A Theoretical Model of Optimistic Recovery in Distributed Systems

Willy Zwaenepoel
Department of Computer Science
Rice University
P.O. Box 1892
Houston, TX 77251-1892
willy@rice.edu

May 8, 1990

Final Technical Report
Contract No. N00014-88-K-0140

1 Overview

The research sponsored by this contract has produced the following results:

1) A model for recovery using message logging and checkpointing, resulting in a major theorem establishing the uniqueness of a maximum recoverable system state;

2) An implementation and a performance evaluation of sender-based message logging;

3) A batch and an incremental algorithm for optimistic recovery, and an implementation and a performance evaluation of optimistic recovery;

4) Preliminary results in extending these methods to nondeterministic processes; AND

5) A new definition of distributed breakpoint, called *causal distributed breakpoint.* (KR)

The initial ideas underlying sender-based message logging were presented at the 1987 Fault-Tolerant Computing Symposium [10]. In a paper recently submitted to Transactions on Computer Systems, we discuss several extensions and optimizations, and we provide a definitive account of an implementation and its performance [13]. The model and a preliminary version of the incremental optimistic recovery algorithm were presented at the 1988 ACM Principles of Distributed Computing symposium [11]. A revised version was invited for a Journal of Algorithms special issue on distributed computing [12]. An implementation of optimistic recovery using a new batch algorithm and its performance are described in David Johnson's Ph.D. thesis [8]. The extensions to nondeterministic processes are described in a paper recently submitted to the Symposium on Reliable Distributed Software [13] and in a submission to the European SigOps Workshop on Fault Tolerance [14]. Finally, we devised a new definition of distributed breakpoint, called *causal distributed breakpoint*, which will appear at the Tenth International Conference on Distributed Computing Systems [7].

Other researchers have continued the development of some of the ideas developed under this contract. In a paper at the 1988 Fault-Tolerant Computing Symposium, Strom et al. discuss the extension of sender-based message logging to multiple failure recovery [17]. Sistla and Welch used our model to derive a distributed recovery algorithm and presented it at the 1989 Principles of Distributed Computing symposium [16].

We also continued our work on optimistic computation, resulting in two papers at the 1989 Sigmetrics and Performance conference: one on optimistic implementation of bulk data transfer protocols [4] and another on optimistic make [3]. An extended version of the latter paper has been submitted to IEEE Transactions on Computers, and is currently being revised before publication.

2 A Model for Message Logging and Checkpointing

The model is structured around the notion of a *dependency*. When a process receives a message from another process, the recipient's state depends on the state of the sender at the time the message was sent. To capture this dependency information efficiently, each process' execution is divided in state intervals, with a new state interval being started each time a message is received. The state interval index, incremented by one on each message receipt, uniquely identifies a state interval within a process. All messages sent within a state interval are tagged with the current state interval index. When a message is received, the recipient process updates its *dependency vector*, a vector with one entry for each process indicating the highest state interval index tag received from that process. For process i, entry i in the dependency vector of state interval σ is set to σ.

The state of an individual process and its dependencies is identified by this dependency vector. A system state is defined as a collection of process states. It is represented by a *dependency matrix*, with each row containing the dependency vector of the corresponding process state.

The dependency matrix allows a crisp mathematical definition of the notion of *consistency*. A system state is consistent if it could have occurred during a failure-free execution of the system. In a system where all interprocess communication is by messages, this is roughly equivalent to requiring that all messages received have been sent. Translated in terms of the dependency matrix, this means that the diagonal element in each column must be no smaller than each off-diagonal element in the same column. Given this observation, it is easy to show that the set of consistent states that have occurred during a computation forms a lattice. Recoverable system states are consistent system states in which all individual process states can be recreated from disk. The set of recoverable system states that have occurred during a computation also forms a lattice, which in turn leads to the central theorem showing the uniqueness of the maximum recoverable system state.

The model is independent of any message logging protocol. We have used it to show the correctness of two message logging protocols: pessimistic sender-based message logging and optimistic logging.

3 Sender-Based Message Logging

Roughly speaking, pessimistic message logging protocols log messages *before* they are received by the destination process. This allows *independent* recovery. By restoring a failed process to its last checkpoint and replaying the messages from the message log, the failed process can be restored to its state before the failure, without the recovery affecting any other processes. Unfortunately, the delay incurred by waiting for each message to be logged before it can be received is generally unacceptable, except for single failure recovery.

For single failure recovery, it suffices to write the messages to a volatile log on a machine different from the receiver. Sender-based message logging logs the message on the machine from which it was sent, thereby avoiding the cost of transmitting it to a log on a third machine. In order to record in the log the order in which the message was received, the receiver returns as part of the acknowledgement a *receive sequence number*, which is inserted with the message in the sender's log. The model described in Section 2 was used to prove the correctness of sender-based message logging and its exact characteristics, namely single failure recovery and multiple failure detection. Additionally, a number of important performance optimizations, including piggybacking RSNs, piggybacking RSN acknowledgements, and "bunching" RSNs were shown to be correct, again using the model.

Sender-based message logging was implemented on a network of SUN 3/60 workstations connected by a 10 Megabit Ethernet and running the V-System [6]. Several applications were run with this system. A full account of the implementation and the performance of sender-based message logging is given in [13]. In summary, the V-System communication primitives incur approximately 25 percent extra overhead as a result of the message logging. The overhead for a specific application is determined by their communication to computation ratio and by the checkpointing frequency. Typical values are between 3 and 15 percent.

Unlike other pessimistic message logging protocols [1, 2, 15], sender-based message logging does not require any special hardware for fault tolerance. Nevertheless, overhead appears quite acceptable, calling into question the need for such specialized hardware.

4 Optimistic Recovery

With optimistic recovery, messages are logged *asynchronously*, resulting in much smaller performance degradation during failure-free computation. However, independent recovery is no longer possible. A failed process can only be recovered to the state obtained by restoring it to its last checkpoint and replaying the messages in its message log. Further states that may have existed before the failure are unrecoverable. If the failed process sent a message from such an unrecoverable state, the receiver of that message becomes an orphan after the failure, and must be rolled back to a point before the message was received. The dependency vector information is crucial in detecting orphans. In effect, the recovery algorithm must determine the "most recent" state that can be recreated from disk and that does not contain any orphans. The model shows that this most recent state is unique. An incremental and a batch algorithm for finding the maximum recoverable state were shown to be correct using the model. The tradeoffs between the two algorithms are not clear at this point and further study, potentially including the development of new algorithms, is required.

A prototype implementation of optimistic recovery using the batch algorithm has been completed [8]. Overhead for the various V-System communication primitives is approximately 10 percent. The running time of the batch recovery algorithm is minimal compared to the overhead of checkpoint restoration and replay. We do not have enough experience at this point to comment on the amount of rollback necessary as a result of orphan handling.

Our recovery algorithm has several advantages over previously published results [16, 18]. It is guaranteed to find the maximum recoverable state, while other algorithms only guarantee to find some recoverable state. Also, it only requires a constant amount of information to be included with each message, while others require an amount of information linear in the number of processes. Finally, to the best of our knowledge, our implementation of optimistic recovery is the first full-fledged implementation of such a system.

5 Extension to Nondeterministic Processes

All our previous results assume that processes execute *deterministically* between received messages. In other words, if two processes start execution in the same state and receive the same sequence of messages, they will terminate in the same state and send the same sequence of messages. This assumption is necessary in order to guarantee that processes by replay of messages received before the failure reach the same state after recovery. This assumption can be violated by the presence of asynchronous events such as signals and interrupts, or by multithreaded processes who behave nondeterministically as a result of scheduling decisions. For nondeterministic processes, only checkpoints can be used in recovery, since message replay will not necessarily result in the same state being reached after recovery.

It appears that our results extend quite well to nondeterministic processes by changing the definition of a state interval [9]. Instead of incrementing the state interval index every time a message is received, for nondeterministic processes the state interval index is incremented every time a message is *sent*. With this modification, the model, the uniqueness theorem, and the algorithms all appear to extend to nondeterministic processes. Furthermore, our algorithms are capable of taking into account checkpoints in computing the maximum recoverable system state.

6 Causal Distributed Breakpoints

A *causal distributed breakpoint* is initiated by a sequential breakpoint in one process of a distributed computation, and restores each process in the computation to its earliest state that reflects all events that "happened before" the breakpoint. A causal distributed breakpoint is the natural extension for distributed programs of the conventional notion of a breakpoint in a sequential program. We have developed an algorithm for finding the causal distributed breakpoint given a sequential breakpoint in one of the processes. *Approximately consistent checkpoint sets* are used for efficiently restoring each process to its state in a causal distributed breakpoint.

References

[1] A. Borg, J. Baumbach, and S. Glazer, "A Message System Supporting Fault Tolerance", *Proceedings of the Ninth Symposium on Operating System Principles*, pp. 90-99 (October 1983).

[2] A. Borg, W. Blau, W. Graetsch, F. Herrman and W. Oberle, "Fault Tolerance under UNIX", ACM Transactions on Computer Systems, Vol. 7, No. 1, pp. 1-24 (February 1989).

[3] R. Bubenik and W. Zwaenepoel, "Performance of Optimistic Make" *Proceedings of the 1989 Sigmetrics and Performance Conference*, pp. 39-48 (May 1989).

[4] J.B. Carter and W. Zwaenepoel, "Optimistic Implementation of Bulk Data Transfer Protocols" *Proceedings of the 1989 Sigmetrics Conference*, pp. 61-69 (May 1989).

[5] K.M. Chandy and L. Lamport, "Distributed Snapshots: Determining Global States of Distributed Systems", ACM Transactions on Computer Systems, Vol. 3, No. 1, pp. 63-75 (February 1985).

[6] D.R. Cheriton and W. Zwaenepoel, "The Distributed V Kernel and its Performance for Diskless Workstations", *Proceedings of the 9th Symposium on Operating System Principles*, pp. 128-140 (October 1983).

[7] J. Fowler and W. Zwaenepoel, "Causal Distributed Breakpoints", to appear in *Proceedings of the Tenth International Conference on Distributed Computing Systems* (May 1990).

[8] D.B. Johnson, "Distributed System Fault Tolerance Using Message Logging and Checkpointing", Ph.D. Thesis, Rice University (December 1989).

[9] D.B. Johnson, P. Keleher and W. Zwaenepoel, "Finding the Maximum Recoverable System State in Optimistic Rollback Recovery Methods", submitted to *Symposium on Reliable Distributed Software* (May 1990).

[10] D.B. Johnson and W. Zwaenepoel, "Sender-Based Message Logging", *Proceedings of the 17th Fault Tolerant Computing Symposium*, pp. 14-19 (June 1987).

[11] D.B. Johnson and W. Zwaenepoel, "Recovery in Distributed Systems Using Optimistic Message Logging and Checkpointing", *Proceedings of the Seventh Principles of Distributed Computing Symposium*, pp. 171-181 (August 1988).

[12] D.B. Johnson and W. Zwaenepoel, "Recovery in Distributed Systems Using Message Logging and Checkpointing", to appear in *Journal of Algorithms* (September 1990).

[13] D.B. Johnson and W. Zwaenepoel, "Distributed Systems Fault Tolerance Using Sender-Based Message Logging", submitted to *Transactions on Computer Systems* (May 1990).

[14] D.B. Johnson and W. Zwaenepoel, "Transparent Optimistic Rollback Recovery", submitted to *European SigOps Workshop on Fault Tolerance* (May 1990).

[15] M.L. Powell and D.L. Presotto, "PUBLISHING: A Reliable Broadcast Communication Mechanism", *Proceedings of the 9th Symposium on Operating System Principles*, pp. 100-109 (October 1983).

[16] A.P. Sistla and J.L. Welch, "Efficient Distributed Recovery Using Message Logging", *Proceedings of the Eighth Principles of Distributed Computing Symposium* (August 1989).

[17] R.E. Strom, D.F. Bacon and S.A. Yemini, "Volatile Logging in n-Fault-Tolerant Distributed Systems", *Proceedings of the 18th Fault Tolerant Computing Symposium* (June 1988).

[18] R.E. Strom and S. Yemini, "Optimistic Recovery in Distributed Systems", ACM Transactions on Computer Systems, Vol. 3, No. 3, pp. 204-226 (August 1985).

INDEX TO PUBLICATIONS:

1. "Recovery in Distributed Systems Using Optimistic Message Logging and Checkpointing", by D.B. Johnson and W. Zwaenepoel, Proceedings of the Sixth Symposium on Principles of Distributed Computing, pp. 171-181 (August 1988)

2. "Causal Distributed Breakpoints", by J. Fowler and W. Zwaenepoel, to appear in Proceedings of the 10th International Conference on Distributed Computing Systems (May 1990).

3. "Recovery in Distributed Systems Using Message Logging and Checkpointing", by D.B. Johnson and W. Zwaenepoel, to appear in Journal of Algorithms (September 1990).

4. "Finding the Maximum Recoverable System State in Optimistic Rollback Recovery", by D.B. Johnson, P. Keleher and W. Zwaenepoel, submitted to the 9th Symposium on Reliable Distributed Software (October 1990).

5. "Transparent Optimistic Rollback Recovery", by D.B. Johnson and W. Zwaenepoel, submitted to the European SigOps workshop on fault tolerance (October 1990).

6. "Sender-Based Message Logging Fault Tolerance for Distributed Systems", by D.B. Johnson and W. Zwaenepoel, submitted to ACM Transactions on Computer Systems.