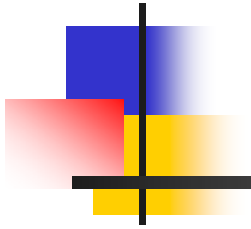
Large-scale Experimentation with Preventive Replication in a Database Cluster



Patrick Valduriez, Esther Pacitti, Cédric Coulon

Atlas group, INRIA and LINA,

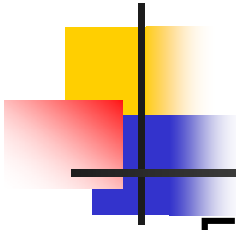
University of Nantes

ACI MDP2P 2003-2006

http://www.sciences.univ-nantes.fr/lina/ATLAS/MDP2P/

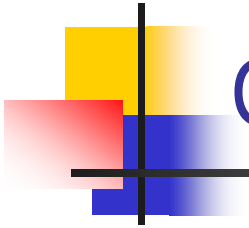
Outline

- Context and architecture
- Preventive Replication
- Replication Manager Architecture
- Optimizations
- RepDB* Prototype
- Experiments
- Conclusion



Related work

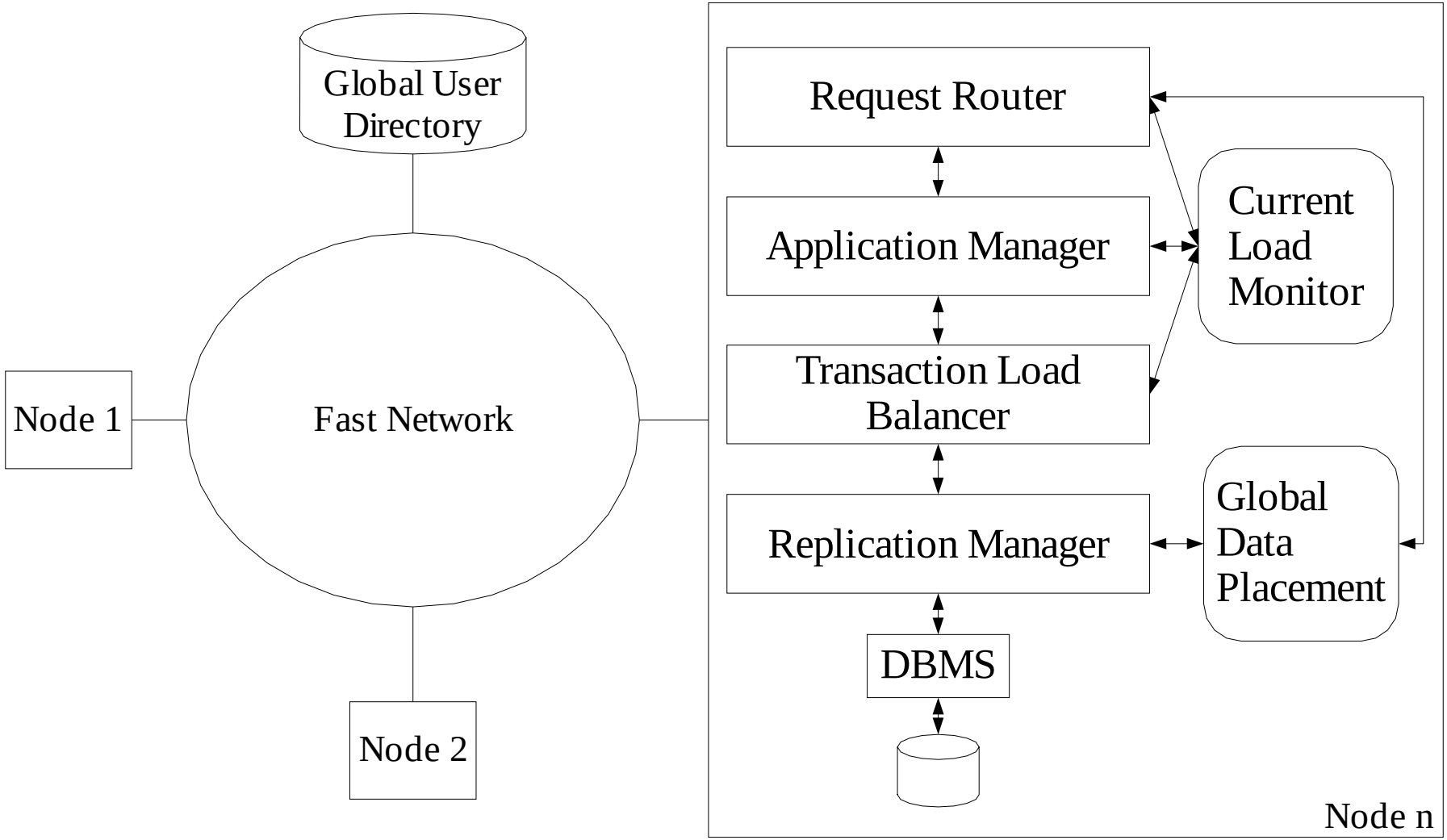
- Eager replication in DB clusters
  - Kemme and Alonso [VLDB00, TODS00]
  - Pedonne and Schiper [DISC98]
- Lazy replication in dist. systems
  - Pacitti et al. [VLDB99, VLDBJ00, DAPD01]
- Replication in middleware
  - Amza et al. [Middleware03]
  - Cecchet [OPODIS03]
  - Gançarski et al. [CoopIS02, Info. Syst.05]
  - Jiménez-Peris et al. [ICDCS02, Middleware04]
- Preventive replication
  - Pacitti et al. [Europar03, Vecpar04, BDA05, IEEE ICPADS05, VLDB-WDDIR05, DAPD05]

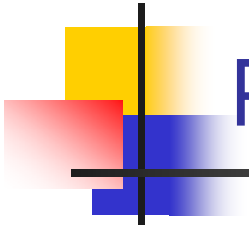


Context

- Update-intensive applications
  - e.g. Application Service Providers
- Database cluster with shared-nothing architecture and fast, reliable network
- Data replication to improve availability and performance

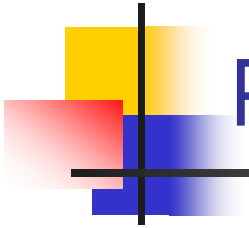
DB cluster architecture





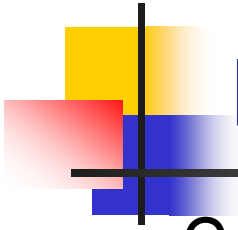
Preventive replication - objectives

- Multi-master replication
  - Each node can update replicas
- Database autonomy
  - Support black-box DBMS
- Strong replica consistency
  - Equiv. to ROWA (read one – write all)
- Non-blocking
  - Unlike 2PC
- Performance
  - Scale-up and speed-up



Preventive Replication - assumptions

- Network interface provides FIFO reliable multicast
- *Max* is the upper bound of the time needed to multicast a message
- Clocks are ε-synchronized
- Each transaction has a chronological timestamp *C* (its arrival time)



Preventive replication - consistency

- Consistency Criteria = total order
  - Transactions are received in the same order at all nodes involved: this corresponds to the execution order
- To enforce total order, transactions are chronologically ordered at each node using their *delivery time*

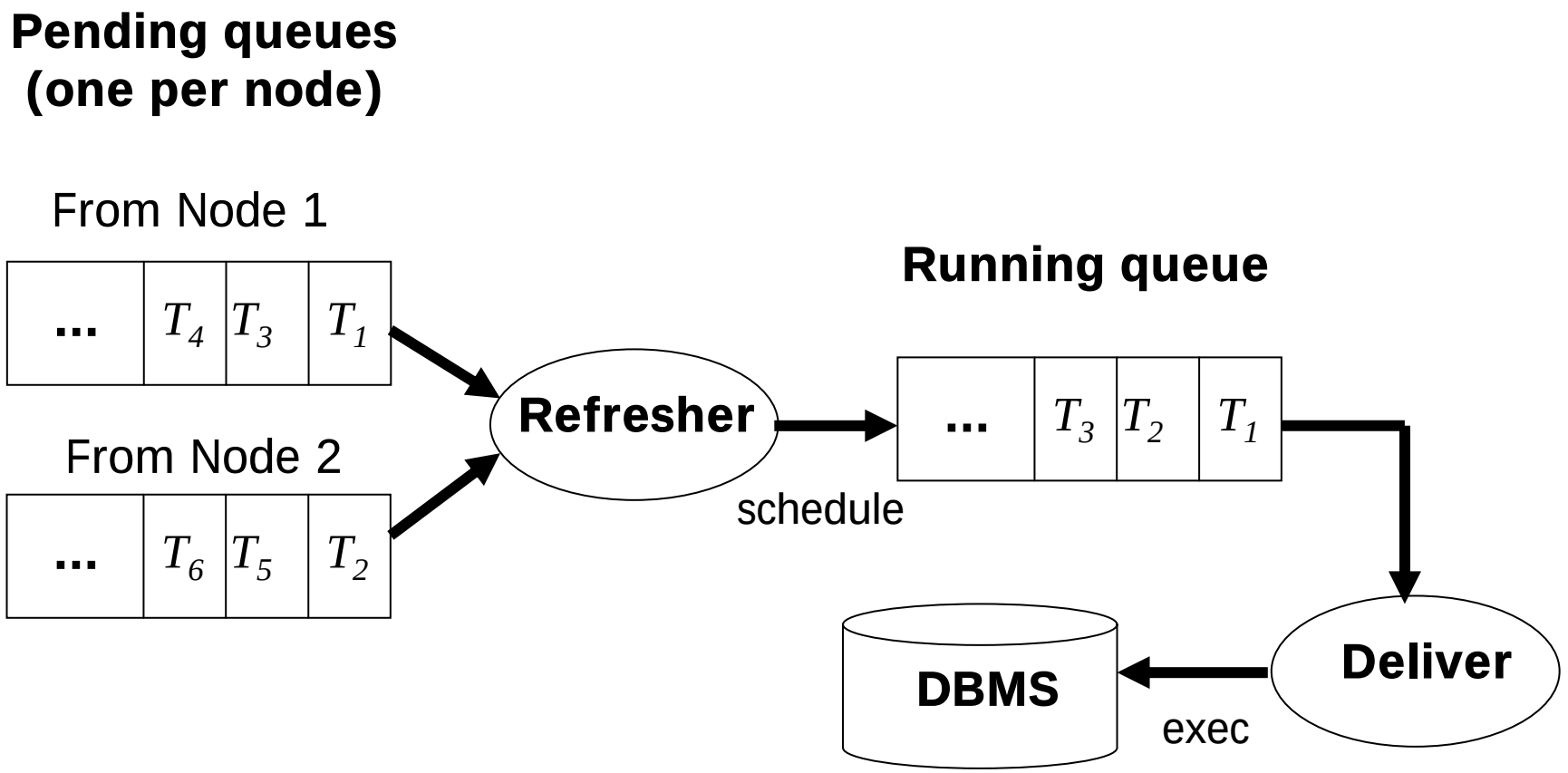
$$\boldsymbol{delivery\_time = C + Max + \varepsilon}$$

- After $Max+\varepsilon$, all transactions that may have committed before C are supposed to be received and executed before T

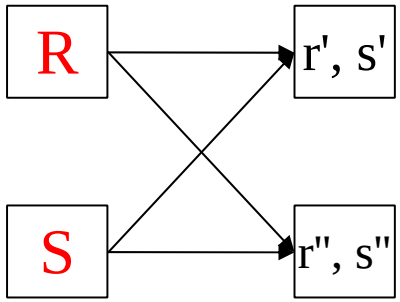
Preventive replication - principle

- Propagation
  - When a node *i* receives a trans. *T*, it multicasts it to all nodes including itself
    - Each node puts *T* in pending queue q_i in FIFO order
- Scheduling
  - At each node, all pending transactions are ordered wrt delivery times
    - After expiration of its delivery time, *T* is put in running queue
- Execution
  - Transactions in running queue are submitted to the DBMS for execution

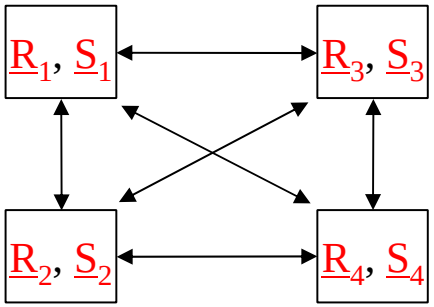
Queues management



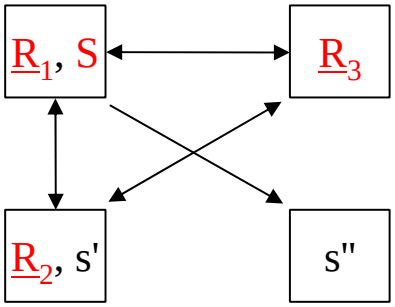
Preventive replication - configurations



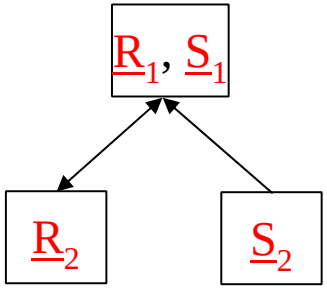
- Bowtie



- Fully replicated



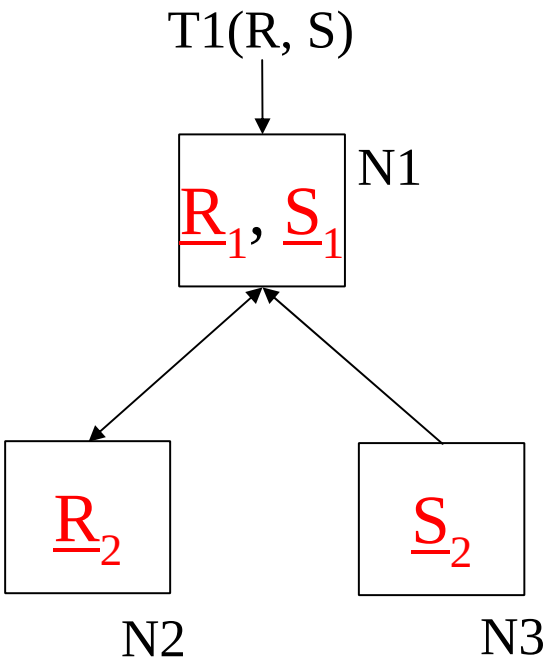
- Partially replicated



- Partially replicated

- PRIMARY copies (R): Can be updated only on master node
- Secondary copies (r): read-only
- MULTIMASTER copies (R_1): Can be updated on more than one node

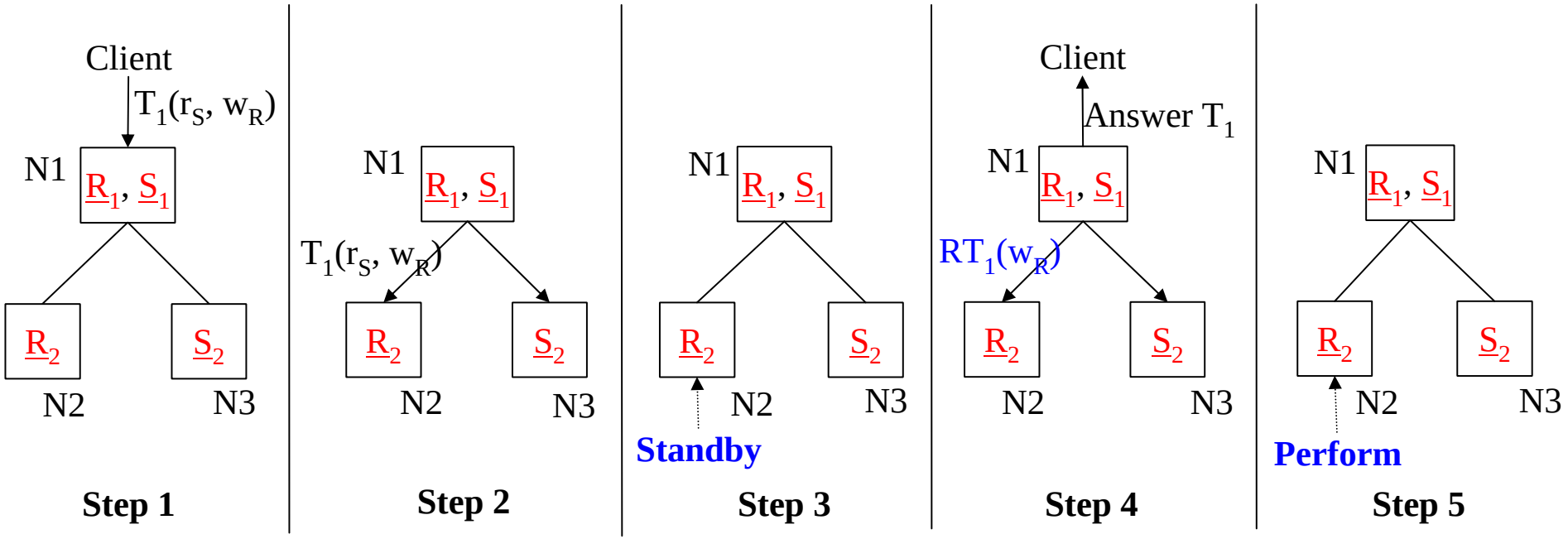
Partially replication - problem



- Some transactions cannot be executed

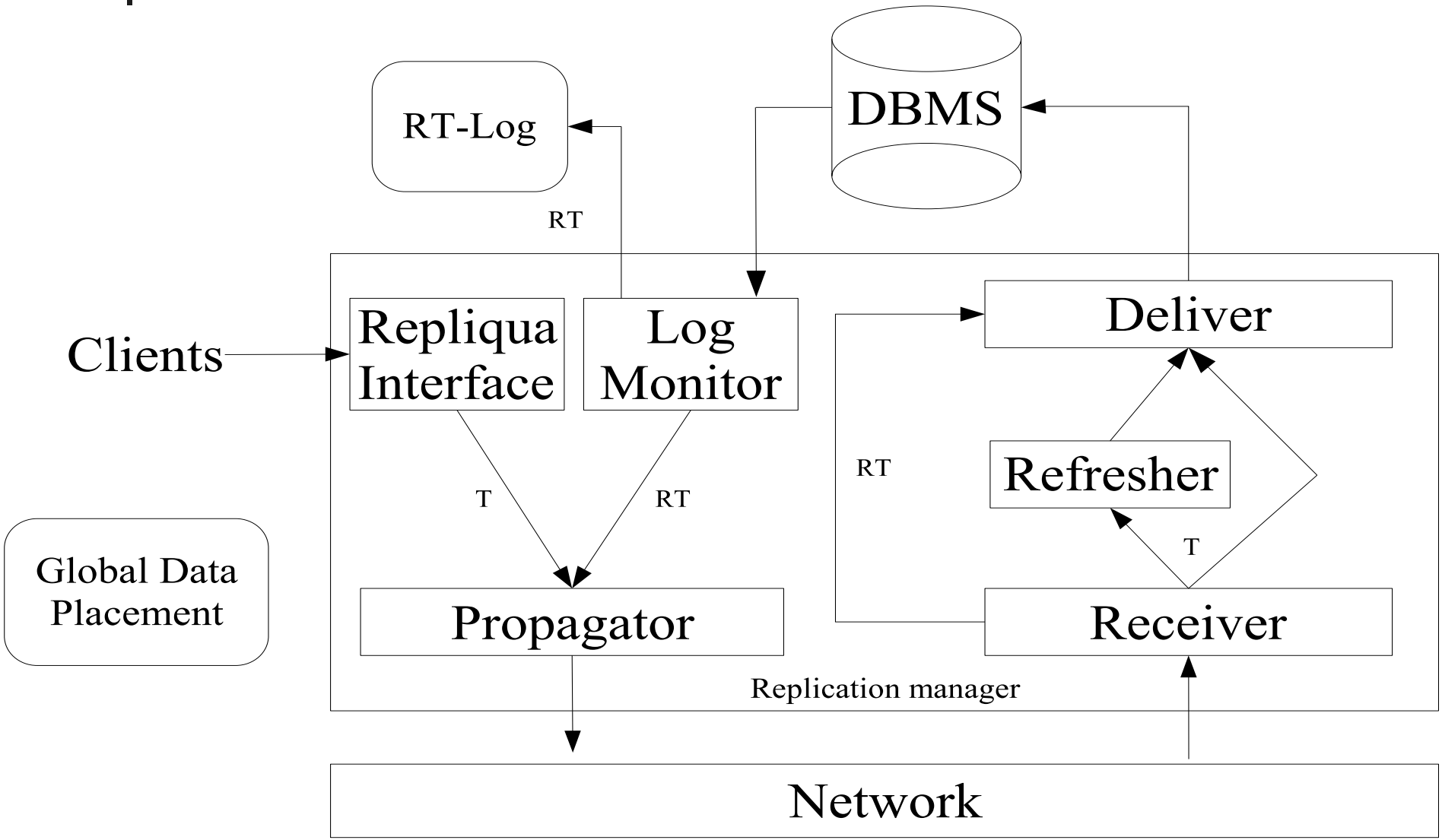
Example@N2
UPDATE R SET c1 WHERE c2 IN
(SELECT c3 FROM S);

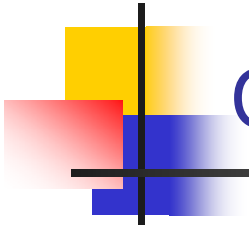
Partially Replicated Configurations (2)



- On the target nodes, T_1 waits (Step 3)
- After execution on the origin node, a *Refresh Transaction* (RT_1) is multicast to target nodes (Step 4)
- RT_1 is executed to update replicated data

Replication Manager Architecture





Optimization: eliminating delay times (1)

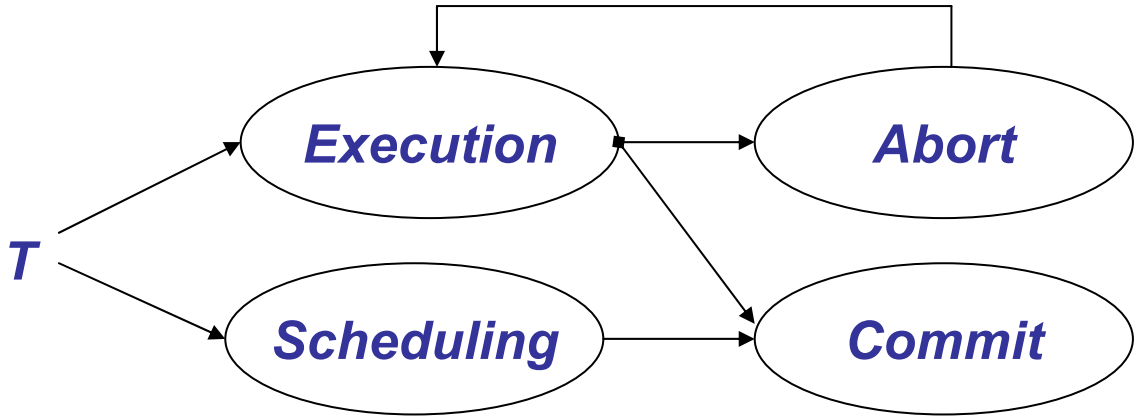
- In a cluster network, messages are naturally totally ordered [Pedonne & Schiper, 1998]
- We can parallelize transaction scheduling and execution
  - Submit a transaction to execution as soon as it is received
  - Schedule the commit order of the transactions
    - A transaction can be committed only after *Max + ε*
  - If a transaction is received out of order, abort and re-execute all younger transactions
- Yields concurrent execution of non conflicting transactions

Optimization: Eliminating delay times (2)

Basic Preventive replication

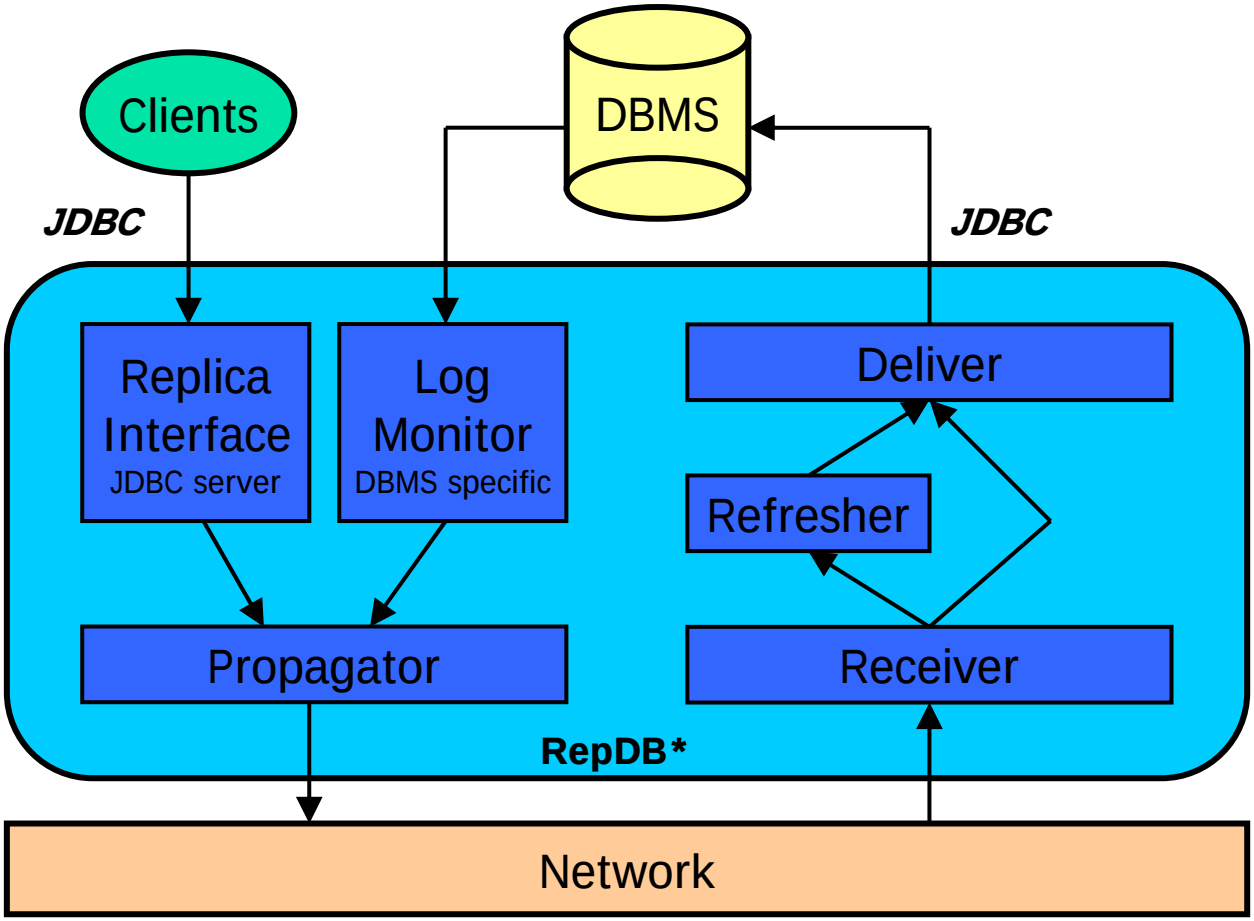
T's refreshment time = ***Max*** **+ ε + t**

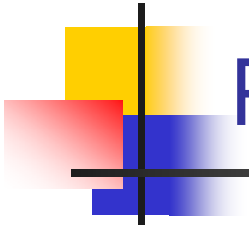
Optimized Preventive Replication



T's refreshment time = *maximum* (***Max*** **+ ε, t)**

RepDB* Prototype: Architecture

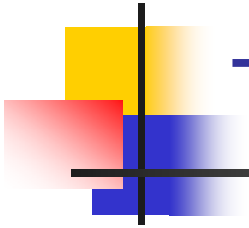




RepDB* Prototype: Implementation

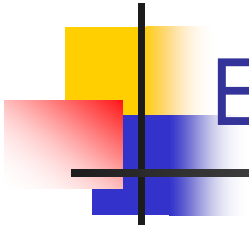
- Open Source Software under GPL
  - 200 downloads after one month of release
  - Java (10K+ lines)
  - DBMS is a black-box
    - PostgreSQL, Oracle
  - JDBC interface (RMI-JDBC)
  - Uses Spread toolkit (Center for Networking and Distributed Systems - CNDS) to manage the network
- Simulation version in SimJava

http://www.sciences.univ-nantes.fr/ATLAS/RepDB



TPC-C benchmark

- 1 / 5 / 10 Warehouses
- 10 clients per Warehouse
- Transactions' arrival rate is 1*s* / 200*ms* / 100*ms*
- Load = combination of
  - Update transactions
    - New-order: high frequency (45%)
    - Payment: high frequency (45%)
  - Read-only transactions
    - Order-status: low frequency (5%)
    - Stock-level: low frequency (5%)

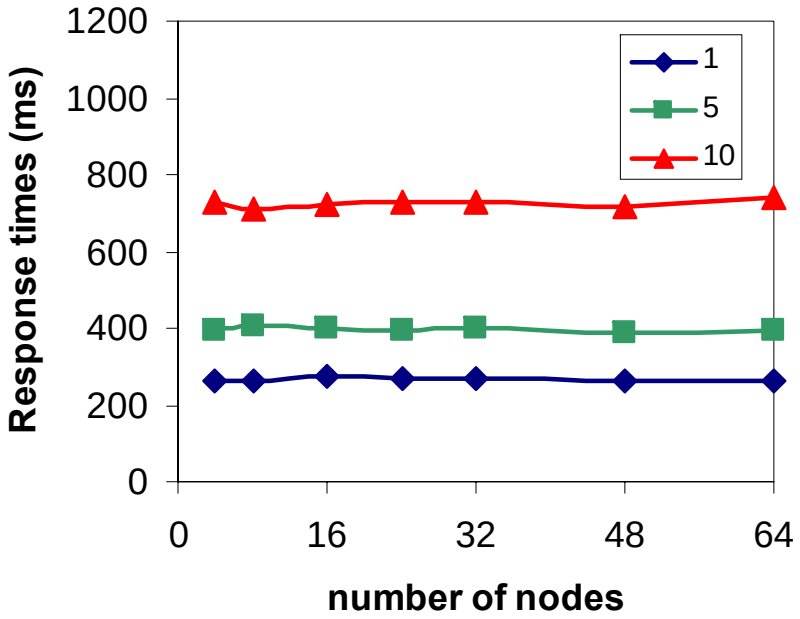


Experiments

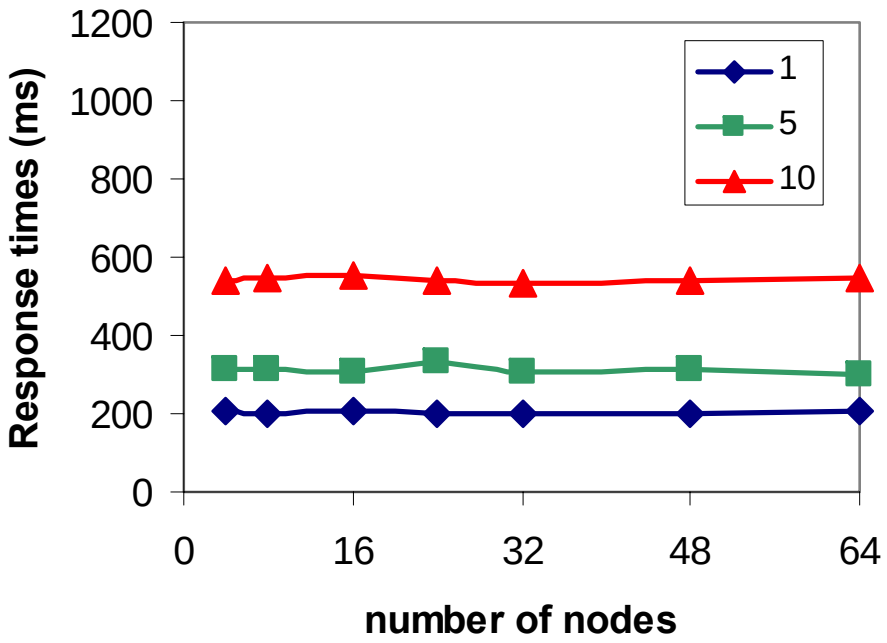
- Cluster of 64 nodes (Paris@irisa.fr)
  - PostgreSQL 7.3.2 on Linux
  - 1 Gb/s network
- 2 Configurations
  - *Fully Replicated* (*FR*)
  - *Partially Replicated* (*PR*): each type of TPC-C transaction runs using ¼ of the nodes

Scale up

Response time of update transactions



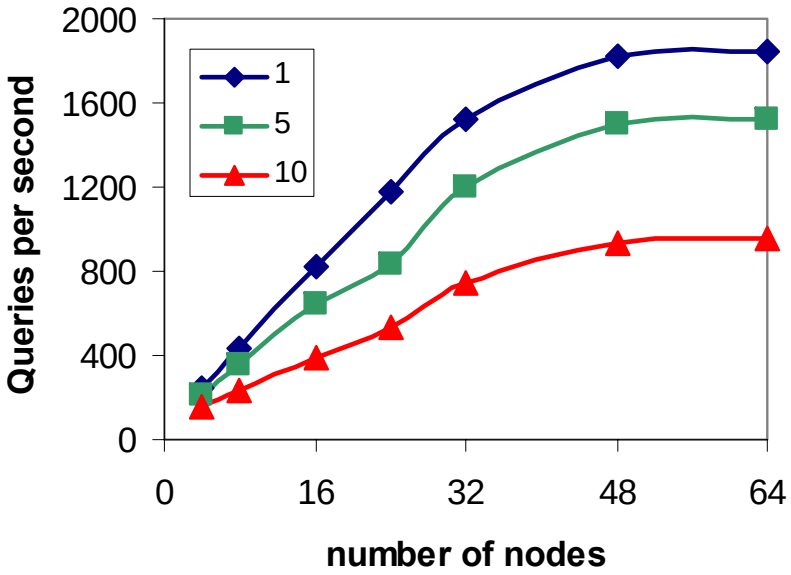
a) Fully Replicated (*FR*)



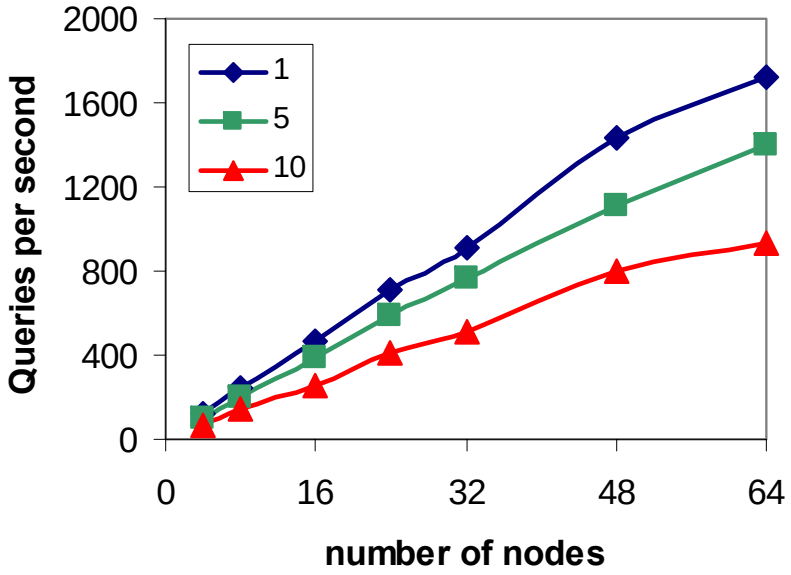
b) Partially Replicated (*PR*)

Speed up

Launch 128 clients that submit Order-status transactions (read-only)

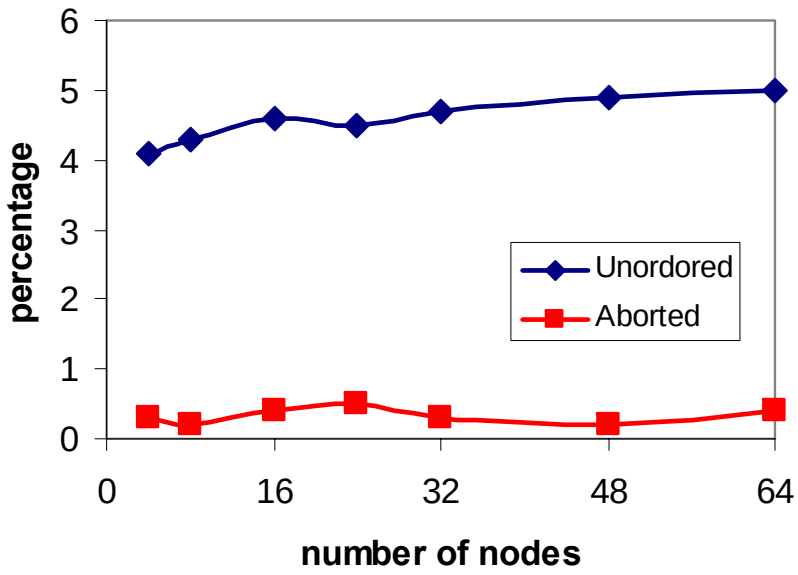


a) Fully Replicated (*FR*)

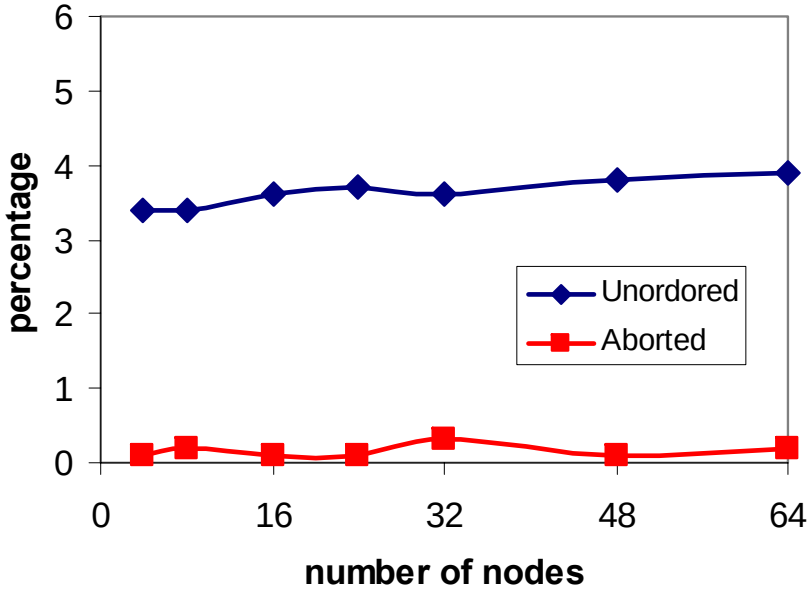


b) Partially Replicated (*PR*)

Impact of optimistic execution

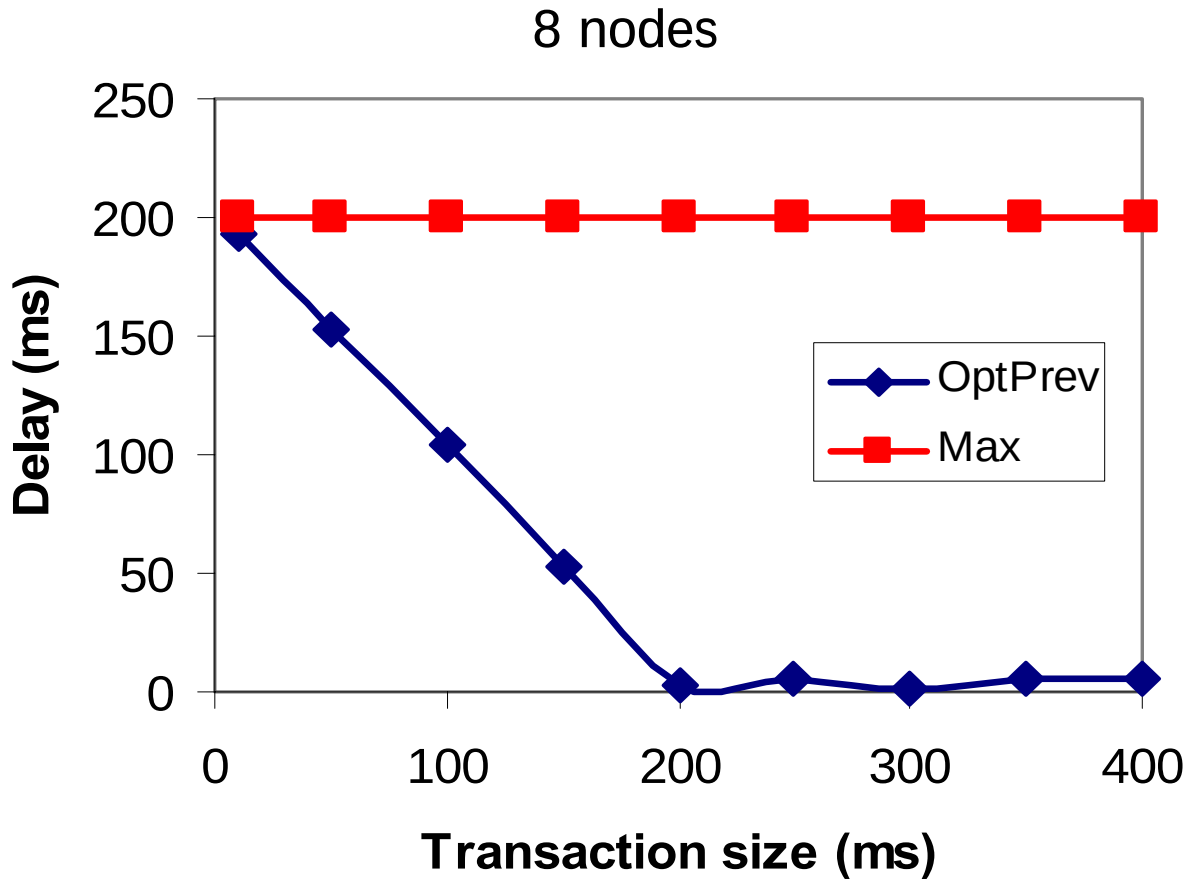


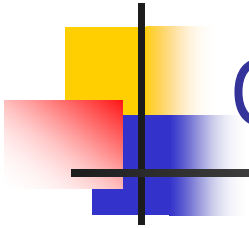
a) Fully Replicated (*FR*)



b) Partially Replicated (*PR*)

Delay vs. Trans. size





Conclusion

- Large-scale experimentation with 64-node cluster
  - Excellent scale-up and speed-up for typical OLTP
  - Important impact of optimizations
  - Time-consuming activity
- Future work
  - More experimentation going on with various loads
  - Extensions to WAN for GRID5000