Lightweight Logging for Lazy Release Consistent Distributed Shared Memory

Manuel Costa, Paulo Guedes, Manuel Sequeira,
Nuno Neves, Miguel Castro

IST-INESC

R. Alves Redol 9, 1000, Lisboa PORTUGAL

{msc, pjg, mds, nuno, miguel}@inesc.pt

Motivation

- DSM systems execute compute-intensive applications
- The probability of failure increases with the number of nodes and the running time
- **DSM needs fault-tolerance mechanisms**
- Process checkpointing and logging+replay are efficient ways to provide fault-tolerance

Contributions

- Definition of a new efficient logging algorithm for recovery of LRC DSM
  - Tolerates single node failures with low overhead
- Implementation on a state-of-the-art DSM system (TreadMarks)
- Experimental evaluation showing that overhead is very low

Main Contribution: Efficiency

- Sends no additional messages
- Sends only a very small amount of extra data
- Only one message is changed
- Logs a minimal amount of data
- No communication-induced accesses to stable storage

⇨ Performance penalty < 1%

System Model

- Lazy-invalidate release consistent DSM (TreadMarks)
- Processes exchange messages reliably
- Fail-Stop
- Piecewise-determinism
  - The only non-deterministic events are shared memory reads
- Programs have no data races

Algorithm Overview: Failure Free

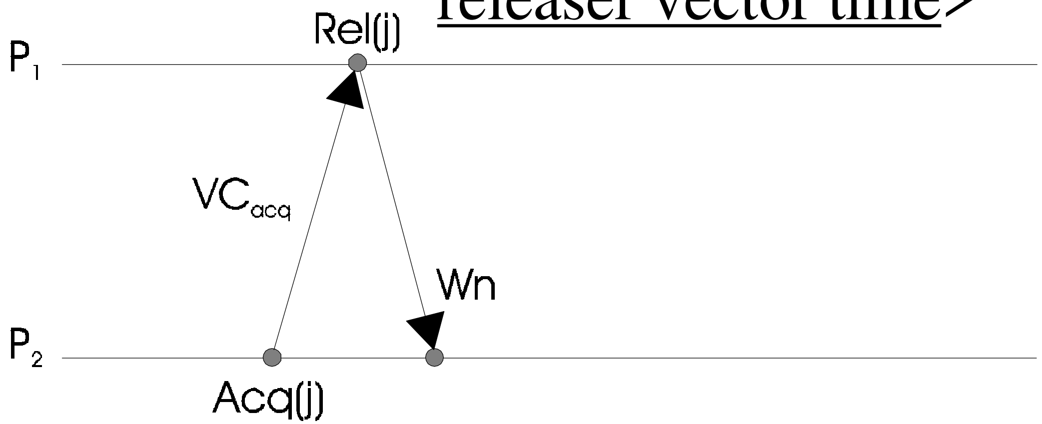
- Logging to tolerate single failures
  - Log vector time pairs and logical times pairs at
    - Lock acquires, Barrier crossings
  - Tag some data structures (diffs) with logical time of creation
- Checkpointing
  - Independent checkpointing to speed up recovery
  - Consistent checkpointing during LRC GC to tolerate multiple failures

Algorithm Overview: Recovery

- Re-start the most recent checkpoint of the failed process on any available processor
- Collect recovery information from the logs of the surviving processes
- Roll-forward the process, replaying page invalidates and shared memory reads
- Asynchronous messages make recovery more complex

Logging at Acquires

Log <acquirer vector time,
releaser vector time>



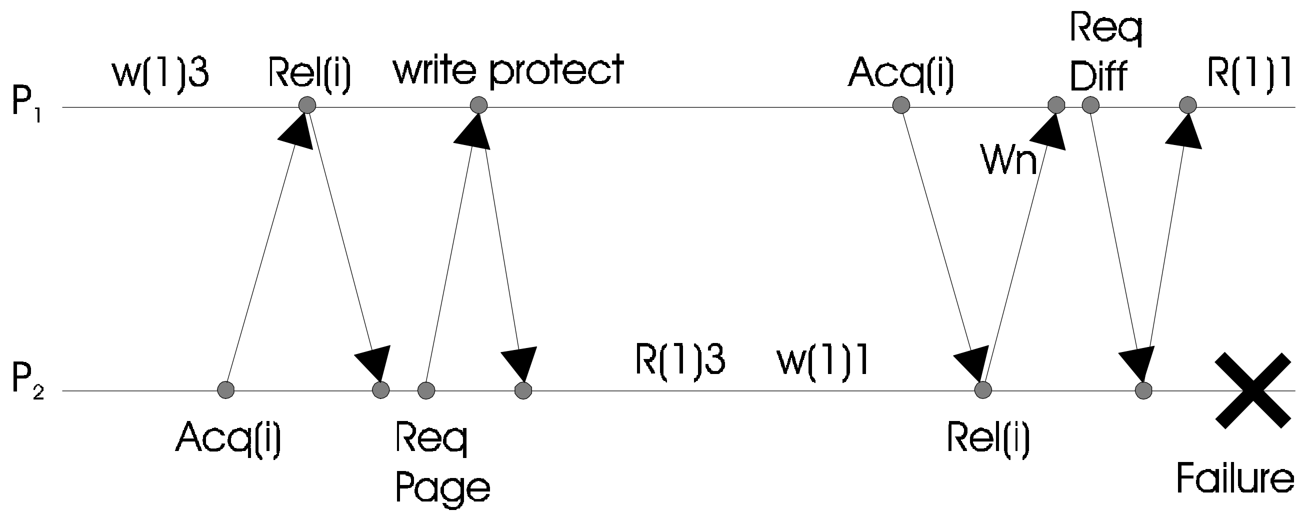
Log <vector time before acq,
vector time after acq>

- **Only pairs of vector times need to be logged**

Recovery

- Data collection
  - Build a list of tuples <lt, intervalSet>
- Execution Replay
  - Page invalidates are replayed at each acquire
  - Same page and diff requests are issued
  - Diffs received are the same
  - Pages received are equivalent

Page Request Replay

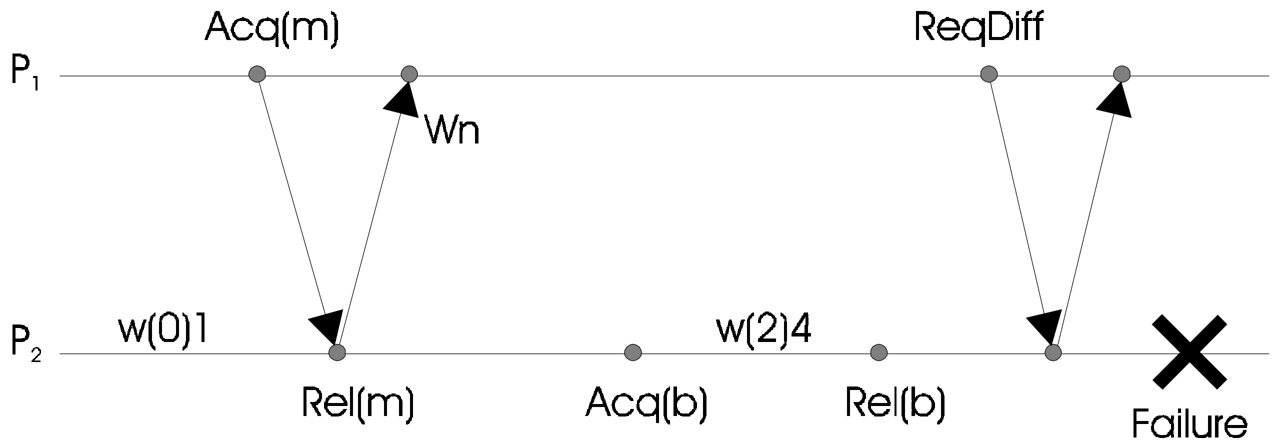


- After GC, all pages are saved in a consistent checkpoint
- Duplicate page requests are serviced from the last consistent checkpoint

Asynchronous Messages

- Some LRC data structures are influenced by asynchronous message handling
  - Intervals
  - Diffs
- Asynchronous messages cannot be replayed
- These data structures are mostly recovered from other processes
- Some require relatively complex recovery procedures

Asynchronous Requests and Recovery Termination



- Diffs are tagged with the logical time of creation
  - recovery ends after the last diff creation

Previous Work

- Most previous algorithms for recoverable DSM require communication-induced accesses to stable storage
  - [Suri 95, Janssens 93, Richard III 93, Wu 90]
- Some protocols only recover shared memory state (not process execution state)
  - [Kermarrec 95, Feeley 94, Stumm 90]
- Sender-based message logging [Johnson 87]:
  - All messages are logged
  - Logs message copies
  - Does not handle asynchronous messages
- Consistent checkpointing [Cabillic 95, Janakiraman 94]

Implementation

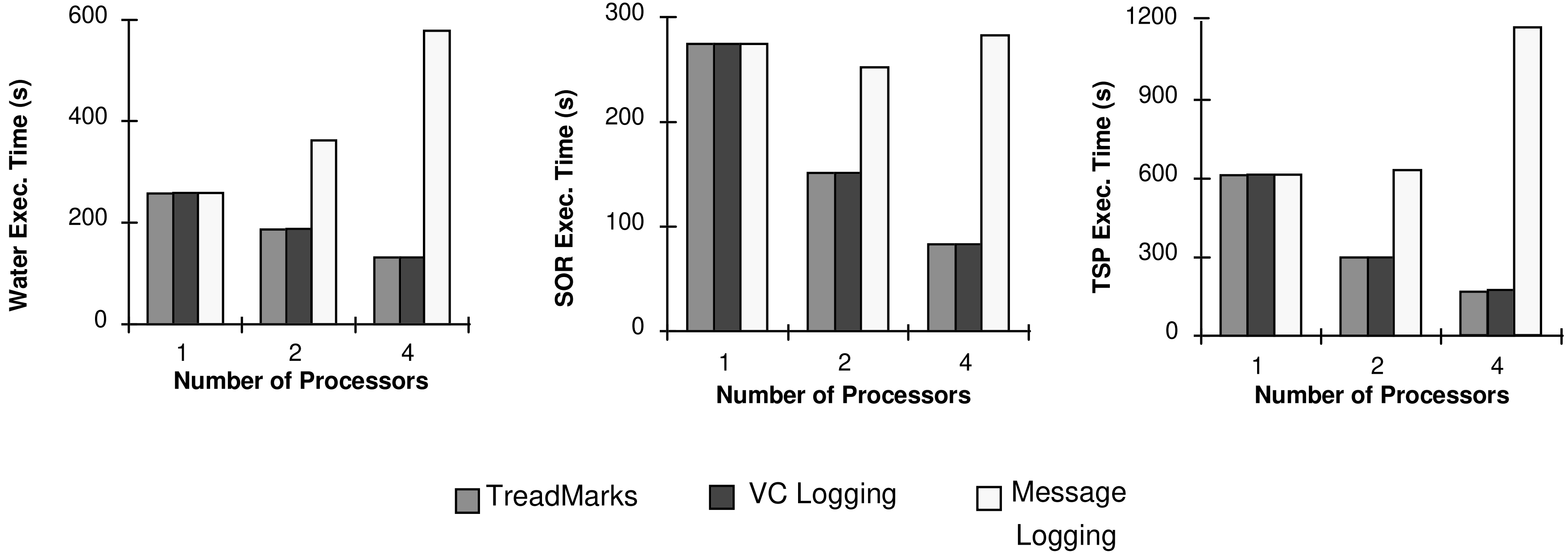
- Logging required small changes to TreadMarks
  - Code was added to log vector time pairs and logical time pairs
- Independent checkpoints were implemented with libckpt from [Plank 95]
- Recovery is partially implemented

Experiments

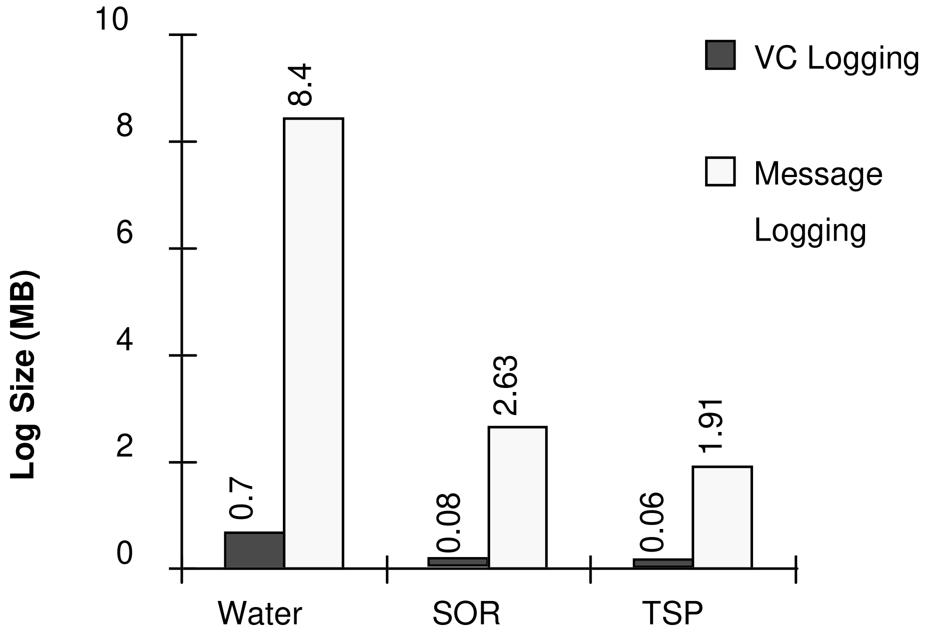
- Network of Sun Sparcstations10/30
- 10 Mbps Ethernet
- Checkpoints are written to an NFS file server
- Measurements
  - TreadMarks
  - VC Logging
  - Message Logging [Suri 95]
    - Logs copies of messages received at acquires
    - Logs references to diffs received at access misses
    - Synchronously flushes the log to the local disk

Time Overhead

- VC Logging overhead is below 1%
- Message Logging eliminates speedup



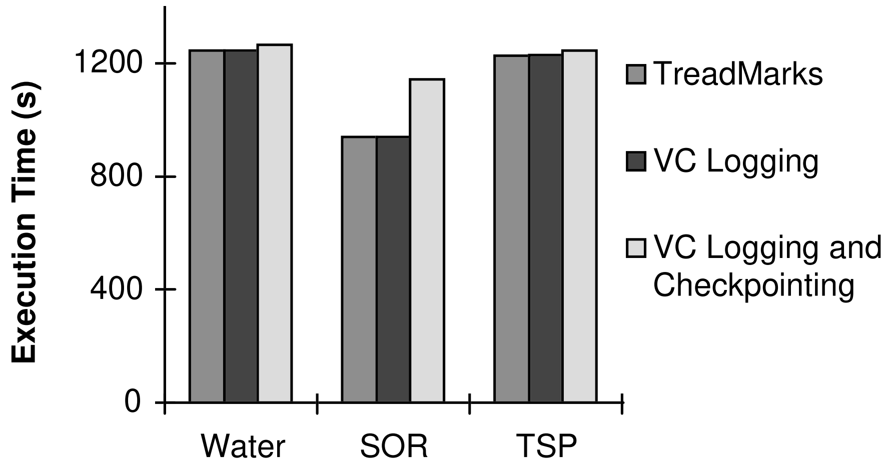
Space Overhead



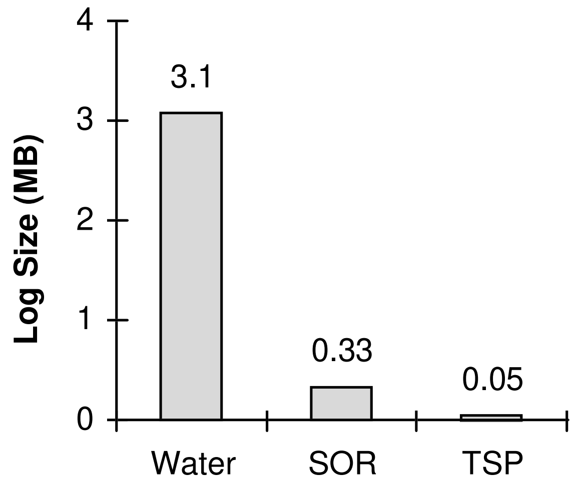
- In VC Logging log sizes are between 3% and 8% of log sizes in Message Logging

Overhead with Checkpointing

Time

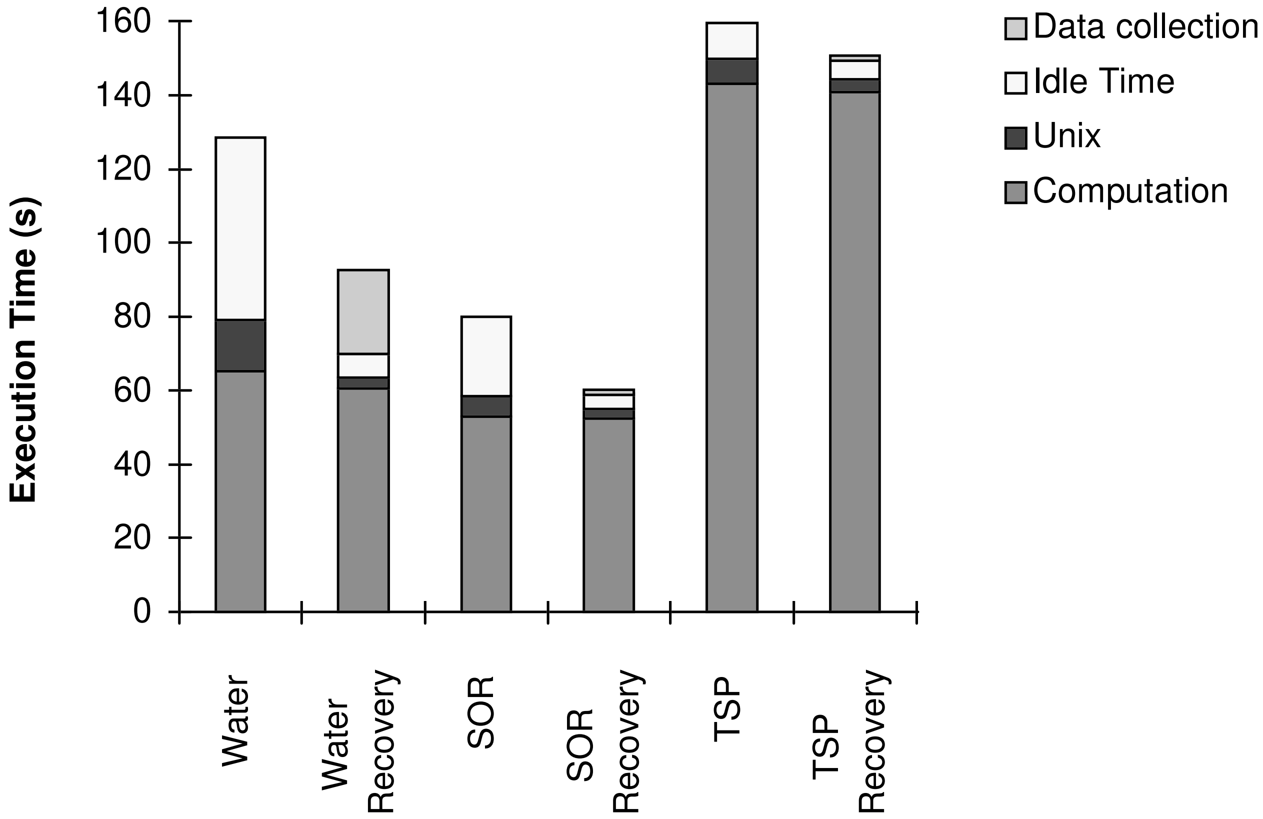


Space



- Overhead of checkpointing is below 2% for Water and TSP, for SOR it is 22%

Recovery Time



- Recovery time is 72% of normal execution for Water, 75% for SOR and 95% for TSP

Conclusions

- Experimental results show that the overhead of logging for recovery of LRC DSM is very low
  - Time overhead below 1%
- Good performance is achieved through tight integration with the LRC memory consistency protocol
- Recovery is potentially faster than in pure consistent checkpointing