JPaxos and Paxos STM for replication

Paweł T. Wojciechowski
Poznań University of Technology

Paris, 19-20 May 2011

INNOWACYJNA
GOSPODARKA
NARODOWA STRATEGIA SPÓJNOŚCI

UNIA EUROPEJSKA
EUROPEJSKI FUNDUSZ
ROZWOJU REGIONALNEGO

Motivations

Replication for

- fault-tolerance
  - crash of a subset of replicas doesn't stop service
- better performance
  - requests could be processed in parallel (load balancing)

Distributed STM has proven to be a good technology for replication.

But there are still some interesting research problems.

Goals

Replication with strong consistency

- comparison of traditional approaches and Distributed STM
- support for crash-recovery
- investigating any trade-offs
- formal semantics (for correctness proofs)

To address the above, we have implemented tools (in Java).

JPaxos

joint work with Jan Kończak, Nuno Santos (EPFL), Tomasz Żurkowski and André Schiper (EPFL)

State Machine Replication Tool for active replication

- distributed consensus using MultiPaxos
- fault-tolerant (crash-recovery)
- no Byzantine faults
- highly optimized (above TCP or UDP)
- public release available

Paxos STM

joint work with Tadeusz Kobus and Maciej Kokociński

Distributed STM for transactional replication

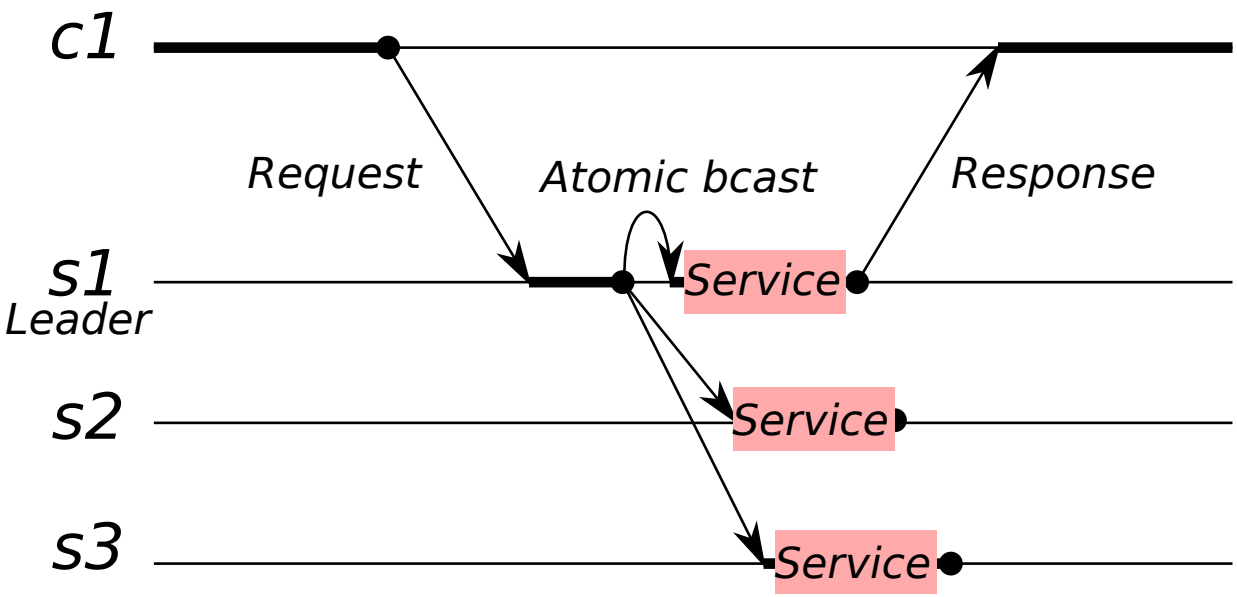
- single system image (full replication and strong consistency)
- object oriented
- atomicity, isolation (one-copy serializability), and (optionally) persistency via checkpoints
- multiple threads per node

Paxos STM (cont.)

- optimistic concurrency control (no blocking)
- certification based on atomic broadcast (multi-master, non-voting)
- built on top of JPaxos
- fault-tolerant (crash-recovery)
- easy to use API

an implementation completed (not optimized yet)

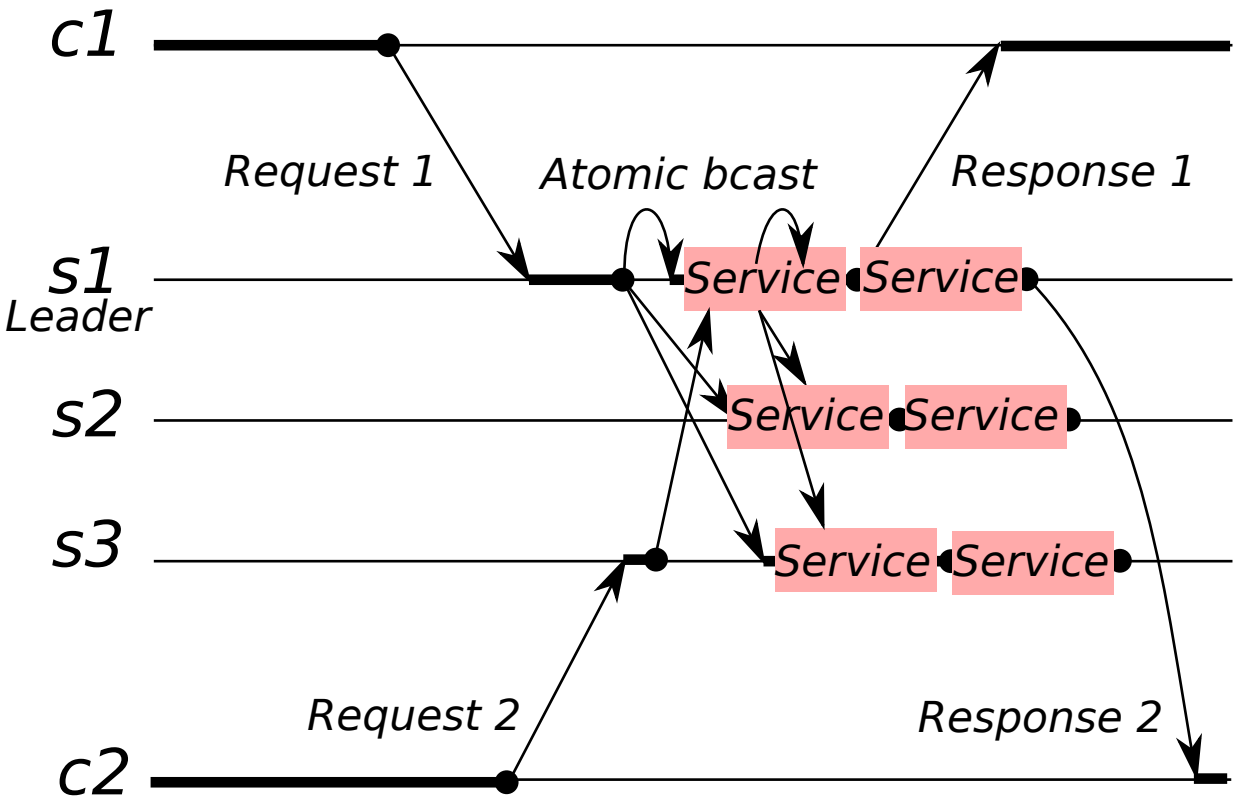
State machine replication (JPaxos)



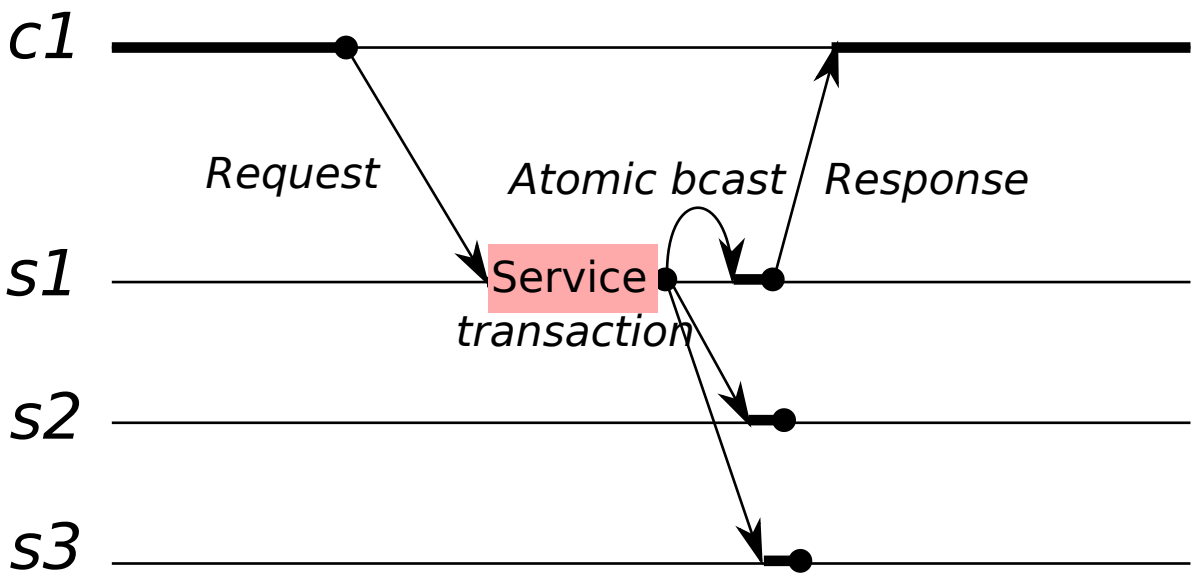
All replicas execute the same set of requests, in the same order.

Services need to be deterministic!

State machine replication (JPaxos)

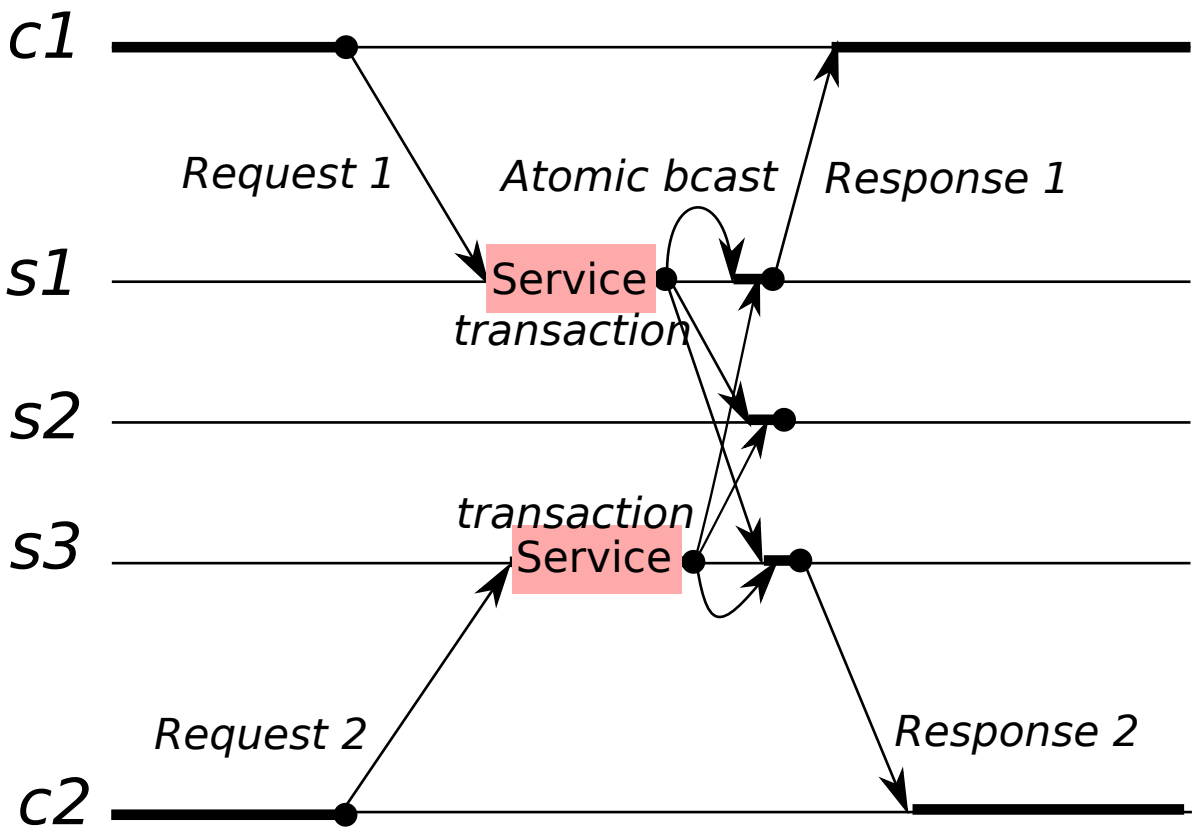


Transactional Replication (Paxos STM)



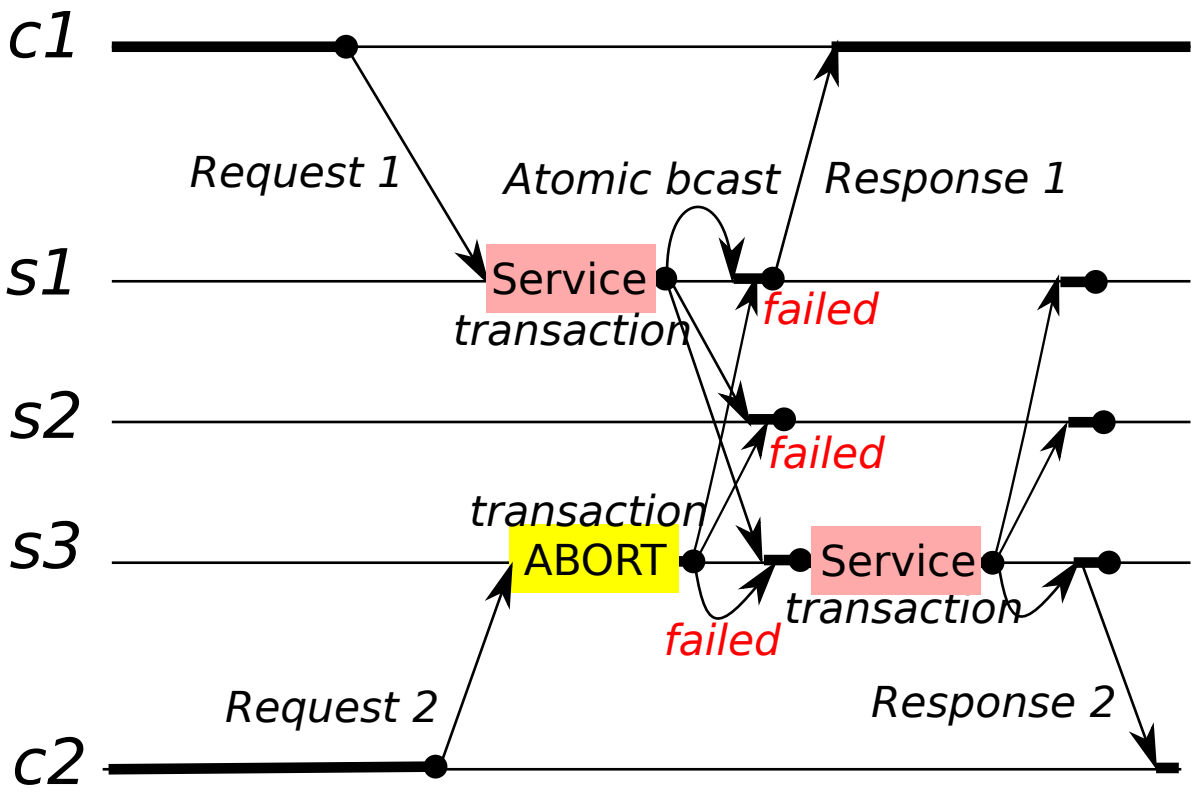
A transaction is executed entirely in a single replica.

Transactional Replication (Paxos STM)



Different transactions may be executed concurrently.

Transactional Replication (Paxos STM)



If the transaction aborts (e.g. conflict), no coordination is required.

JPaxos

Optimizations

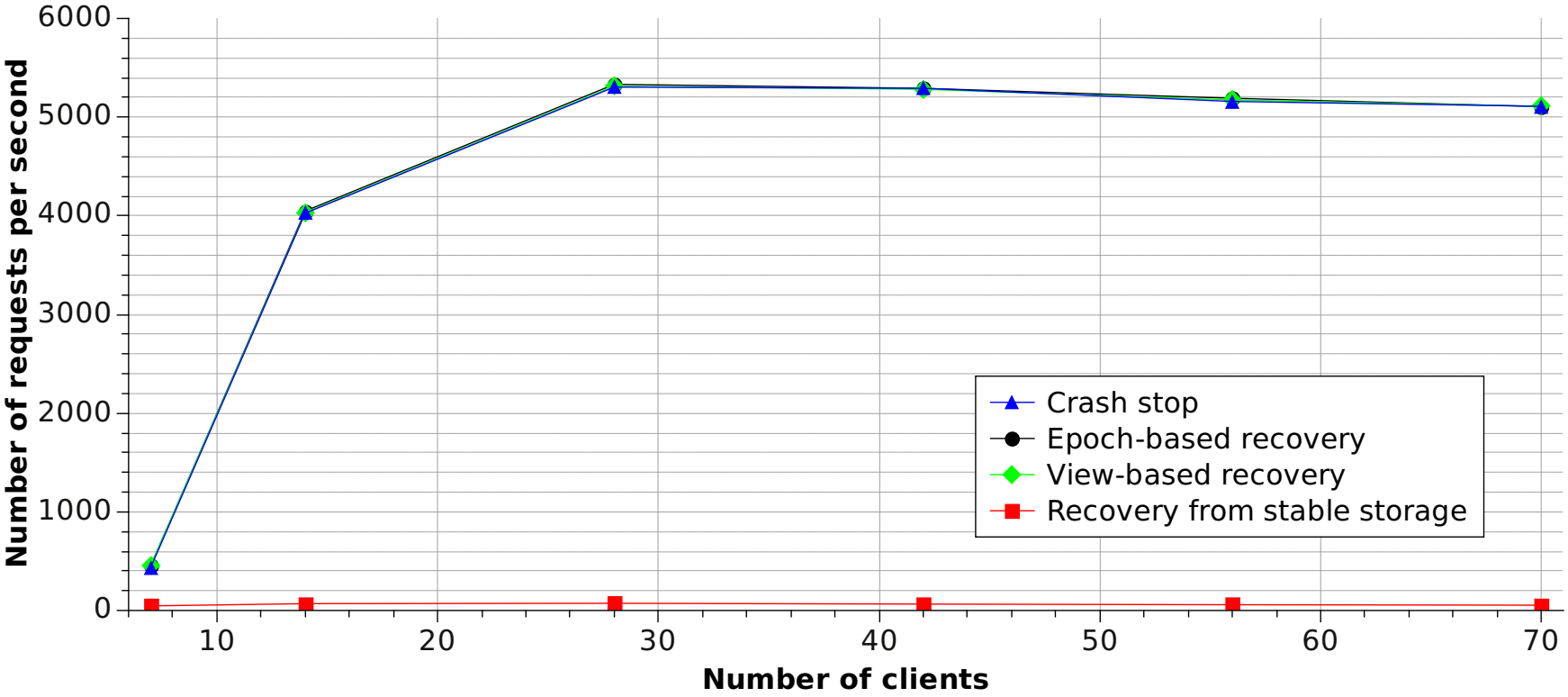
- concurrent consensus instances
  - *à la* window mechanism in network protocols
  - 2-3 concurrent instances give the best results
- message batching
  - a batch of requests processed by a consensus round
  - the batch size depends on the choice of TCP or UDP
- byte arrays or objects (with efficient custom serialization)

Crash-recovery

A choice of recovery algorithms:

- using stable storage
  - recovery from stable storage and from the state of other replicas
  - slows down the normal operation (no crashes) due to synchronous checkpoints to disk
  - all n nodes can crash (catastrophic failure)
- epoch-based or view-based recovery
  - recovery from the in-memory state of other replicas only
  - $\left\lfloor \frac{n-1}{2} \right\rfloor$ nodes can crash

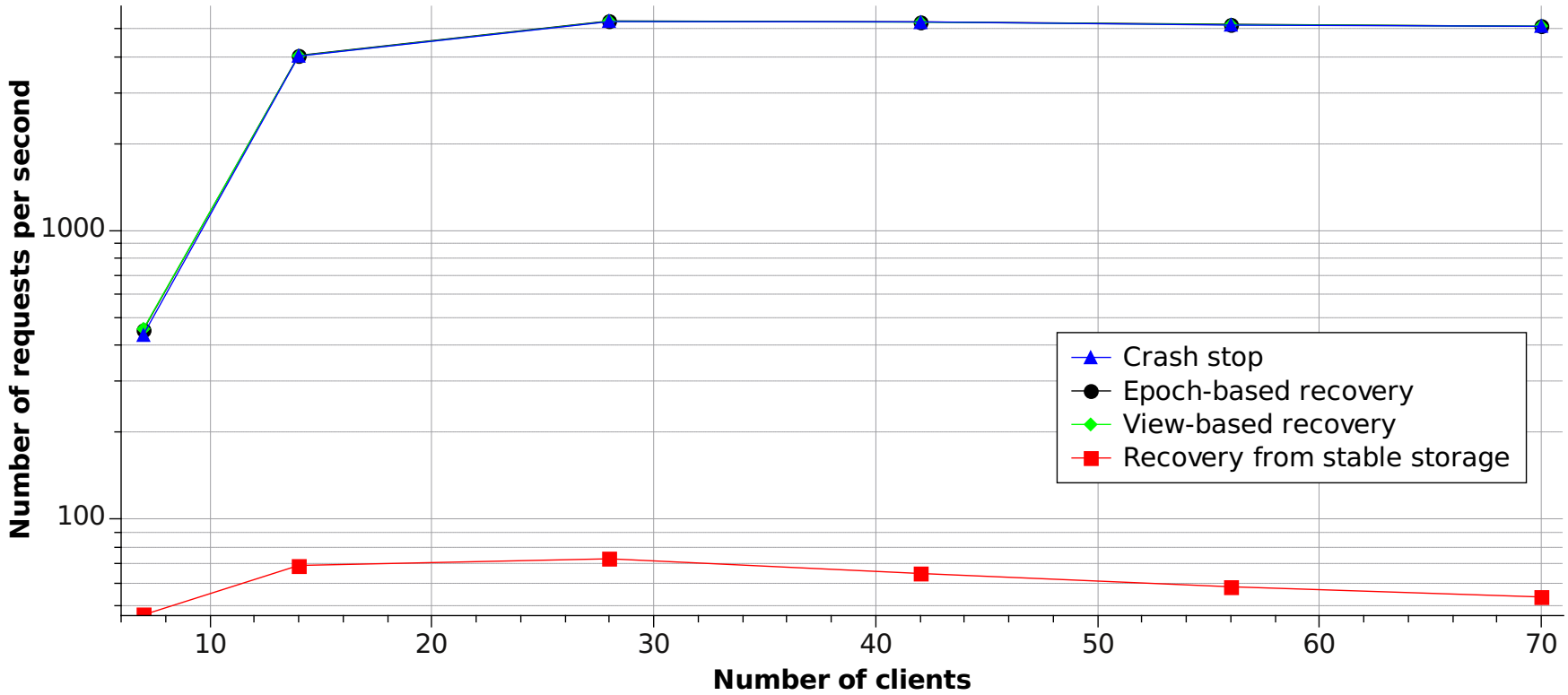
Continuous Request-Response



Normal operation (no crashes), 3 server replicas, each request and response - 8 KB.

Each node: Xeon Quad-core X3230, 2.66 GHz, L2 cache 2x4 MB, 4 GB RAM

Continuous Request-Response (in log scale)



Normal operation (no crashes), 3 server replicas, each request and response - 8 KB.

Each node: Xeon Quad-core X3230, 2.66 GHz, L2 cache 2x4 MB, 4 GB RAM

Paxos STM

Example code

```
@TransactionObject
class Account {
  private float funds;
  public float balance() { return funds; }
  public void deposit(float amount) {...}
  public void withdraw(float amount) {...}
}

new Transaction() {
  public void atomic() {
    float amount = 100;
    if (accountA.balance() >= amount)
      accountA.withdraw(amount);
    else
      retry();
    accountB.deposit(amount)
  }
};
```

Language issues

- no extensions to Java
- code instrumented before loading to JVM
- strong atomicity wrt. transactional objects
- transactional object accesses traced automatically
- additional construct for making checkpoints
- `ObjectWrapper` for OS libs

Optimizations

- reordering based certification (for fewer aborts)
  - so far, no visible gains in our benchmarks

Future work

- multi-versioning
- replacement of protocols at runtime (for different workloads)
- contention management

Validation

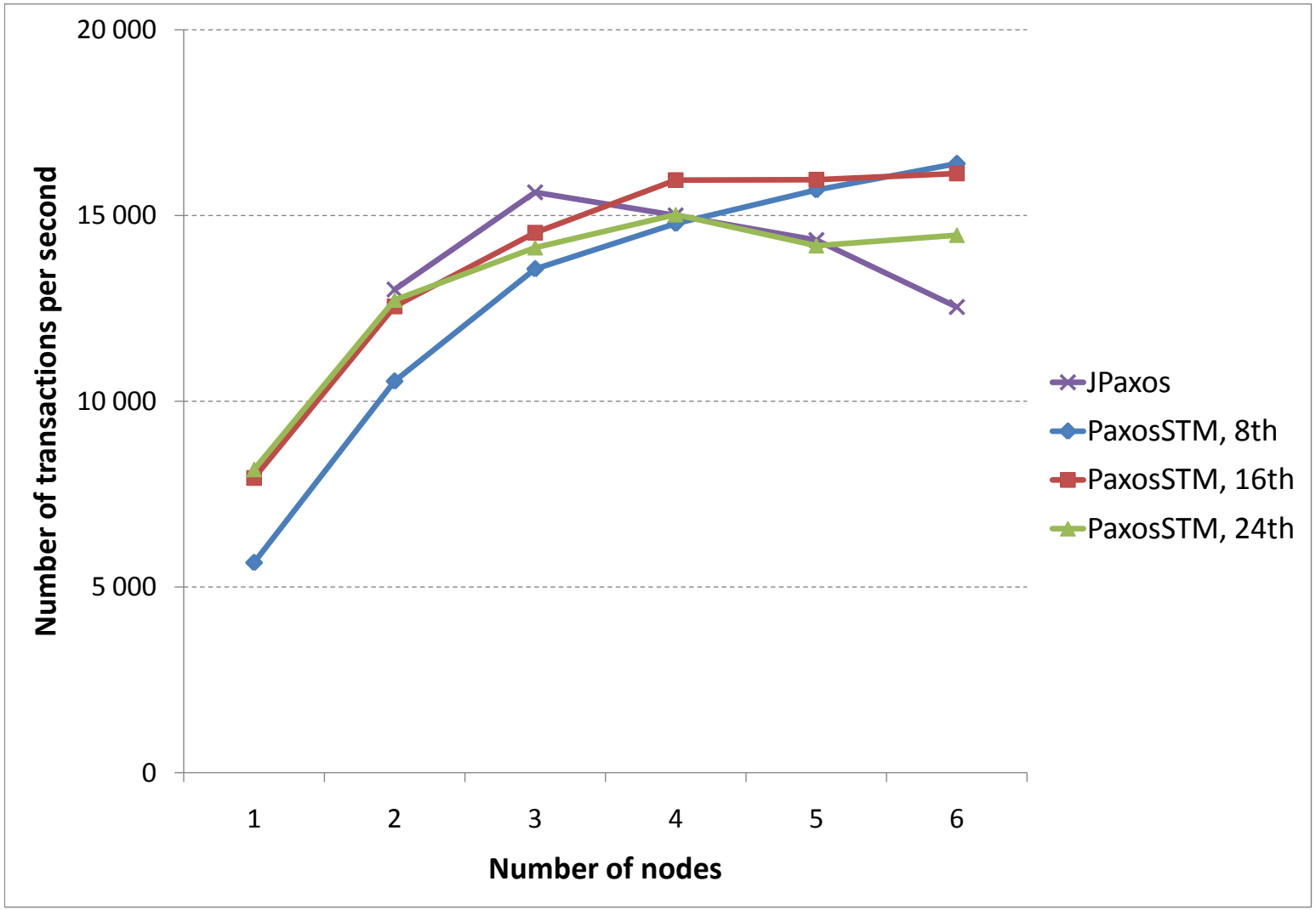
The HashMap benchmark

- a hash map of 10k elements (50% saturation)
- each element: 1 integer (4 bytes)
- transactions
  - Read-Only (RO): 100 get requests,
  - Read-Write (RW): 8 get requests + 2 put or delete

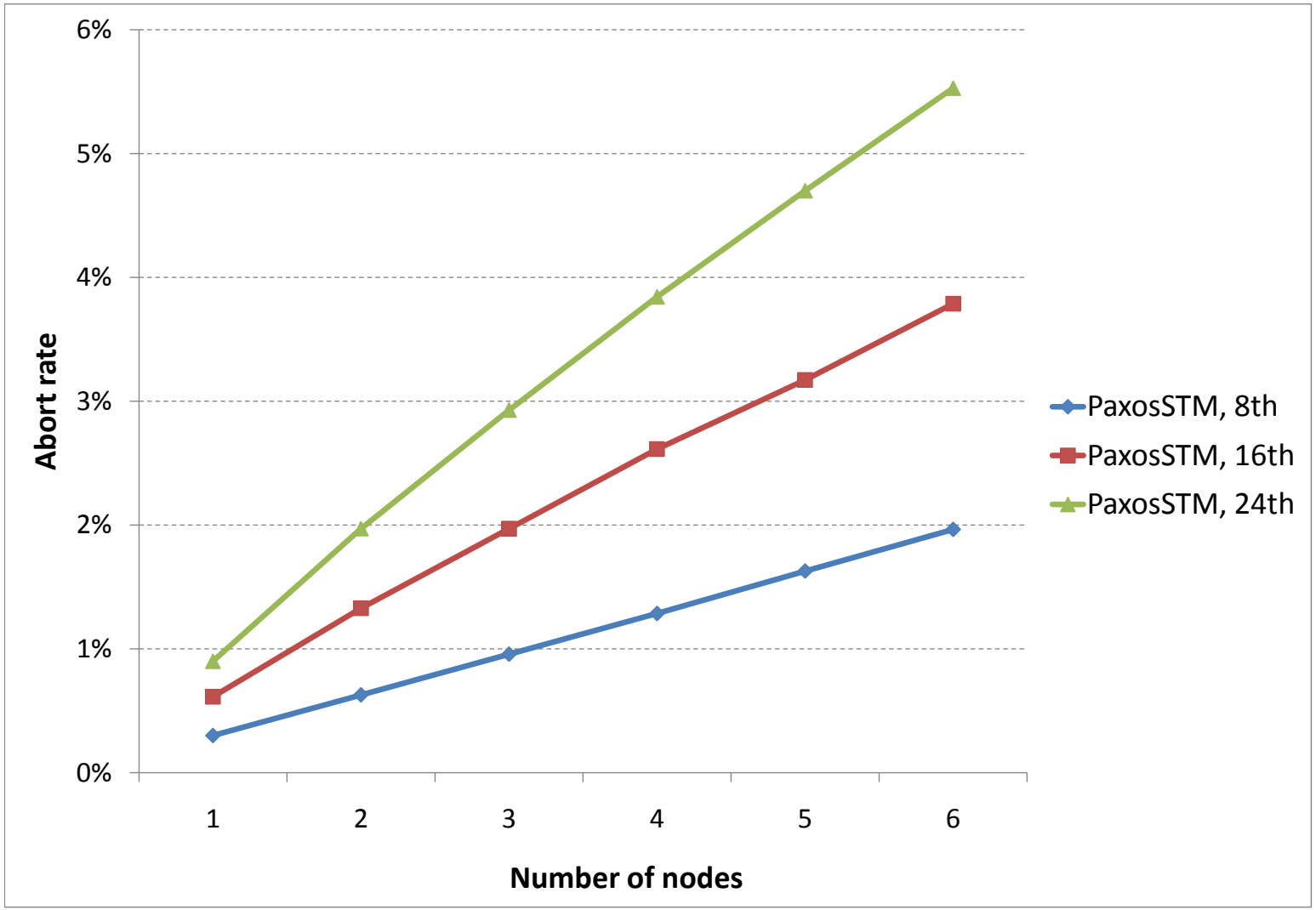
Evaluation environment

- cluster of nodes
- each node: Xeon Quad-core X3230, 2.66 GHz, L2 cache 2x4 MB, 4 GB RAM

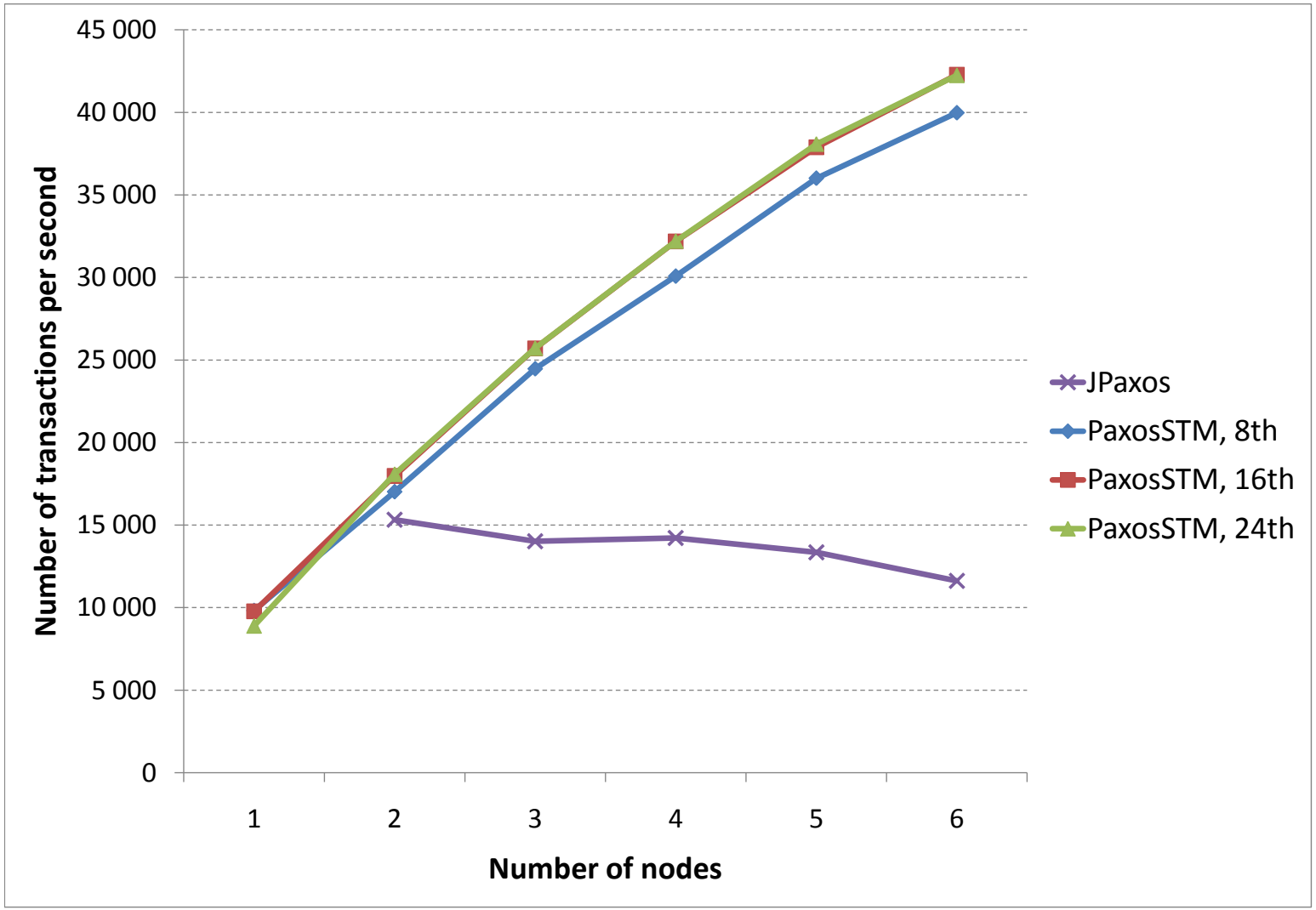
HashMap: 50% Read-Only/50% Read-Write



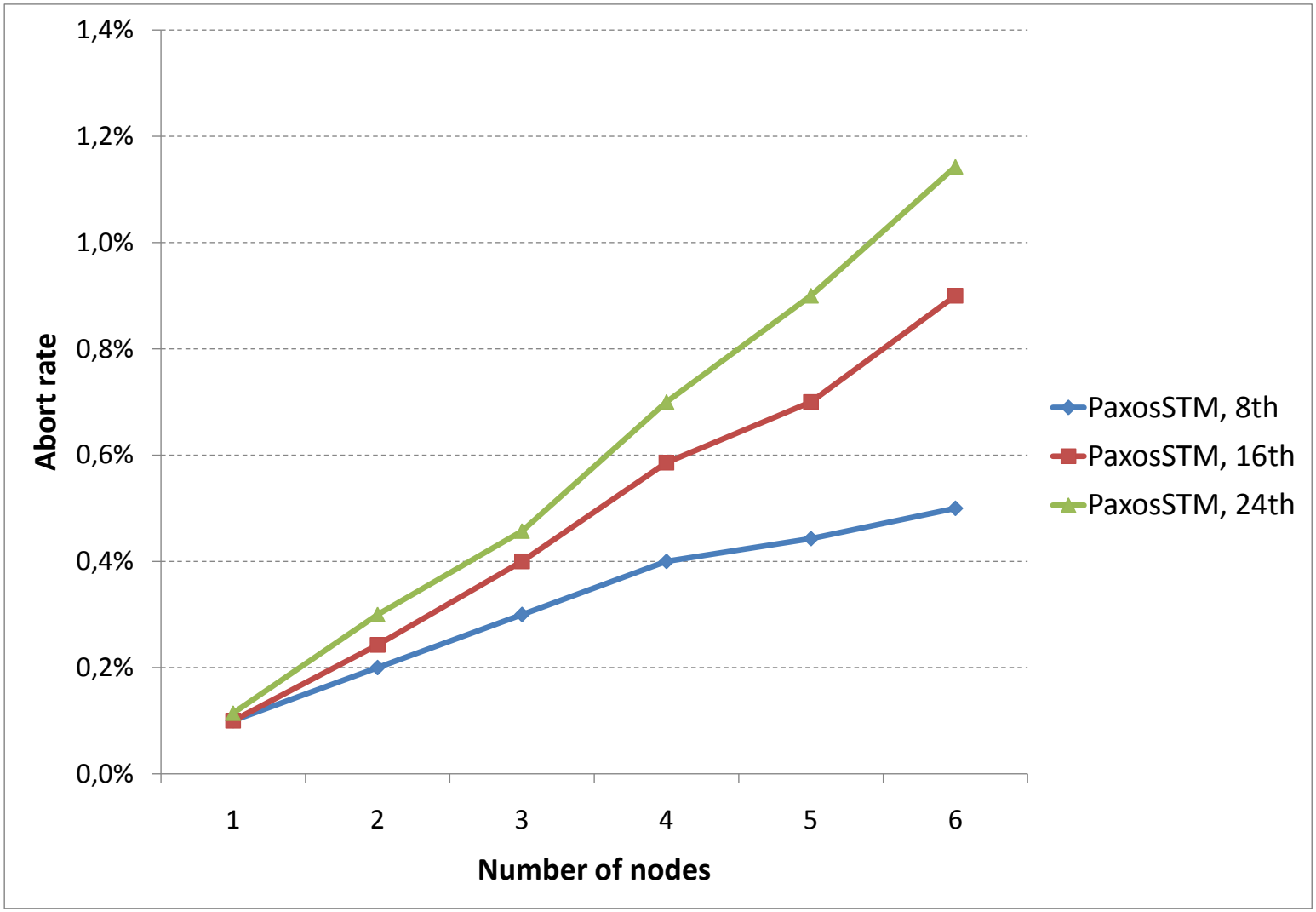
HashMap: 50% Read-Only/50% Read-Write



HashMap: 90% Read-Only/10% Read-Write



HashMap: 90% Read-Only/10% Read-Write



More information

JPaxos and Paxos STM

- `http://www.it-soa.eu/jpaxos`
- `http://www.it-soa.eu/paxosstm`

Other work (not for replication)

- Atomic RMI
  - distributed transactions in Java
  - `http://www.it-soa.eu/atomicrmi`
- Atomic REST
  - distributed transactions with automatic compensations for RESTful Web Services
  - `http://www.it-soa.eu/atomicrest`