An Optimality Proof for Asynchronous Recovery Algorithms in Distributed Systems

Mukesh Singhal†
Dept. of Computer
and Information Science
The Ohio State University
Columbus, OH 43210
USA
{E-mail: singhal@cis.ohio-state.edu}

Friedemann Mattern
Dept. of Computer Science
University of Saarbrücken
Im Stadtwald 36
66123 Saarbrücken
Germany
{E-mail: mattern@cs.uni-sb.de}

May 17, 1995

Abstract

We prove the optimality of asynchronous recovery algorithms for distributed systems in the sense that irrespective of the order of roll backs by sites, all sites incrementally converge to a unique consistent cut which is the "latest" of all past consistent cuts formed by the local recovery points of the sites.

Key Words: Asynchronous recovery algorithms, distributed systems, consistent cut.

---

†This work was supported by the German Academic Exchange Program (DAAD), Bonn and the University of Saarbrücken, Germany.

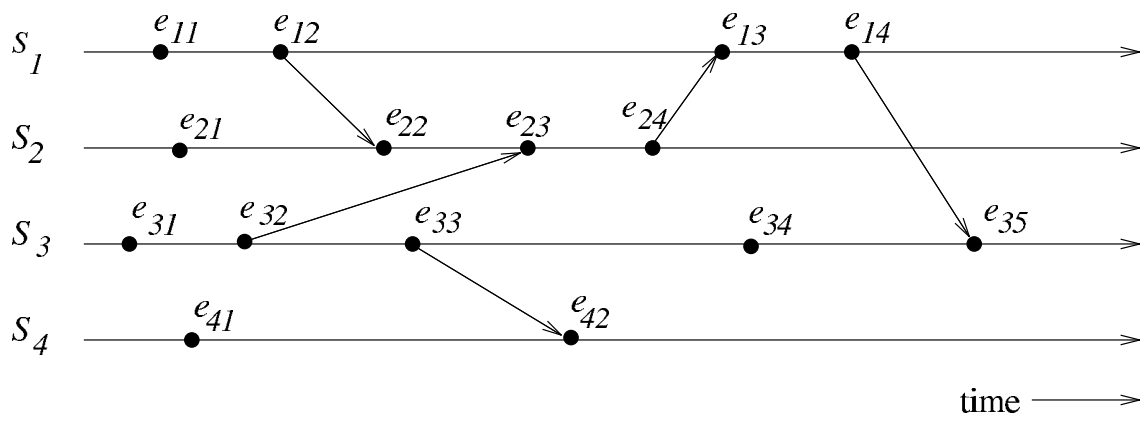

Figure 1: The space-time diagram of a distributed computation.

1 Problem Statement

A distributed system consists of a collection of geographically dispersed autonomous sites, say S_1, S_2, ..., S_n, which are connected by a communication network. The sites do not share any memory and communicate solely by message passing. Message propagation delay is finite but unpredictable. There is no common physical clock. We assume that the underlying communication medium is reliable and if a site crashes, it does so in a fail-stop manner.

In a computation on such a system, the actions performed by sites are modeled as *events*. Typically, three types of events are discerned: message *send* events, message *receive* events, and *internal* events. The local computation at site S_i is modeled as a sequence of events $E_i = e_{i1}, e_{i2}, e_{i3}, \ldots$. Figure 1 shows the space-time diagram of a distributed computation where events are depicted as dots and messages in the computation are depicted by arrows.

In distributed recovery algorithms with asynchronous checkpointing [1, 3, 7–10], sites periodically take checkpoints of their local states asynchronously (i.e., without any coordination with other sites). When a site fails, failure recovery is performed by restoring the system state to a globally consistent state of local checkpoints. Consistency is defined in terms of causality between the events on sites. The causality relation in a distributed computation [5], denoted by $\rightarrow$, is defined as the smallest transitive relation on the events such that (1) if e_{ij} occurs before e_{ik} on S_i (i.e., $j < k$), then $e_{ij} \rightarrow e_{ik}$ and (2) if e_{ix} is the send event of a message at site S_i and e_{jy} is the receive event of this message at S_j, then $e_{ix} \rightarrow e_{iy}$. Clearly, since messages don't go backwards in time, the causality relation is irreflexive and asymmetric, and thus is a strict partial order on the events of a distributed computation.

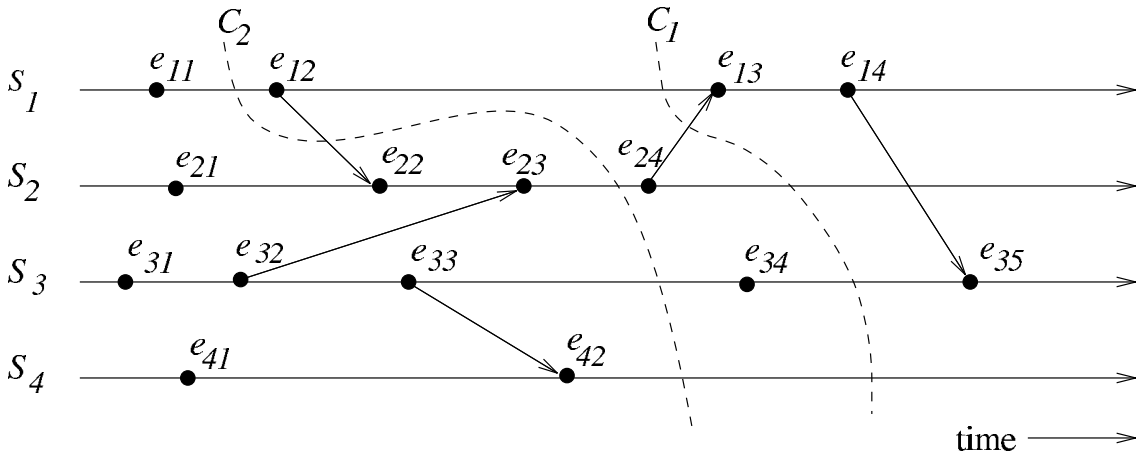

Figure 2: Consistent and inconsistent states in a distributed computation.

For example, in Figure 1, events e_{11} and e_{23} are causally related. So are events e_{21} and e_{35}, but not events e_{21} and e_{34}.

A global state of the system consists of a set of local states of sites, one state for each site. A global state is *consistent* provided the following holds: if the receive event of a message has been recorded in the state of a site, then its send event is also recorded in the state of the sender site. In space-time diagrams, a global state can be depicted by a cut, a zigzag line cutting the space-time diagram transversely (e.g., C_1 and C_2 in Figure 2). For a site, a cut denotes the site's state where the cut crosses the site's line. In Figure 2, the set of local states defined by cut C_1 represents a consistent system state, while the set of local states defined by cut C_2 does not represent a consistent system state because the state of S_2 in cut C_2 contains the message receive event e_{22}, but the state of S_1 in the cut does not contain the corresponding message send event e_{12}. Note that the local states of sites S_1 and S_2 are causally related in the global system state represented by cut C_2. The local states of sites must not be causally related in a consistent global state. Events at a site cause state transitions at that site. Therefore, the definition of causality can be extended to include site states in the following way: $e \rightarrow s$ iff event e causes a site to make a transition to state s, and $s \rightarrow e$ iff event e occurs in state s at a site.

In asynchronous recovery algorithms, at the time of recovery, a consistent state is obtained by rolling back sites to appropriate local checkpoints. Sites communicate with other sites to determine if their local states are causally related. If they are, sites that received messages which are responsible for the causal dependencies, roll back to eliminate these causal

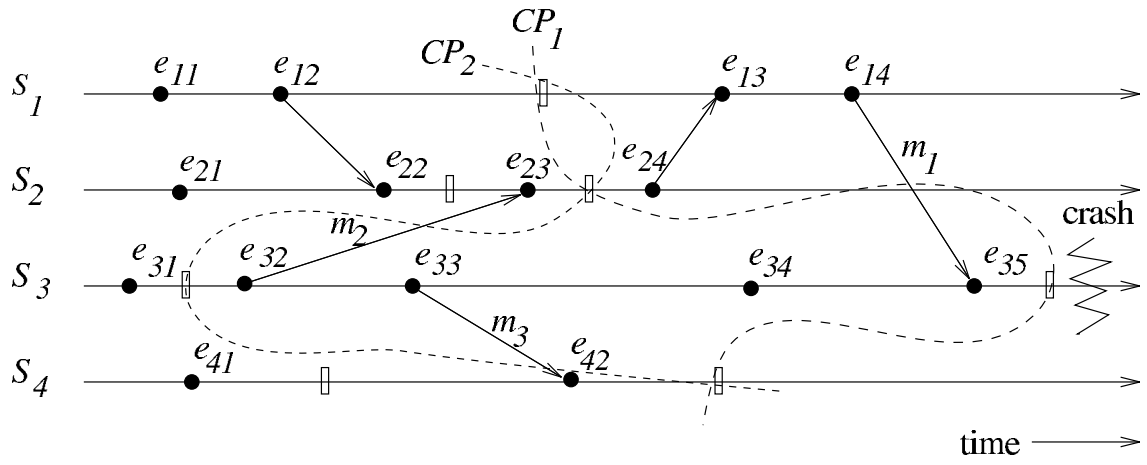

Figure 3: Asynchronous checkpoints and recovery to a consistent state.

dependencies. This process is repeated until the local states of all the sites are free from causal dependencies. For example, Figure 3 illustrates the operation of an asynchronous recovery algorithm. Local checkpoints at sites are denoted by small rectangles. When sites S_3 crashes, all sites roll back to their last local checkpoints (denoted by cut CP_1). After having exchanged the information regarding their current checkpoints, sites S_1 and S_3 discover that S_3's state reflects the receipt of message m_1 but S_1's state does not reflect its send. To fix the problem, S_3 rolls back to its earlier local checkpoint yielding the global checkpoint CP_2. However, this checkpoint is inconsistent due to messages m_2 and m_3. To fix this problem, S_2 and S_4 roll back to their earlier local checkpoints and the resulting system state is consistent. We observe that rolling back of a site to eliminate a causal dependency may introduce further causal dependencies, inducing roll backs of other sites. This is called the *cascaded roll-backs* [6]. Note that in asynchronous recovery algorithms, generally a site cannot roll back to any state – a site can roll back only to its checkpoints. However, finer granularity of roll backs can be achieved by logging of messages [3,7].

Due to unpredictable message delays and disparity in processors' speeds, the order in which sites will perform roll backs is nondeterministic. Intuitively, the consistent checkpoint to which sites converge depends upon the order of roll backs by sites and thus, sites' roll back may not be optimal. Informally, a roll back by sites is optimal if each site performs least possible roll back to restore the system to a consistent global state.

In this paper, we define the optimal roll back by sites in terms of a suitable definition of the maximal consistent cut in a distributed computation. We also prove that asynchronous

recovery algorithms are optimal in the sense that irrespective of the order of roll back by sites, all sites always converge to the maximal consistent cut. Previously, optimality arguments have been given for specific recovery algorithms only (e.g., [2,3,4,10]). We show that an asynchronous recovery algorithm that uses a general method of iteratively rolling back sites to eliminate causal dependencies always converges to the maximal consistent cut.

2 Maximal Consistent Cut

In this section, we discuss the notion of maximal consistent cut by mapping global states in a distributed computation to sets of events and using the set inclusion relation to model causality. Johnson and Zwaenepoel [3] demonstrated the existence of such a unique maximal cut by showing that the global states in a distributed computation form a lattice.

2.1 Consistent Cuts

A cut in a distributed computation represents a local state of every site in the system. Notationally, a cut C is a set $\{c_1, c_2, ..., c_n\}$, where c_i is the local state of site S_i which is also called the *cut event* of S_i. Graphically, a cut is a transverse line through the space-time diagram that cuts every site at its cut event.

The cut event or the local state of a site in a cut is modeled as a set of events on that site. Cut event c_i at site S_i consists of all events on S_i that causally precede the state represented by c_i. Thus, a cut $C = \{c_1, c_2, ..., c_n\}$ divides all the events in a distributed computation in two sets, $past(C)$ and $future(C)$. The former consists of all events in $c_1 \cup c_2 \cup ... \cup c_n$ and the latter consists of the remaining events.

The following definitions and assertion are due to [5].

Definition: A cut C_1 is *later* than a cut C_2 iff $C_2 \subseteq C_1$.

In Figure 2, cut C_1 is later than C_2 and in Figure 3, the cut denoted by CP_1 is later than the cut denoted by CP_2.

Definition: (Consistent Cut) A cut C in a distributed computation is *consistent* if

$$\forall\, e \in C,\ \exists e' ::\ e' \rightarrow e \Rightarrow e' \in C.$$

Assertion 1: If C_1, C_2,..., C_n are consistent cuts in a distributed computation, then $C_1 \cup C_2 \cup$ ... $\cup C_n$ is also a consistent cut in the computation.

Obviously, the cut $C_1 \cup C_2 \cup$ ... $\cup C_n$ is later than each of the cuts C_1, C_2,..., C_n.

A cut defines a global state of the system in which the local state of a site corresponds to the cut event of that site and all message arrows cutting the cut correspond to the messages in transit.

2.2 Maximal Consistent Cut

A cut C is a maximal consistent cut among the set of all consistent cuts through a distributed computation (the set is denoted by C_s) iff all other consistent cuts in C_s lie in the $past(C)$; that is, the maximal consistent cut is later than all other consistent cuts. Using the set union operation, the maximal consistent cut for the distributed computation is given by $\bigcup_{c \in C_s} c$.

Definition: (Maximal Consistent Cut) Cut $C = \{c_1, c_2, ..., c_n\}$ is a maximal consistent cut in a distributed computation if there is no cut event in $future(C)$ that lies on a consistent cut.

It should be noted, however, that when we talk about consistent cuts in connection with asynchronous recovery algorithms, cut events are generally the local checkpoints taken by sites. We cannot have a consistent cut passing through arbitrary states of sites (unless message logging is done to enable reconstruction of a site's state at a finer granularity).

Assertion 2: There always exists a maximal consistent cut in the local states recorded by sites in an asynchronous recovery algorithm.

Proof: All sites start in a consistent state. So the initial states recorded by all sites form a consistent state. If the computation has progressed and other consistent cuts exist in the distributed computation, then the maximal consistent cut at any time is defined by the union of all those consistent cuts. □

3 Proof of Optimality

In asynchronous recovery algorithms, a site performs recovery by iteratively rolling back to an earlier local checkpoint to eliminate its causal dependencies with other sites. The recovery is complete when the local state of every site is free from causal dependencies with all other sites.

It is clear that for optimality, an asynchronous recovery algorithm should cause sites to roll back only up to the unique maximal consistent cut with respect to the "current" computation. Intuitively, the consistent cut to which sites converge depends upon the order of roll backs by sites, and due to unpredictable message delays and disparity in the speed of processors, the order in which sites will perform roll backs is nondeterministic. We next show that all sites, nonetheless, always converge to the maximal consistent cut.

Theorem: Irrespective of the order in which roll backs are performed by various sites in an asynchronous recovery algorithm, the sites always converge to the maximal consistent cut.

Proof: From Assertion 2, a maximal consistent cut exists in the local states (i.e., checkpoints) recorded by all sites. Let the maximal consistent cut be denoted by $\{c_1, c_2, ..., c_n\}$.

To prove the theorem, we need to prove the following two assertions:

1. No site has to roll back beyond its cut event on the maximal consistent cut.

2. Each site will roll back at least up to its cut event on the maximal consistent cut.

The proof of the latter is straightforward. Since there is no consistent cut in the future of the maximal consistent cut in a distributed computation, all sites must roll back at least up to the maximal consistent cut to reach a consistent global state.

The proof of the former is more involved. Unpredictable order of roll backs by sites will affect the order in which sites will reach their cut events on the final consistent cut or will roll back beyond the maximal consistent cut (if at all they have to roll back beyond the maximal consistent cut). This is tantamount to assuming, in the proof, a generalized order

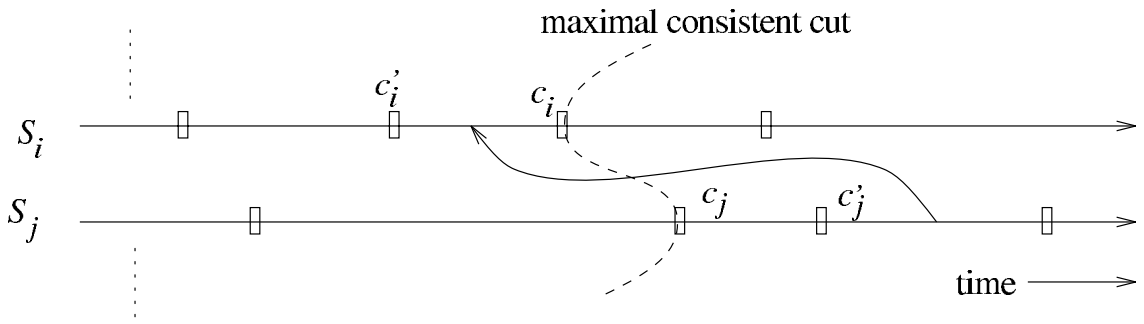

Figure 4: A scenario for Case 1.

in which sites reach their local states on the final consistent cut or roll back beyond the maximal consistent cut.

Assume that S_i is the first site that is asked by a site, say S_j, to roll back beyond its local state on the maximal consistent cut. We have the following two cases:

Case 1: Site S_i was in the state corresponding to its cut event c_i on the maximal consistent cut when S_j caused it to roll back further.

Case 2: Site S_i had not yet reached the state corresponding to its cut event c_i on the maximal consistent cut when S_j caused it to roll back beyond c_i.

Let us first consider Case 1 and show that this will never occur. The scenario of Case 1 is shown in Figure 4, where when site S_j rolls back to state c_j' (or beyond), it asks site S_i to roll back from state c_i to state c_i'. This implies that c_j' and c_i are causally related; a message is sent after cut event c_j' by S_j that was received by S_i before cut event c_i but after c_i'. That is,

$$c_j' \rightarrow c_i$$
$$\Rightarrow c_j \rightarrow c_i \qquad \text{(because } c_j \rightarrow c_j'\text{)}$$

This is a contradiction to our premise that c_i and c_j lie on a consistent cut. Therefore, when a site is in the state corresponding to its cut event c_i on the maximal consistent cut, it will not be asked to roll back further.

Let us now show that Case 2 will never occur. The corresponding scenario is shown in Figure 5. When site S_j rolls back to state c_j', it asks site S_i to roll back from state c_i'' to state c_i'. Since S_j is asking S_i to roll back from cut event c_i'' to beyond cut event c_i in order

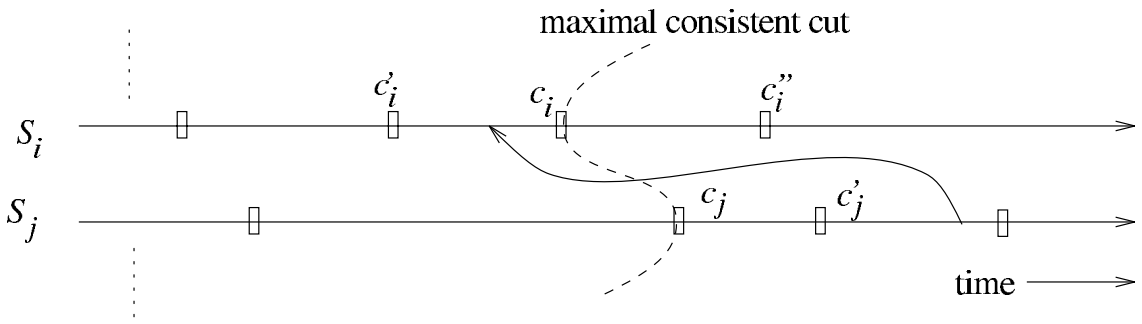

Figure 5: A scenario for Case 2.

to obtain a consistent cut, c_j' and c_i are causally related (that is, a message is sent after cut event c_j' by S_j that was received by S_i before cut event c_i). That is, $c_j' \rightarrow c_i$. Since $c_j \rightarrow c_j'$, this implies $c_j \rightarrow c_i$ which is again a contradiction!

Both cases also hold for $c_j' = c_j$. Therefore, no site ever rolls back beyond its cut event on the maximal consistent cut. □

4 Summary

We first defined the optimal roll back by sites in asynchronous recovery algorithms in terms of a suitable definition of the maximal consistent cut in a distributed computation. We then proved the optimality of asynchronous recovery algorithms in the sense that irrespective of the order of roll back by sites, all sites converge to this unique maximal consistent cut.

Acknowledgements: The authors are indebted to anonymous referees for valuable comments on an earlier version of the paper.

References

1. B. Bhargava and S.R. Lian, Independent Checkpointing and Concurrent Rollback Recovery – An Optimistic Approach, Proc. of IEEE Symp. on Reliable Distributed Systems, 1988, pp. 3-12.

2. E. Elnozahy and W. Zwaenepoel, Manetho: Transparent Rollback-Recovery with Low Overhead, Limited Rollback, and Fast Output Commit, IEEE Trans. on Computers, Vol 41, No 5, pp. 526-531, May 1992.

3. D. Johnson and W. Zwaenepoel, Recovery in Distributed Systems Using Optimistic Message Logging and Checkpointing, Journal of Algorithms, Vol 11, No. 3, 1990, pp. 462-491.

4. R. Koo and S. Toueg, Checkpointing and Rollback Recovery in Distributed Systems, IEEE Trans. on Software Engineering, Vol SE-13, No 1, pp. 23-31, January 1987.

5. F. Mattern, Virtual Time and Global States of Distributed Systems, Parallel and Distributed Algorithms, M. Cosnard et al. (editors), North-Holland, 1989, pp. 215-226.

6. B. Randell, System Structure for Software Fault Tolerance, IEEE Trans. on Software Engineering, Vol 1, No. 2, 1975, pp. 220-232.

7. G. Richard and M. Singhal, Using Logging and Asynchronous Checkpointing to Implement Recoverable Distributed Shared Memory, Proc. of the 12th Symposium on Reliable Distributed Systems, October 1993.

8. A.P. Sistla and J.L. Welch, Efficient Distributed Recovery Using Message Logging, Proc. of the 8th ACM Symp. on PODC, August 1989, pp. 223-238.

9. R.E. Strom and S. Yemini, Optimistic Recovery in Distributed Systems, ACM Trans. on Computer Systems, Vol 3, No. 3, 1985, pp. 204-226.

10. S. Venkatesan and Tony Juang, Low-Overhead Optimistic Crash Recovery, Proc. of the 11th Intl. Conf. on Distributed Computing Systems, May 1991, pp. 454-461.

11. Y.-M. Wang and W. Fuchs, Optimistic Message Logging for Independent Checkpointing in Message-Passing Systems, Proc. of the 11th Symp. on Reliable Distributed Systems, pp. 147-154, October 1992.