment

Conclusion: Partial Abort

- Closed nesting best applies for applications with high contention
- Performance of closed nesting increases with increase in the level of contention and transaction length
- Automated closed nesting is best suited for applications where workload changes during run-time
- Checkpointing has performance degradation

QR-ON: Open Nesting in QR-DTM

- Client Node
  - Acquire abstract lock to protect change
  - Commit inner transaction globally
  - On abort, compensation for already committed transactions
- Remote Node
  - Manage abstract locks

QR-OON: Optimistic Open Nesting in QR-DTM

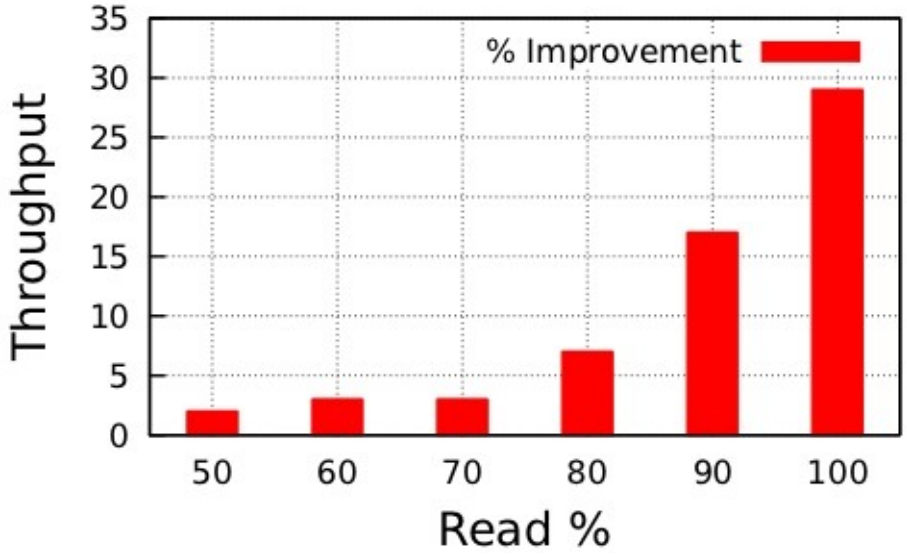
- Client Node
  - Current inner transaction commits asynchronously
  - Next inner transaction reads speculatively
  - If current commits, next continues its execution
  - If current aborts, abort next too and restart current

- Remote Node
  - Same as QR-ON

QR-ER: Early Release in QR-DTM

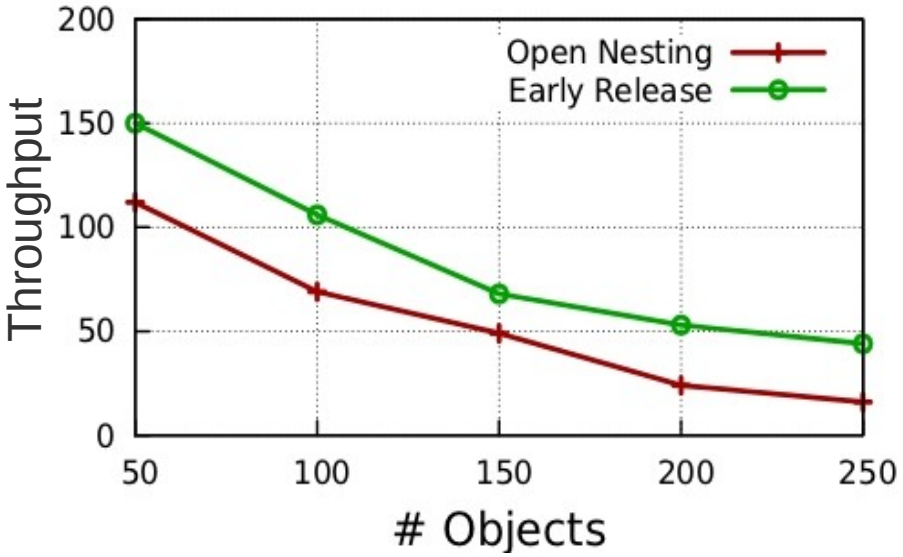
- Local Node
  - Release objects from read-set which will not affect transaction semantics
  - For these objects set flat *validate* to false
  - Validate request only consists of *validate* objects
- Remote Node
  - Same as QR-DTM

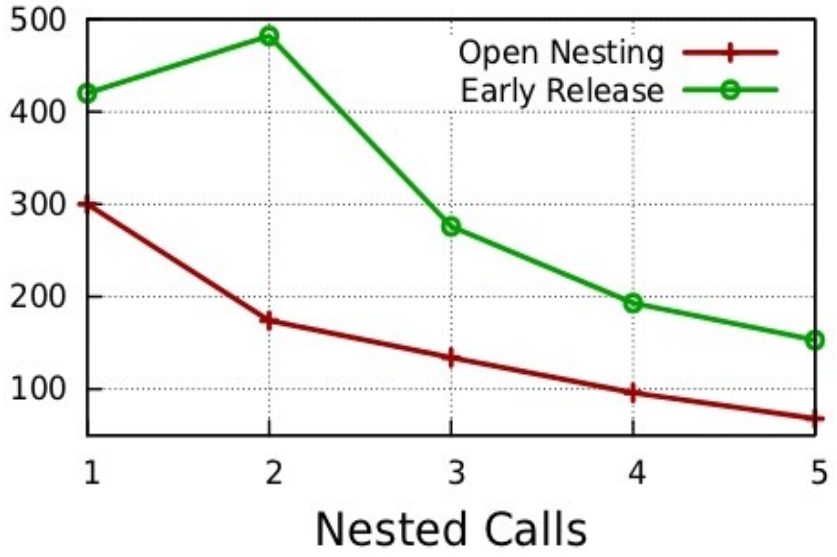
QR-OON vs QR-ON



Hashmap: % Throughput Improvement over QR-ON

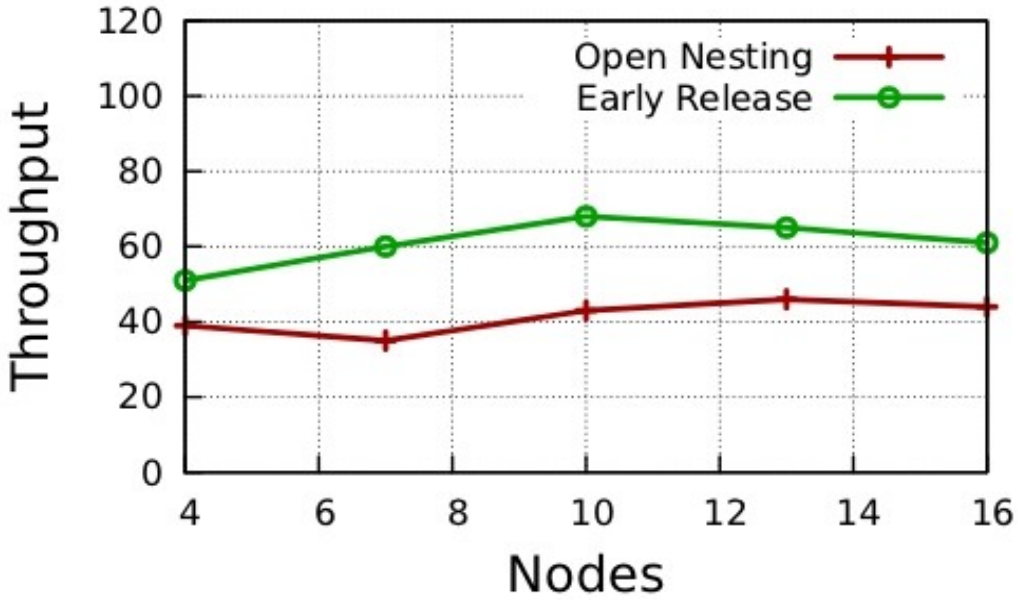
QR-ER vs QR-ON





Hashmap: Variation with #Object and Nested Calls

QR-ER vs QR-ON



TPC-C: Variation with Nodes

Conclusion: Reduce Validation Costs

- Open nesting has significant commit overhead
- Optimistic open nesting can outperform open nesting in low contention scenarios
- Early release can provide improvement – up to an order of magnitude – over its open nesting counter-parts

Thank you! Questions?