April 1993

UILU-ENG-93-2213
CRHC-93-07

Center for Reliable and High-Performance Computing

OPTIMAL MESSAGE LOG RECLAMATION FOR INDEPENDENT CHECKPOINTING

Yi-Min Wang and W. Kent Fuchs

Coordinated Science Laboratory
College of Engineering
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) UILU-ENG-93-2213 CRHC 93-07	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Coordinated Science Lab University of Illinois 6b. OFFICE SYMBOL (*If applicable*) N/A	7a. NAME OF MONITORING ORGANIZATION National Aeronautics and Space Administration ICLASS and Office of Naval Research
6c. ADDRESS (*City, State, and ZIP Code*) 1308 W. Main St. Urbana, IL 61801	7b. ADDRESS (*City, State, and ZIP Code*) Moffett Field CA94035 Arlington VA 22217
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 7a 8b. OFFICE SYMBOL (*If applicable*)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (*City, State, and ZIP Code*) 7b	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO

11. TITLE (*Include Security Classification*)
Optimal message Log Reclamation for Independent Checkpointing

12. PERSONAL AUTHOR(S)
WANG, Yi-Min and W. Kent Fuchs

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (*Year, Month, Day*)	15. PAGE COUNT
Technical	FROM _____ TO _____	1993 April 14	19

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (*Continue on reverse if necessary and identify by block number*)
FIELD	GROUP	SUB-GROUP	fault tolerance, parralel and distributed systems, independent checkpointing,message logging, garbage collectio

19. ABSTRACT (*Continue on reverse if necessary and identify by block number*)

Independent (uncoordinated) checkpointing for parallel and distributed systems allows maximum process autonomy but suffers from possible domino effects and the associated storage space overhead for maintaining multiple checkpoints and message logs. In most research on checkpointing and recovery it has been assumed that only the checkpoints and message logs older than the global recovery line can be discarded. We show in this paper how recovery line transformation and decomposition can be applied to the problem of efficiently identifying all discardable message logs, thereby achieving optimal garbage collection. Communication trace-driven simulation for several parallel programs is used to show the benefits of the proposed algorithm for message log reclamation.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (*Include Area Code*) / 22c. OFFICE SYMBOL

Optimal Message Log Reclamation for Independent Checkpointing

Yi-Min Wang and W. Kent Fuchs

Primary contact: Yi-Min Wang

Center for Reliable and High-Performance Computing
Coordinated Science Laboratory
1308 West Main Street
University of Illinois
Urbana, IL 61801

E-mail: ymwang@crhc.uiuc.edu
Phone: (217) 244-7161
FAX: (217) 244-5686

Abstract

Independent (uncoordinated) checkpointing for parallel and distributed systems allows maximum process autonomy but suffers from possible domino effects and the associated storage space overhead for maintaining multiple checkpoints and message logs. In most research on checkpointing and recovery it has been assumed that only the checkpoints and message logs older than the global recovery line can be discarded. We show in this paper how recovery line transformation and decomposition can be applied to the problem of efficiently identifying all discardable message logs, thereby achieving optimal garbage collection. Communication trace-driven simulation for several parallel programs is used to show the benefits of the proposed algorithm for message log reclamation.

Key words: fault tolerance, parallel and distributed systems, independent checkpointing, message logging, garbage collection

[1] *Acknowledgement:* This research was supported in part by the National Aeronautics and Space Administration (NASA) under Grant NASA NAG 1-613, in cooperation with the Illinois Computer Laboratory for Aerospace Systems and Software (ICLASS), and in part by the Department of the Navy and managed by the Office of the Chief of Naval Research under Contract N00014-91-J-1283.

1 Introduction

Numerous checkpointing and rollback recovery techniques have been proposed in the literature for parallel and distributed systems. They can be classified into three primary categories. *Coordinated checkpointing* schemes [1–5] synchronize computation with checkpointing by coordinating processors during a checkpointing session in order to maintain a consistent set of checkpoints. Each processor only keeps the most recent successful checkpoint and rollback propagation is avoided at the cost of potentially significant performance degradation during normal execution. *Loosely-synchronized checkpointing* schemes [6–8] reduce the coordination overhead by taking advantage of loosely-synchronized checkpointing clocks and by bounding the message transmission delay. *Independent checkpointing* schemes [9–19] replace the checkpoint synchronization by dependency tracking and possibly message logging in order to allow maximum process autonomy. Rollback propagation is managed by searching for a consistent system state based on the dependency information. Process autonomy during normal execution is preserved by either allowing slower recovery or assuming a piecewise deterministic execution model [15]. Typically, each processor has to maintain multiple checkpoints and message logs to ensure successful recovery.

This paper considers independent checkpointing schemes for nondeterministic execution [10]. Most research on this subject has concentrated on algorithms for finding the latest consistent set of checkpoints, i.e., the *recovery line*, during rollback recovery. The same algorithms can be applied to the set of existing checkpoints during normal execution to determine the *global recovery line*[2]. All the checkpoints and message logs older than the global recovery line then become obsolete and can therefore be discarded. Based on the observation that some of the non-obsolete checkpoints can also be discarded, we previously derived the necessary and sufficient conditions for a checkpoint to be non-discardable [20]. Let N be the number of processors, it was shown that there exists a set of N recovery lines which contains all the checkpoints possibly useful for any future recovery. We will show in

[2]The global recovery line can be used for recovery when the entire system crashes. A local recovery line is used when a subset of processors needs to roll back [9].

this paper how to identify all discardable *message logs* in order to further reduce the space overhead[3] for systems with message logging in addition to checkpointing [12].

The outline of the paper is as follows. Section 2 describes the checkpointing and recovery protocol and the technique of recovery line transformation and decomposition. Section 3 derives the necessary and sufficient conditions for identifying all discardable message logs and the experimental evaluation is described in Section 4.

2 Checkpointing Protocol and Recovery Lines

2.1 Checkpointing and Recovery Protocol

The system model considered in this paper consists of a number of concurrent processes for which all process communication is through message passing. Processes are assumed to run on fail-stop processors [21] and each processor is considered as an individual *recovery unit* [13]. We do not assume deterministic execution or the existence of any mechanism for detecting and recording internal nondeterministic events [19, 22]. Consequently, if the sender of a message is rolled back, the corresponding message log will be invalid during reexecution, which means the receiver also has to be rolled back in order to undo the effect of the message.

During normal execution, the state of each processor is periodically saved as a *checkpoint* on stable storage. Let $CP_{i,k}$ denote the kth checkpoint of processor p_i with $k \geq 0$ and $0 \leq i \leq N-1$, where N is the number of processors. A *checkpoint interval* is defined to be the time between two consecutive checkpoints on the same processor and the interval between $CP_{i,k}$ and $CP_{i,k+1}$ is called the kth checkpoint interval. Each message is tagged with the current checkpoint ordinal number and the processor number of the sender. Each processor takes its checkpoint independently and updates the *direct dependency information*

[3] A simple *sufficient* condition based on *local* information exists for identifying some discardable messages before they are logged [12]. This paper considers the *necessary and sufficient* conditions based on *global* information for identifying all discardable logged messages.

table (or *input table* [10]) as follows: if at least one message from the mth checkpoint interval of processor p_j had been processed during the previous checkpoint interval, the pair (j, m) is added to the table entry for the new checkpoint.

A centralized garbage collection algorithm can be invoked by any processor periodically to reduce the space overhead. First, the dependency information for all existing checkpoints is collected to construct the *checkpoint graph* [9] (Fig. 1(b)). The *rollback propagation algorithm* [9] shown in Fig. 2 is executed on the checkpoint graph to determine the global recovery line according to the definition of consistency described later. All the checkpoints and message logs before the global recovery line then become *obsolete* and their space can therefore be reclaimed. The same procedure can also be invoked by any processor which initiates a rollback to determine the *local recovery line*. The only differences are each surviving processor takes an additional *virtual checkpoint* [10] so that the dependency information during the current checkpoint interval is also included in the checkpoint graph (called the *extended checkpoint graph* [9]), and each processor will roll back to the appropriate checkpoint when it is informed of the local recovery line.

Two situations need to be considered for checkpoint consistency. In Fig. 3(a), $CP_{i,k}$ and $CP_{j,m}$ are inconsistent because of the *orphan message* [8] M_a, or equivalently because $CP_{j,m}$ *happened before* [24] $CP_{i,k}$. In Fig. 3(b), the message M_b is an *in-transit message*, i.e., recorded as "sent but not yet received", with respect to the system state containing $CP_{i,k}$ and $CP_{j,m}$. It has been shown [1, 7] that checkpoints like $CP_{i,k}$ and $CP_{j,m}$ can be considered consistent if M_b is logged. Pessimistic (synchronous) message logging protocols [25–27] can ensure such a message is always properly recorded at the receiving end. This is also true for an optimistic logging protocol if the inclusion of a new checkpoint in the checkpoint graph is properly delayed based on the message logging progress [12]. As a result, we consider the situation in Fig. 3(b) as consistent.

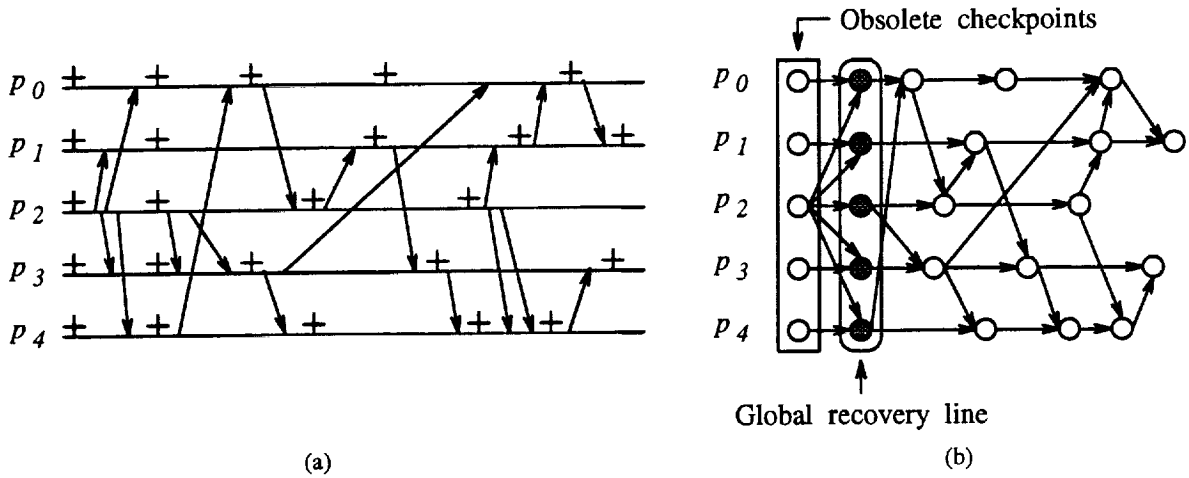


Figure 1: Example checkpoint graph (a) the checkpoint and communication pattern; (b) the corresponding checkpoint graph with each directed edge representing a *happened before* relation.

```
/* CP stands for checkpoint. Initially, all the CPs are unmarked */
include the latest CP of each processor in the root set;
mark all CPs strictly reachable [23] from any CP in the root set;
while (at least one CP in the root set is marked) {
    replace each marked CP in the root set by the latest unmarked CP on the
    same processor;
    mark all CPs strictly reachable from any CP in the root set;
}
the root set is the recovery line.
```

Figure 2: The rollback propagation algorithm.

Figure 3: Checkpoint consistency (a) orphan message M_a; (b) in-transit message M_b.

2.2 Recovery Line Transformation and Decomposition

We define a *global checkpoint* as a set of N checkpoints, one from each processor. Based on the previous description of checkpoint consistency, a *consistent global checkpoint* is a set of N checkpoints, one from each processor and no two of which are related through the *happened before* relation. A *recovery line* refers to the latest available consistent global checkpoint.

Note that being obsolete is simply a sufficient condition for being discardable. Our goal is to derive the necessary and sufficient conditions for identifying all discardable checkpoints and message logs. A checkpoint is non-discardable if and only if it can possibly belong to a future recovery line, and a message log is non-discardable if and only if it can possibly become an in-transit message with respect to a future recovery line. (For the ease of presentation, if a message M is an in-transit message with respect to a recovery line L, we will say M *intersects* L or the dependency edge corresponding to M *intersects* L.) The difficulty comes from the fact that there are an infinite number of possible future recovery lines. Therefore, our first step is to find a finite set of recovery lines, which suffices for the purpose of optimal garbage collection.

An *operational session* [10] is the interval between the start of normal execution and the instance of error recovery. Between two consecutive operational sessions is a *recovery session.* The entire program execution can be viewed as consisting of several operational sessions and recovery sessions. Within an operational session, new vertices are added to the checkpoint graph and can not have any outgoing edges to any existing vertices[4]. (If a graph G' can be obtained by adding new vertices to another graph G in this way, G' is called a *potential supergraph* of G.) Within a recovery session, existing vertices after the local recovery line are removed from the checkpoint graph. The above rules for checkpoint graph evolution then determine the possible future checkpoint graphs, and therefore the future recovery lines.

We first define a set of 2^N *immediate potential supergraphs* which are the supergraphs of G and the subgraphs of $\hat{G}$ as shown in Fig. 4. $\hat{G}$ is constructed by adding an *n-node* n_i

[4]Vertices with incoming edges from not-yet-collected vertices are temporarily excluded from the checkpoint graph.

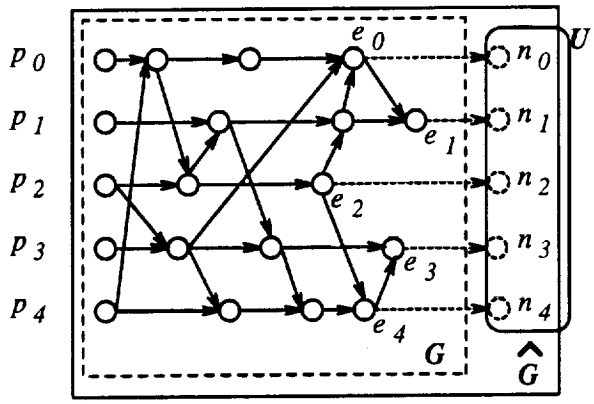


Figure 4: The immediate potential supergraphs.

with single incoming edge at the end for each process p_i. Let U denote the set of all such *n-nodes* and $\mathcal{RL}(G)$ denote the recovery line of a checkpoint graph G. The *recovery line transformation* procedure first transforms every possible future recovery line of G backwards in time into the recovery line of one of G's 2^N immediate potential supergraphs. The *recovery line decomposition* procedure then further reduces this set of 2^N recovery lines $\{\mathcal{RL}(\hat{G}-W) : W \subseteq U\}$ to the set of N recovery lines $\{\mathcal{RL}(\hat{G} - n_i) : n_i \in U\}$. We will describe the transformation and decomposition procedures by using the example in Fig. 5. Formal proofs can be found in [20].

Suppose G in Fig. 5(a) is the current checkpoint graph considered for garbage collection. Fig. 5(b) shows the extended checkpoint graph when p_3 later initiates the first rollback and G_c is the checkpoint graph immediately after the recovery. Fig. 5(d) shows another possible extended checkpoint graph when p_0 initiates a second rollback. We now describe how to transform and decompose $\mathcal{RL}(G_d)$, a typical future recovery line of G.

Transformation within an operational session: First we consider G_c and G_d where G_c is the starting checkpoint graph of a new operational session and G_d is a potential supergraph of G_c. For checkpoints X, Y and Z which belong to $\mathcal{RL}(G_d)$ but are not in graph G_c, we replace them by their corresponding *n-nodes* P, Q and R for G_c as shown in Fig. 5(g). $\mathcal{RL}(G_d) = \{A, B, X, Y, Z\}$ is then transformed into $\mathcal{RL}(G_g) = \{A, B, P, Q, R\}$ where G_g is

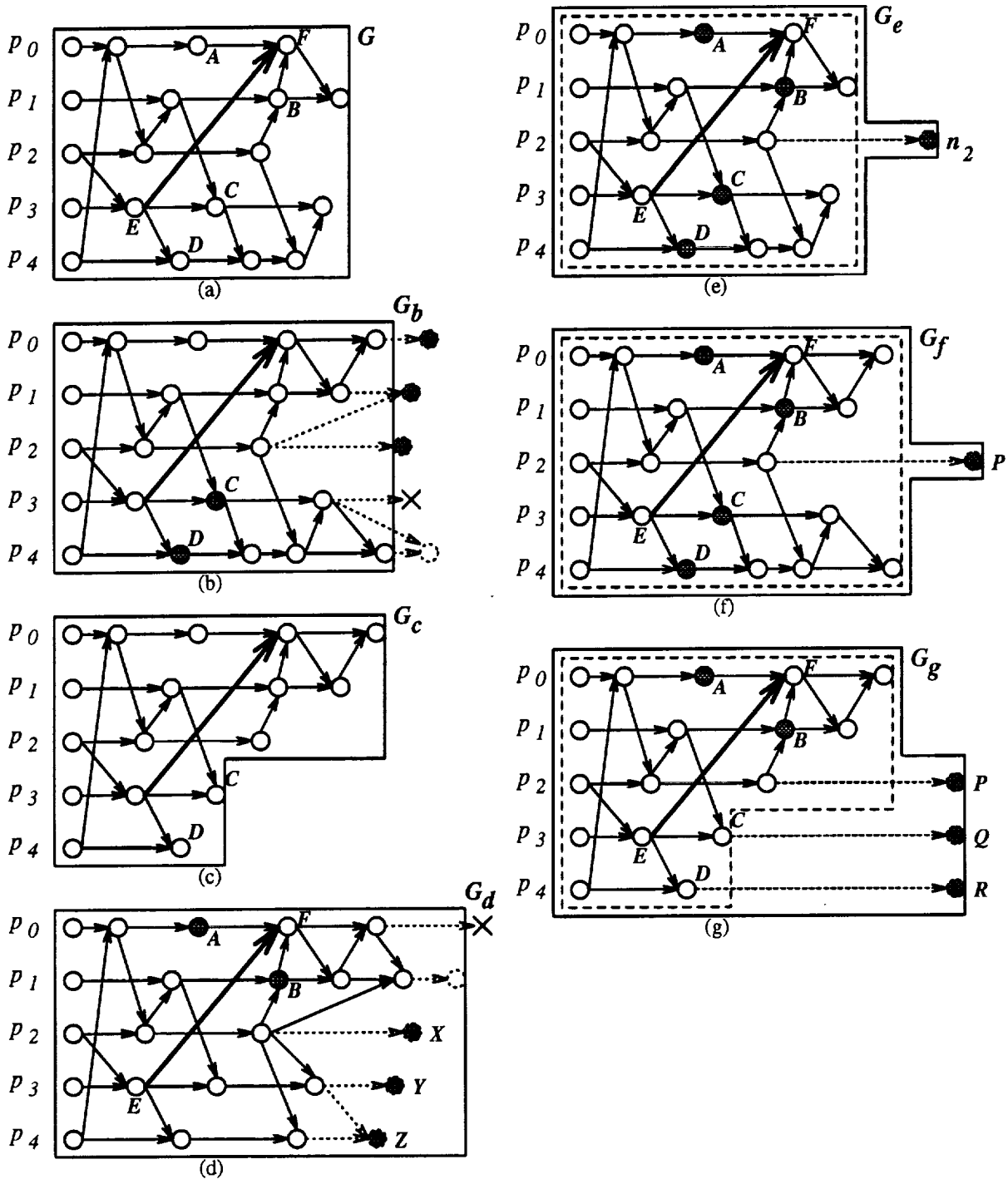


Figure 5: Example recovery line transformation.

an immediate potential supergraph of G_c.

Transformation across consecutive operational sessions: Now we consider G_g and G_b, the last checkpoint graph of the first operational session. Of the three *n-nodes* P, Q and R in $\mathcal{RL}(G_g)$, only Q and R come from the processors which were rolled back during the first recovery. We replace them by C and D, the corresponding checkpoints which were on the local recovery line. $\mathcal{RL}(G_g)$ is then transformed into $\mathcal{RL}(G_f) = \{A, B, P, C, D\}$. Notice that G_f is an immediate potential supergraph of G_b and is therefore a potential supergraph of G. By repeatedly and alternately applying the above two transformation procedures, every future recovery line can be transformed into another recovery line in the following set: $\{\mathcal{RL}(\hat{G} - W) : W \subseteq U\}$.

Recovery line decomposition: Let $min(S)$ denote the set of minimal elements, i.e., vertices with no incoming edges, of S. By utilizing the *lattice* properties of the maximum-sized antichains on a partially ordered set [24, 28], each of the 2^N recovery lines can be decomposed as:

$$\mathcal{RL}(\hat{G} - W) = min(\bigcup_{n_i \in W} \mathcal{RL}(\hat{G} - n_i)). \tag{1}$$

For example, the recovery line of $G_e = \hat{G} - \{n_0, n_1, n_3, n_4\}$ in Fig. 5(e) has the following decomposition (refer to Fig. 6)

$$\begin{aligned} \mathcal{RL}(G_e) &= min(\mathcal{RL}(\hat{G} - n_0) \cup \mathcal{RL}(\hat{G} - n_1) \cup \mathcal{RL}(\hat{G} - n_3) \cup \mathcal{RL}(\hat{G} - n_4)) \\ &= min(\{A, B, n_2, n_3, n_4, n_0, I, n_1, J, C, D\}) = \{A, B, n_2, C, D\}. \end{aligned}$$

3 Message Log Reclamation

By using the techniques described in the previous section, it has been shown that the set of all non-discardable checkpoints is equal to the union of the N recovery lines $\mathcal{RL}(\hat{G} - n_i)$, $n_i \in U$ (except for the n_i's) [20]. For the example shown in Fig. 6, while all the checkpoints in

G are non-obsolete, only those checkpoints corresponding to the shaded vertices in Fig. 6(f) are non-discardable.

In addition to the checkpoints, message logs[5] constitute another storage space overhead [12]. By following the transformation and decomposition procedures, we will show in the following that a message log is non-discardable, i.e., can possibly *intersect* a future recovery line, if and only if it *intersects* one of $\mathcal{RL}(\hat{G} - n_i)$'s.

3.1 Recovery Line Transformation and Decomposition

Instead of considering each individual message, we use its corresponding edge in the checkpoint graph for our discussion. Let (a, b) represent the directed edge starting at vertex a and pointing to vertex b. Clearly, (a, b) *intersects* a recovery line $\mathcal{RL}(G)$ if a is on the left hand side of $\mathcal{RL}(G)$ and b is on the right hand side of $\mathcal{RL}(G)$.

LEMMA 1 *If (a, b) can possibly intersect a future recovery line, (a, b) must intersect $\mathcal{RL}(\hat{G} - W)$ for some $W \subseteq U$.*

Sketch of the proof. Again, we use the example in Fig. 5. The edge (E, F) in G can *intersect* a possible future recovery line $\mathcal{RL}(G_d)$. We will show that (E, F) must also *intersect* $\mathcal{RL}(G_e)$.

Transformation within an operational session: First consider G_c, $\mathcal{RL}(G_g)$ and $\mathcal{RL}(G_d)$. Any vertex of G_c which is on the left (right) hand side of $\mathcal{RL}(G_d)$ must remain on the left (right) hand side of $\mathcal{RL}(G_g)$. Therefore, any edge of G_c *intersecting* $\mathcal{RL}(G_d)$, for example (E, F), must also *intersect* $\mathcal{RL}(G_g)$ after the recovery line transformation.

Transformation across consecutive operational sessions: Now consider G_c, $\mathcal{RL}(G_g)$ and $\mathcal{RL}(G_f)$. All vertices of G_c which are on the right hand side of $\mathcal{RL}(G_g)$ must remain on the right hand side of $\mathcal{RL}(G_f)$ because the transformation can only push the recovery line to the left. Those on the left hand side of $\mathcal{RL}(G_g)$ remain on the left hand side of $\mathcal{RL}(G_f)$

[5]The message logs considered in this paper are used for recording the state of the channels [1] instead of replaying for deterministic state reconstruction [13].

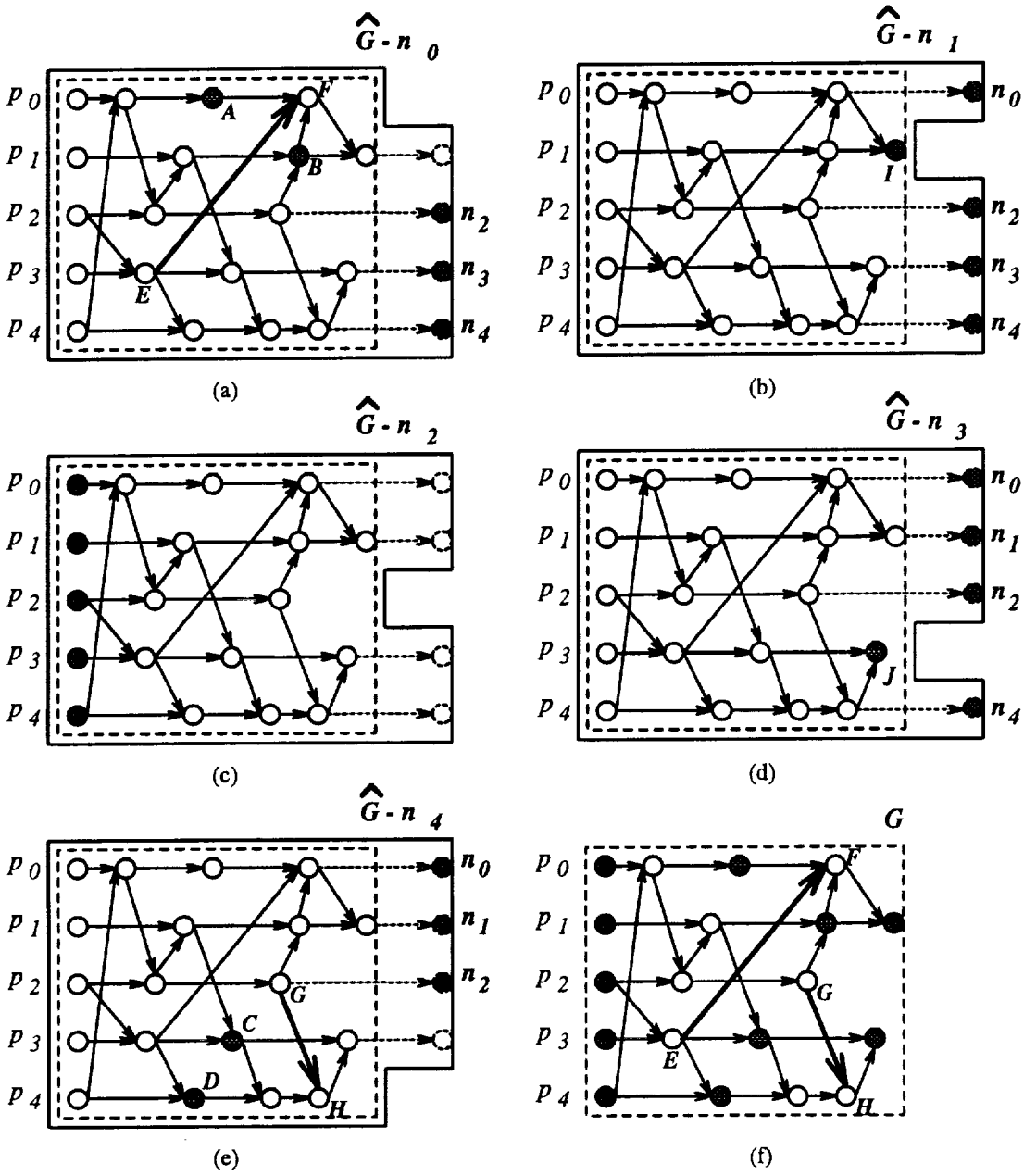


Figure 6: Example execution of our algorithm.

except for C and D. However, C and D can not have any outgoing edges in G_c because they were part of the local recovery line and therefore all such edges must have been removed during the recovery. Hence, any edge of G_c *intersecting* $\mathcal{RL}(G_g)$, for example (E, F), must also *intersect* $\mathcal{RL}(G_f)$ after the transformation.

Finally, we can show that (E, F) also *intersects* $\mathcal{RL}(G_e)$ by again applying the transformation within an operational session. □

LEMMA 2 *$min(\bigcup_{n_i \in W} \mathcal{RL}(\hat{G} - n_i))$ in Eq. (1) is equivalent to the set of the N leftmost checkpoints, one from each processor, among the checkpoints in the union.*

Proof. If a checkpoint v of p_i is not the leftmost checkpoint of p_i in the union, then v can not be a minimal element because there exists at least one checkpoint on its left. Conversely, if v is the leftmost checkpoint of p_i, v must be in $min(\bigcup_{n_i \in W} \mathcal{RL}(\hat{G} - n_i))$ because there are only N such checkpoints and $\mathcal{RL}(\hat{G} - W) = min(\bigcup_{n_i \in W} \mathcal{RL}(\hat{G} - n_i))$ must consist of N checkpoints. □

LEMMA 3 *If (a, b) intersects $\mathcal{RL}(\hat{G} - W)$ for some $W \subseteq U$, (a, b) must intersect $\mathcal{RL}(\hat{G} - n_i)$ for some $n_i \in U$.*

Proof. Suppose (a, b) does not *intersect* any of the N recovery lines $\mathcal{RL}(\hat{G} - n_i)$, $n_i \in U$. Then each of the N recovery lines must lie either entirely on the right hand side of (a, b) or entirely on the left hand side of it.

Recovery line decomposition: Given any of the 2^N recovery lines $\mathcal{RL}(\hat{G} - W)$, $W \subseteq U$, if all $\mathcal{RL}(\hat{G} - n_i)$'s, $n_i \in W$, are entirely on the right hand side of (a, b), $\mathcal{RL}(\hat{G} - W)$ must also lie on the right hand side of (a, b) by Eq. (1) and Lemma 2; if at least one $\mathcal{RL}(\hat{G} - n_i)$, $n_i \in W$, lies entirely on the left hand side of (a, b), $\mathcal{RL}(\hat{G} - W)$ will be on the left hand side of (a, b) again by Lemma 2. Therefore, we have shown that (a, b) can not *intersect* any $\mathcal{RL}(\hat{G} - W)$ if it does not *intersect* any $\mathcal{RL}(\hat{G} - n_i)$. Conversely, if (a, b) *intersects* $\mathcal{RL}(\hat{G} - W)$ for some $W \subseteq U$, (a, b) must *intersect* $\mathcal{RL}(\hat{G} - n_i)$ for some $n_i \in U$. □

3.2 The Algorithm

We now state the necessary and sufficient conditions for a message log to be non-discardable.

THEOREM 1 *A message log is non-discardable if and only if its corresponding edge in the checkpoint graph intersects* $\mathcal{RL}(\hat{G} - n_i)$ *for some* $n_i \in U$.

Proof. The *only if* part follows immediately from Lemmas 1 and 3. The *if* part comes from the fact that every $\mathcal{RL}(\hat{G} - n_i)$ is also a possible future recovery line. □

Theorem 1 also gives the algorithm for finding all non-discardable message logs: first compute the N recovery lines $\mathcal{RL}(\hat{G} - n_i)$, $n_i \in U$; only those message logs with their corresponding edges *intersecting* any of the N recovery lines are non-discardable. In Fig. 6, the edge (E, F) *intersects* $\mathcal{RL}(\hat{G} - n_0)$, (G, H) *intersects* $\mathcal{RL}(\hat{G} - n_4)$ and none of the edges *intersects* $\mathcal{RL}(\hat{G} - n_1)$, $\mathcal{RL}(\hat{G} - n_2)$ or $\mathcal{RL}(\hat{G} - n_3)$. Therefore, although all the edges in Fig. 6(f) are non-obsolete, only those message logs corresponding to (E, F) and (G, H) need to be retained.

There is an interesting difference between checkpoint reclamation and message log reclamation. While the set of non-discardable checkpoints is determined by the *union* of the N recovery lines $\mathcal{RL}(\hat{G} - n_i)$, $n_i \in U$, the set of non-discardable message logs is affected by the position of each *individual* recovery line. Fig. 7 illustrates such a difference. The non-discardable checkpoints a, b, c and d in Fig. 7(a) remain non-discardable in Fig. 7(b) when e is added to the graph. However, the non-discardable message logs corresponding to the edges (b, d) and (c, d) in Fig. 7(a) become discardable as the addition of e changes the positions of $\mathcal{RL}(\hat{G} - n_1)$ and $\mathcal{RL}(\hat{G} - n_2)$.

4 Experimental Results

Three hypercube programs are used to illustrate the message log reclamation capabilities and benefits of our algorithm. They are Cell placement, Channel router and QR decomposition, running on an 8-node Intel iPSC/2 hypercube. Communication traces are collected by

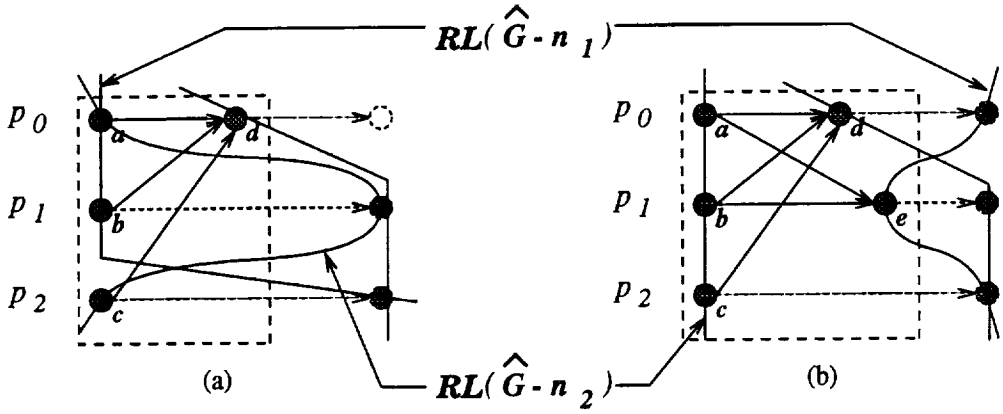


Figure 7: The difference between the reclamation of checkpoints and message logs.

intercepting the "send" and "receive" system calls. Communication trace-driven simulation is then performed to obtain the results. The execution time for each program is listed in Table 1. The checkpoint interval is arbitrarily chosen to be approximately one tenth of the execution time.

Table 1: Execution time and checkpoint interval.

Programs	Cell placement	Channel router	QR decomposition
Execution time (sec)	324	469	370
Checkpoint interval (sec)	35	40	35

Figs. 8-10 compare our algorithm with the traditional garbage collection algorithm for the three programs in terms of the number and size of the retained message logs. Each curve shows the remaining space overhead after garbage collection if the algorithm is invoked after a certain number of checkpoints have been taken. Since the checkpointing clocks on all nodes are approximately synchronized, checkpoints #$8n$ through #$8(n+1)$-1 are taken at about the same time, which explains the fact that the number of messages is almost constant within that interval.

The domino effect is illustrated by the constant increase in the number of non-obsolete message logs as the total number of checkpoints increases, for example, between checkpoints

#40 and #64 in Fig. 8(a) and between checkpoints #48 and #88 in Fig. 9(a). The figures show that our algorithm performs consistently better than the traditional algorithm and is particularly effective when the domino effect persists.

5 Concluding Remarks

We have shown that some of the non-obsolete message logs in an independent checkpointing protocol can be discarded because they can never be useful for any possible future recovery. An algorithm was developed for finding all discardable message logs in order to minimize the space overhead. Communication trace-driven simulation results for three hypercube programs showed that the algorithm can be effective in reducing the message log space overhead for real applications.

Acknowledgement

The authors wish to express their sincere thanks to Pi-Yu Chung for her valuable discussions, to Junsheng Long for his help with the experimental results and to Prith Banerjee for his hypercube programs.

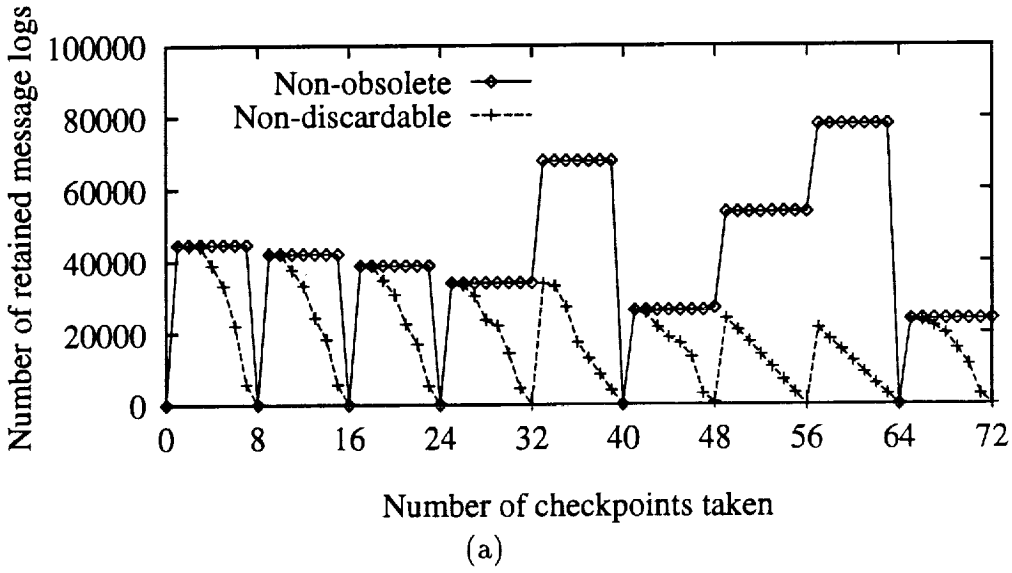
References

[1] K. M. Chandy and L. Lamport, "Distributed snapshots: Determining global states of distributed systems," *ACM Trans. on Computer Systems*, vol. 3, pp. 63–75, Feb. 1985.

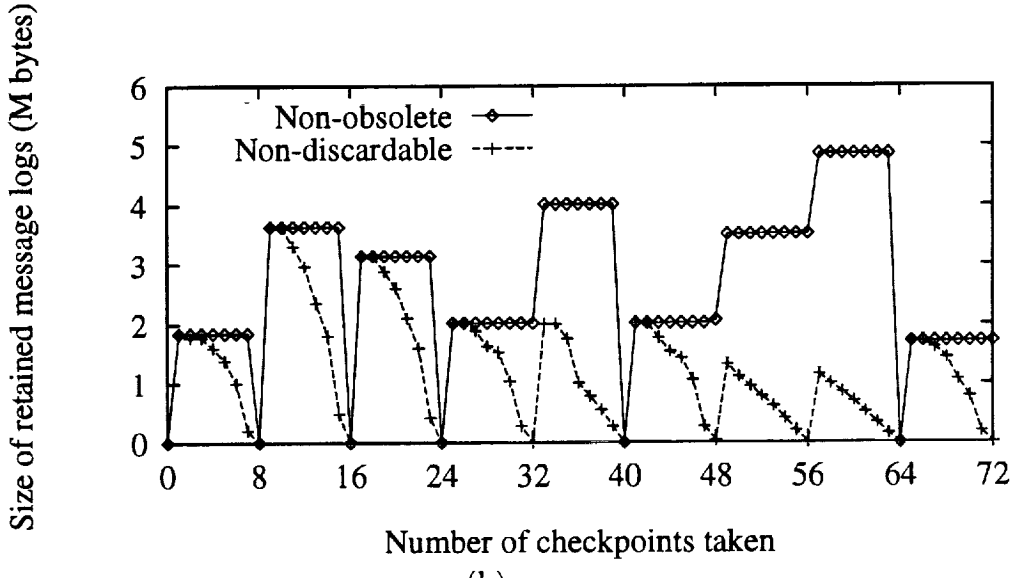
[2] T. H. Lai and T. H. Yang, "On distributed snapshots," *Information Processing Letters*, vol. 25, pp. 153–158, May 1987.

[3] R. Koo and S. Toueg, "Checkpointing and rollback-recovery for distributed systems," *IEEE Trans. on Software Engineering*, vol. SE-13, pp. 23–31, Jan. 1987.

[4] K. Li, J. F. Naughton, and J. S. Plank, "Checkpointing multicomputer applications," in *Proc. IEEE Symp. on Reliable Distr. Syst.*, pp. 2–11, 1991.

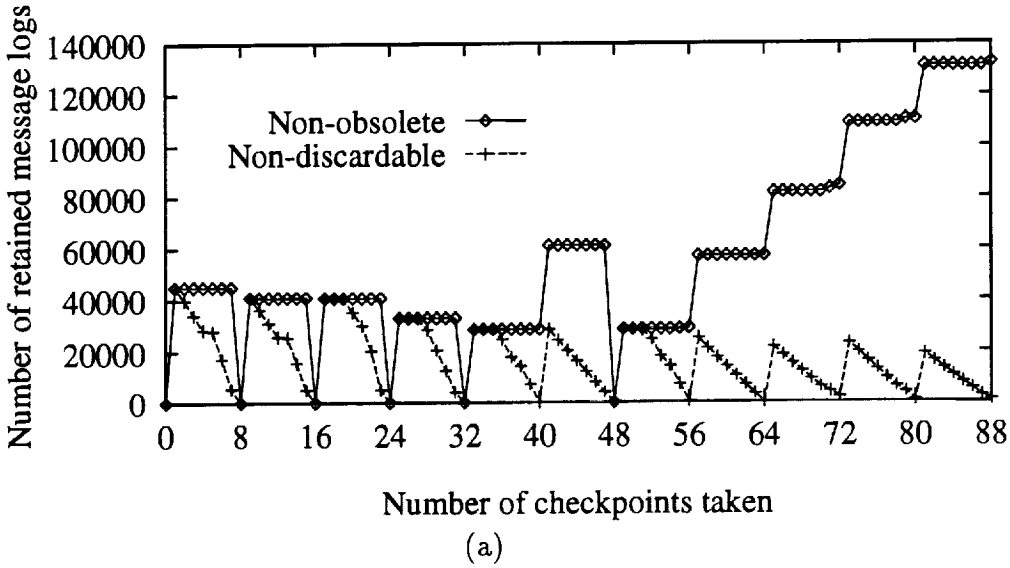


(a)

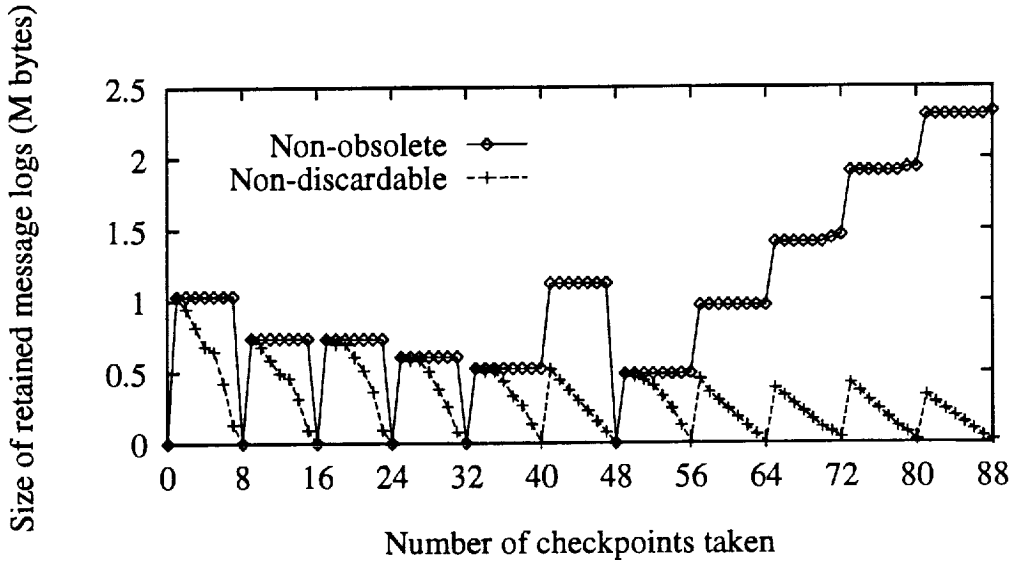


(b)

Figure 8: Message log reclamation for the Cell placement program.

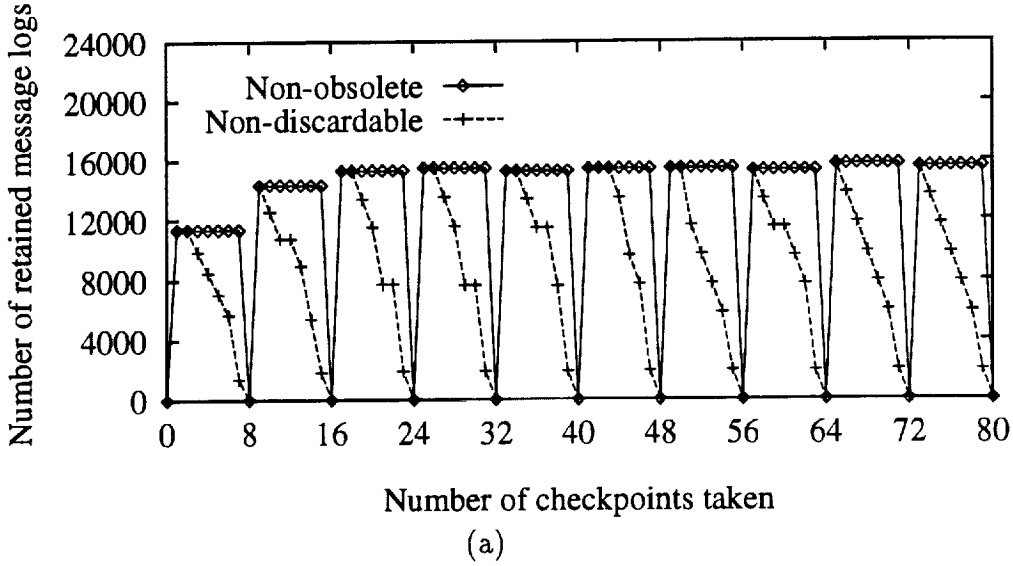


(a)

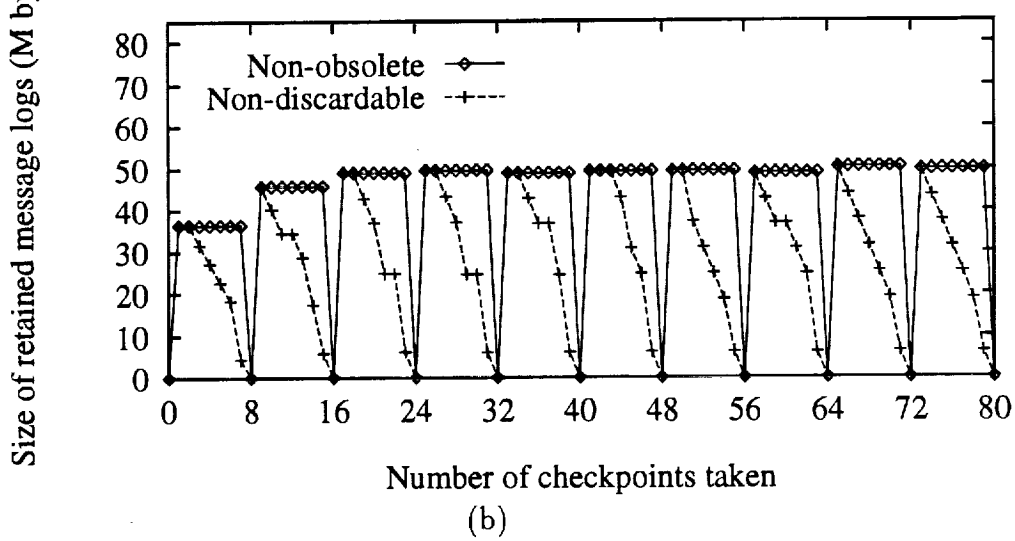


(b)

Figure 9: Message log reclamation for the Channel router program.



(a)



(b)

Figure 10: Message log reclamation for the QR decomposition program.

[5] E. N. Elnozahy, D. B. Johnson, and W. Zwaenepoel, "The performance of consistent checkpointing," in *Proc. IEEE Symp. on Reliable Distr. Syst.*, pp. 39–47, Oct. 1992.

[6] P. Ramanathan and K. G. Shin, "Checkpointing and rollback recovery in a distributed system using common time base," in *Proc. IEEE Symp. on Reliable Distr. Syst.*, pp. 13–21, 1988.

[7] F. Cristian and F. Jahanian, "A timestamp-based checkpointing protocol for long-lived distributed computations," in *Proc. IEEE Symp. on Reliable Distr. Syst.*, pp. 12–20, 1991.

[8] Z. Tong, R. Y. Kain, and W. T. Tsai, "Rollback recovery in distributed systems using loosely synchronized clocks," *IEEE Trans. on Parallel and Distributed Systems*, vol. 3, pp. 246–251, Mar. 1992.

[9] K. Tsuruoka, A. Kaneko, and Y. Nishihara, "Dynamic recovery schemes for distributed processes," in *Proc. IEEE 2nd Symp. on Reliability in Distributed Software and Database Systems*, pp. 124–130, 1981.

[10] B. Bhargava and S. R. Lian, "Independent checkpointing and concurrent rollback for recovery - An optimistic approach," in *Proc. IEEE Symp. on Reliable Distr. Syst.*, pp. 3–12, 1988.

[11] Y. M. Wang and W. K. Fuchs, "Scheduling message processing for reducing rollback propagation," in *Proc. IEEE Fault-Tolerant Computing Symposium*, pp. 204–211, July 1992.

[12] Y. M. Wang and W. K. Fuchs, "Optimistic message logging for independent checkpointing in message-passing systems," in *Proc. IEEE Symp. on Reliable Distr. Syst.*, pp. 147–154, Oct. 1992.

[13] R. E. Strom and S. Yemini, "Optimistic recovery in distributed systems," *ACM Trans. on Computer Systems*, vol. 3, pp. 204–226, Aug. 1985.

[14] D. B. Johnson and W. Zwaenepoel, "Sender-based message logging," in *Proc. IEEE Fault-Tolerant Computing Symposium*, pp. 14–19, 1987.

[15] R. E. Strom, D. F. Bacon, and S. A. Yemini, "Volatile logging in n-fault-tolerant distributed systems," in *Proc. IEEE Fault-Tolerant Computing Symposium*, pp. 44–49, 1988.

[16] D. B. Johnson and W. Zwaenepoel, "Recovery in distributed systems using optimistic message logging and checkpointing," *J. of Algorithms*, vol. 11, pp. 462–491, 1990.

[17] A. P. Sistla and J. L. Welch, "Efficient distributed recovery using message logging," in *Proc. 8th ACM Symposium on Principles of Distributed Computing*, pp. 223–238, 1989.

[18] T. T.-Y. Juang and S. Venkatesan, "Crash recovery with little overhead," in *Proc. IEEE Int'l Conf. on Distributed Computing Systems*, pp. 454–461, 1991.

[19] E. N. Elnozahy and W. Zwaenepoel, "Manetho: Transparent rollback-recovery with low overhead, limited rollback and fast output commit," *IEEE Trans. on Computers*, vol. 41, pp. 526–531, May 1992.

[20] Y. M. Wang, P. Y. Chung, I. J. Lin, and W. K. Fuchs, "Checkpoint space reclamation for independent checkpointing in message-passing systems," Tech. Rep. CRHC-92-06, Coordinated Science Laboratory, University of Illinois at Urbana–Champaign, 1992.

[21] R. D. Schlichting and F. B. Schneider, "Fail-stop processors: An approach to designing fault-tolerant computing systems," *ACM Trans. on Computer Systems*, vol. 1, pp. 222–238, Aug. 1983.

[22] D. B. Johnson and W. Zwaenepoel, "Transparent optimistic rollback recovery," *ACM Operating Systems Review*, pp. 99–102, Apr. 1991.

[23] K. P. Bogart, *Introductory combinatorics.* Pitman Publishing Inc., Massachusetts, 1983.

[24] L. Lamport, "Time, clocks and the ordering of events in a distributed system," *Comm. of the ACM*, vol. 21, pp. 558–565, July 1978.

[25] M. L. Powell and D. L. Presotto, "Publishing: A reliable broadcast communication mechanism," in *Proc. 9th ACM Symp. on Operating Systems Principles*, pp. 100–109, 1983.

[26] A. Borg, J. Baumbach, and S. Glazer, "A message system supporting fault-tolerance," in *Proc. 9th ACM Symp. on Operating Systems Principles*, pp. 90–99, 1983.

[27] A. Borg, W. Blau, W. Graetsch, F. Herrmann, and W. Oberle, "Fault tolerance under UNIX," *ACM Trans. on Computer Systems*, vol. 7, pp. 1–24, Feb. 1989.

[28] I. Anderson, *Combinatorics of finite sets.* Clarendon Press, Oxford, 1987.