

Fault Tolerance with Independent Checkpointing in Distributed Shared Memory

Florin Sultan, Liviu Iftode

Department of Computer Science, Rutgers University, New Jersey
{sultan, iftode}@cs.rutgers.edu

1. Introduction and motivation

A straightforward approach to recovery from failures in a distributed system is to restore its state from a globally consistent checkpoint. The cost of taking a coordinated checkpoint can be amortized if the checkpoint is taken during system or application induced global synchronization operations. Typical Distributed Shared Memory (DSM) systems like TreadMarks [4] need to periodically run a garbage collector to reclaim memory used by the DSM protocol data structures. Existing fault tolerant implementations of DSM systems [2] rely on global synchronization implicit in the operation of the system, which is used to take coordinated checkpoints. Any previous checkpoints and logs on stable storage can thus be safely discarded and the system does not have to face the problem of management of checkpointed state.

Our primary goal is to investigate how to implement fault tolerant DSM using independent checkpointing. There exist DSM protocols like the Home-Based Lazy Release Consistency protocol (HLRC) [1] that do not enforce global operations, and artificially introducing them with fault tolerance support would impact performance. In addition, with uncoordinated checkpointing individual processes can decide on when and what to checkpoint. Finally, taking a coordinated checkpoint may not be always possible, for example in applications that tolerate weakly connected operation.

2. Background

A major problem in distributed rollback recovery [3] is enforcing a global consistent cut in the state of individual processes after a failure and restart. One technique is to use coordinated checkpointing to establish a consistent recovery line. The alternative - taking independent checkpoints - suffers from the problem of domino effect, when a single failure may potentially force all processes to rollback. Among advantages of independent checkpointing is that it enables optimizations like memory exclusion [5]. An orthogonal technique is to use checkpoints and log messages received by a process. During re-execution, messages are replayed in the order

they were received. Such log-based protocols can ensure that the maximum recoverable state is exactly the state of the system before crash. Pessimistic logging of all messages has this desired feature, while not suffering from the well-known problems of uncoordinated checkpointing. The downside is the high overhead incurred by logging every message.

In a DSM, updates to memory pages are encoded as *diffs* representing the differences between a reference version and a version modified due to writes performed during a given interval, labeled by the logical time at the moment of their generation. Pages are versioned by a vector timestamp that encodes knowledge about the last *diffs* that were applied from some process.

3. Our approach

We want to build a DSM system that can recover from single node failures and retain the advantages of uncoordinated checkpointing and fast log-based recovery. Because every memory access may potentially result in an inter-process interaction, logging overhead is critical. The single node failure assumption allows using efficient sender-based logging in volatile memory. Additionally, we aggressively use protocol knowledge to reduce overhead by selectively logging protocol messages.

For example, in a home-based DSM [1] full page transfers need not be logged. Instead, we log messages carrying *diffs* and checkpoint pages at their home nodes. The recovery procedure uses a checkpointed reference version of a page to which it incrementally applies *diff* logs to locally construct evolving versions of the page. This procedure is applied both for restoring a page at a home node and for the local replay of accesses during recovery by a non-home node. In the second case, we dynamically reconstruct a minimal LRC-consistent version of the page, instead of the actual page that was received by the process during the failure-free execution.

4. Checkpoint and log management

In log-based protocols with independent checkpointing the volatile logs must be saved to stable storage and there

are no pre-determined moments in time when they can be safely discarded. To control log size in either volatile or stable storage, obsolete log entries, i.e., which will never be needed by some recovering process, are discarded.

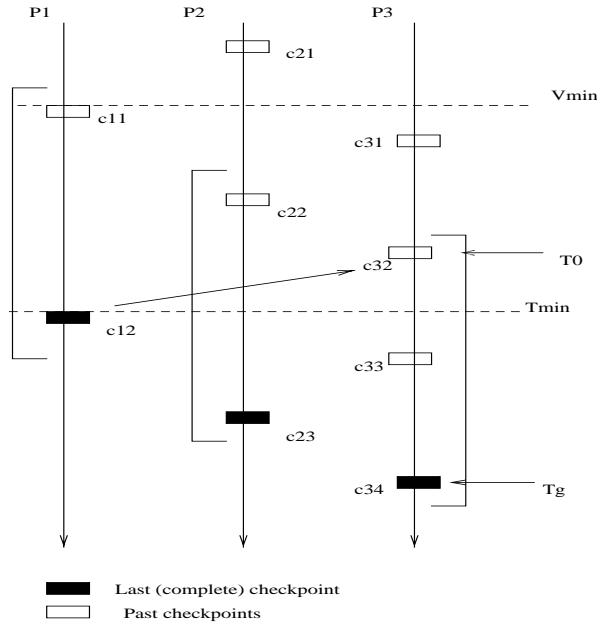


Figure 1. Checkpoint and log management

Our approach is to do diff log management (trimming) in a lazy fashion, avoiding global synchronization. The mechanism we propose allows a process to locally initiate and execute the trimming procedure, without involving other processes.

We timestamp checkpoints with any good approximation of the global logical time. A candidate approximation could be the vector timestamp used in DSM protocols to track the (partial) ordering of synchronization events occurring at any process. Unfortunately, the DSM vector timestamp accurately represents the global logical time only after a global synchronization operation is performed. In the absence of such operations, it may arbitrarily lag behind. Our solution is to use a centralized manager to approximate the global time. Processes (lazily) propagate their local logical time to the manager. When taking a checkpoint, a process obtains a checkpoint vector timestamp from the manager. Log entries and pages included in the checkpoint are labeled with the checkpoint vector timestamp.

In order to support recovery of any other process P_j , a process P_i keeps checkpointed copies of pages for which it is home, starting from the most recent checkpoint timestamp which is not more recent than T_{min} , the oldest timestamp of the last checkpoint over all other processes. As a result P_i retains minimal page versions that can cover the recovery of any other process. For trimming diff logs, note that no process ever needs diffs older than the oldest

copy of a page retained from any checkpoint. A minimum over the oldest checkpoint vector timestamps of pages retained by any process gives a bound V_{min} on the timestamps of log entries that need to be kept. A process can discard diff log entries labeled with checkpoint vector timestamps no larger than V_{min} .

Since the two bounds T_{min} and V_{min} needed for the trimming decisions are only based on checkpoint vector timestamps stored at the manager, they can be computed by the manager and provided to a process about to take a checkpoint, upon request.

In Figure 1, process P_3 takes a checkpoint at global time T_g . Brackets enclose checkpoints from which page copies are to be retained. There is a dependency from the bound T_{min} (attained by P_1) to checkpoint c_{32} , so only pages from c_{32} and c_{33} are retained. Only log entries created after V_{min} , the oldest timestamp of a copy of a checkpointed page in the system, are retained by P_3 . Horizontal lines represent the ideal global time, which is approximated by the global time as seen by the manager. The key observation is that the manager enforces a total order on the checkpoint vector timestamps that are used in computing the bounds for all trimming decisions.

We have obtained simulation results showing that our scheme effectively bounds the size of checkpointed data (number of per-page copies and diff logs). On three update-intensive applications from the SPLASH-2 suite we have achieved a common case value of three checkpointed copies per shared page to be kept by a home node. We are currently implementing a prototype of the system for the HLRC protocol running on a cluster of Pentium PC's interconnected by a Myrinet network and using a virtual memory-mapped communication system.

5. References

- [1] L. Iftode, "Home-Based Shared Virtual Memory", Ph.D. Thesis, Princeton University, June 1998.
- [2] M. Costa, P. Guedes, M. Sequeira, N. Neves, M. Castro, "Lightweight Logging for Lazy Release Consistent Distributed Shared Memory", Proc. USENIX Symposium on Operating Systems Design and Implementation (OSDI), October 1996.
- [3] E. N. Elnozahy, D.B. Johnson, Y. M. Wang, "A Survey of Rollback-Recovery Protocols in Message-Passing Systems", TR CMU-CS-96-181, Carnegie Mellon University, October 1996.
- [4] P. Keleher, S. Dwarkadas, A.L. Cox, W. Zwaenepoel, "TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems", Proceedings of the Winter 94 Usenix Conference, January 1994, pp. 115-131.
- [5] J. S. Plank, Y. Chen, K. Li, M. Beck, G. Kingsley, "Memory Exclusion: Optimizing the Performance of Checkpointing Systems", Software -- Practice and Experience, Volume 29, Number 2, 1999, pp. 125-142.