Design and Analysis of an Integrated Checkpointing and Recovery Scheme for Distributed Applications

Bina Ramamurthy & Shambhu Upadhyaya
Dept. of Computer Science & Engineering
State University of New York
Buffalo, New York 14260
(716) 645-3180
{bina,shambhu}@cse.buffalo.edu

Bharat Bhargava
Department of Computer Science
Purdue University
West Lafayette, IN 47907
765-494-6013
bb@cs.purdue.edu

Key Words: Checkpointing, concurrent error detection, distributed systems, message logging, rollback recovery

Abstract

An integrated checkpointing and recovery scheme which exploits the low latency and high coverage characteristics of a concurrent error detection scheme is presented. Message dependency which is the main source of multi-step rollback in distributed systems is minimized by using a new message validation technique derived from the notion of concurrent error detection. The concept of a new global state matrix is introduced to track error checking and message dependency in a distributed system and assist in the recovery. The analytical model, algorithms and data structures to support an easy implementation of the new scheme are presented. The completeness and correctness of the algorithms are proved. A number of scenarios and illustrations that give the details of the analytical model are presented. The benefits of the integrated checkpointing scheme are quantified by means of simulation using an object-oriented test framework.

1 Introduction

Checkpointing is a temporal redundancy technique for the recovery from transient failures [1]. Though a number of techniques exist for checkpointing in distributed systems, their main focus has been the global state consistency. With no clear measures to control the recovery time, these techniques cannot be used for recovery in time-critical systems. Replication and comparison based schemes such as triple modular redundancy (TMR) provide the alternative where timely recovery is crucial. While TMR schemes are affordable in mission critical applications, they are not viable in commercial applications due to their prohibitive cost.

In this paper, we consider a class of distributed applications that are not necessarily mission-critical to necessitate replication, but at the same time, desire a rapid failure recovery. Examples include embedded control systems, modeling and simulation programs, and transaction processing systems where timeliness and accuracy of computation are important. All these applications, and many more likely to be available on the internet, have a common characteristic in that they are executed a large number of times on different data sets. We propose a non-replication based solution that integrates concurrent error detection with recovery so that timely recovery is possible at a fraction of the cost of duplication.

Most concurrent error detection schemes are known to be hardware based due to the requirement of real-time monitoring. The proposed checkpointing and recovery scheme which integrates a concurrent error detection tool is thus a hybrid of hardware and software elements. Cache-assisted recovery has been used in [2] to provide a user-transparent support for recovery data. Upadhyaya and Saluja [3] developed an optimization model to determine the effects of recovery cache in rollback systems. Tamir et. al [4] have proposed an on-chip technique to facilitate micro-rollback. Mendelson and Suri [5] devised a cache-assisted recovery method in roll-forward techniques. Our notion of hardware assistance in recovery comes in a different form – we integrate hardware-based concurrent error detection with recovery. The main idea behind considering error detection and recovery together is that certain detection characteristics can be exploited toward developing more efficient and cost-effective recovery schemes.

The integrated checkpointing lets us think differently since much of the research on checkpointing has addressed only the recovery issue and makes implicit assumptions on error detection. Conversely, the concurrent error detection literature is mostly concerned with the detection of anomalous processor operations and does not address recovery. By augmenting the computational nodes of a system with a low cost concurrent error detection tool, the detection and recovery from the most prevalent transient failures will become local without resorting much to system-level methods.

In general, when an error is detected during the execution of a distributed application, recovery is initiated by rolling back one or more processes to a consistent state and resuming execution from that state. The rollback distance and the number of processes involved in the rollback must be contained in order to ensure responsive computation. In the integrated checkpointing and recovery scheme developed by us and first reported in [6], the rollback distance of the process in error is minimized with the aid of the low latency error detection scheme. The number of processes rolled back is limited by a novel message validation mechanism.

The message validation scheme based on concurrent error detection plays a central role in monitoring the processes and initiating the recovery. We present the analysis of this unique recovery scheme and address the design of algorithms and data structures to facilitate its implementation. Our research establishes the correctness and completeness of the scheme. The mechanisms of the scheme and the significance of the various theorems developed are illustrated via different distributed computation scenarios. The benefits of using this scheme and the process completion time overhead are quantified by simulation.

Section 2 provides the background on concurrent error detection that plays a vital role in error containment by offering low latency, low overhead and high coverage. Section 3 identifies the important features of the new recovery scheme including the message validation technique that links error detection to recovery. The algorithms and data structures are analyzed, theorems establishing the properties of the scheme are stated and proved in Section 4. The operational mechanisms of the scheme are illustrated in Section 5. The details of simulation and a discussion of results appear in Section 6. A comparison with related work is given in Section 7. The paper concludes with a discussion on the use and the implementation of the scheme in embedded real-time systems and summarizes future directions of this research.

2 Concurrent Error Detection

Existing checkpointing and recovery schemes (asynchronous checkpointing-based) make fail-stop assumptions or advocate consistency check for error detection. In these schemes errors are detected at the end of a time-out period or at the time of consistency check. This means that the instant of error occurrence and the extent of contamination are unknown. In such cases, the process in error is rolled back to a point where the system is known to be error-free. A dependency graph [7] is generally used to coordinate recovery. With the use of a low latency and high coverage concurrent error detection (CED) tool, an error could be detected immediately with higher (instruction) resolution. Thus, by considering detection and recovery simultaneously, it is possible to locate errors more precisely and consequently limit the rollback distance.

2.1 Details of the Scheme

The domain of our fault model includes transient faults. Research studies [8] have shown that more than 80% failures in computer systems are of non-catastrophic nature. These failures, termed temporary faults, include power supply glitches, radiation hazards, operator errors, unusual combination of events on the chips etc. Any concurrent error detection tool that offers low latency error detection can be used in our integrated checkpointing scheme. Such tools are applicable at instruction level and are based on signature analysis [9], [10]. In these schemes, a graph model is built for the assembly code of the application programs. *Branch-exit* and *branch-entry* points in the code are used to prepare a control flow graph, where a node represents a *branch-free* interval and an edge represents the control flow between the nodes. We propose to use a low cost CED tool developed earlier [11]. In this scheme, the signature for a branch-free sequence of code is derived by applying a signature generating function such as modulo-2 sum successively on the opcode until the signature forms an m-out-of-n code for a specified m and n [11]. An n bit code is an m-out-of-n code if and only if it has exactly m 1's in n positions. The locations in the memory that corresponds to an m-out-of-n code are tagged for signature verification. The tagging of an instruction for the m-out-of-n characteristic is done for the entire code prior to program execution. During execution, the generated signature at a tagged location is checked to determine whether it forms an m-out-of-n code. If it fails to form an m-out-of-n code at the tagged location, an error is indicated. Since no signature is embedded in the instruction stream, the memory overhead is low. It has been shown in [6] that both instruction bit errors and control flow errors caused by transient faults are detected with low latency and high coverage. Transient faults may affect the data, however, data errors can be detected and corrected by error detecting/correcting codes, thereby not requiring any special hardware.

It may be necessary to force a verification at the end of a sequential block of code if the last instruction in the block does not form an m-out-of-n coded signature. Also, whenever a branch instruction is encountered in a block of code, the signature accumulation is continued along the branch, including the branch instruction opcode. At this point, a signature verification point must be forced in order to check for any errors in the transfer of control. This can be done by using one additional byte per branch and adjusting the accumulated signature to form an m-out-of-n code. The signature analysis is an independent process which can be done with minimal interference with the CPU and the operating system. The low overhead implementation of such a scheme has been discussed elsewhere [12]. Note that the described preanalysis and signature generation constitute a one-time cost and is well justified in fault tolerant applications that may be executed a large number of times.

For the sake of completeness, two performance measures of the CED tool are restated here from [6]. In a block of code with n bits of the opcode used to derive the signatures using modulo-2 sum function, the error detection latency is given by

$$\frac{2^n}{\binom{n}{m}} \tag{1}$$

For instance, if n is 8 and m is chosen as, say 3, the average latency is 5.3 instructions.

If the probability of multiple bit error detection at a verification point of a sequential block of code is p, given that an error goes undetected at the 1st, 2nd, and $(l-1)^{th}$ verification point, under certain simplifying conditions, the probability of this error being detected before the l^{th} verification point can be shown to be:

$$p_l = p.\sum_{j=1}^{l}(1-p)^{j-1} \tag{2}$$

By making l reasonably large, this probability of detection can be made very close to 1. The following definitions are required for the development of the integrated checkpointing and recovery scheme.

Definition 1 **A signature verification point** *(SVP) is a location in the code where the signature naturally becomes or is forced by a branch instruction to become an m-out-of-n code.*

Definition 2 *A* **detection point (DP)** *is an SVP that corresponds to the end of a sequence of l blocks such that an error (possibly, multiple-bit error) occurring at the 1st block is detected with a predetermined probability before the end of lth block.*

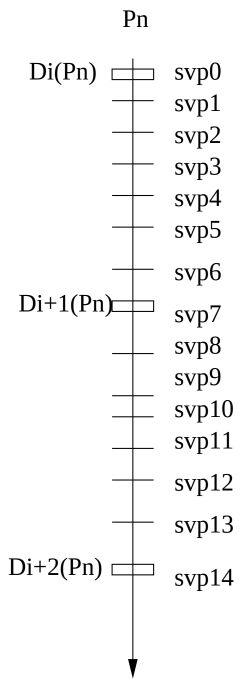


Figure 1: Signature Verification Points and Detection Points

Figure 1 illustrates the execution sequence of an application with a delineation of SVPs and DPs. In this figure, the number of blocks between DPs is 7. The 0^{th} DP coincides with the origin of the process. The $(i+1)^{th}$ DP is l blocks away from i^{th} DP. The detection points play a vital role in realizing a faster commit with reference to communications, which in turn aids error containment and quick recovery.

3 A Model for the Integrated Checkpointing and Recovery Scheme

The system activities of the fault tolerant node, message logging, the new validation scheme based on the merging of concurrent error detection with recovery are described in this section.

3.1 Notation

P_i – i^{th} process

$D_p(P_i)$ – p^{th} detection point of process P_i

$C_p(P_i)$ – p^{th} checkpoint of process P_i

$M_p(P_i, P_j)$ – p^{th} message from P_i to P_j

$A_p(P_j, P_i)$ – Acknowledgment message from P_j to P_i for $M_p(P_i, P_j)$

$V_p(P_i, P_j)$ – Validation signal for message $M_p(P_i, P_j)$

$R_p(P_i, P_j)$ – Rollback initiator message (error signal) from P_i to P_j corresponding to p^{th} error scenario

S_p – Global system state matrix at time T_p

3.2 System Activities

A distributed system comprising a set of processors or nodes and a distributed application with N processes $\{P_0,...P_{N-1}\}$ is considered. Each process consists of instructions to carry out computations interleaved with communication for synchronization or data transfer among the processes. The processes communicate through message passing and do not share any memory. No assumption is made on the topology and the type of network. The activities during the execution of a distributed application are broadly classified into (i) those taking place within a process – *intra-process activities*, and (ii) that cause dependency among processes – *inter-process activities*. Each node has a CED hardware called a watchdog processor that is integrated into it. Since most watchdog processors do not directly support process switch, we assume that a process is allowed to continue on a processor until it is completed or until an error is detected. This assumption can be relaxed by extending the capabilities of the watchdog. The watchdog is assumed to detect temporary malfunctions by monitoring instruction bit errors and control flow errors. The system architecture is shown in Figure 2 where the hardware and software components are separated for clarity.

3.2.1 Intra-process Activities

The three main intra-process activities are (i) instruction execution (ii) concurrent error monitoring and (iii) checkpointing. The error detection mechanism monitors for errors concurrently with instruction execution. The execution path of a process is logically delineated by detection points as shown in Figure 1. In order to handle rollback recovery and message logging effectively, an event trace is maintained for each node. This trace includes the following information for each message: the type of message – received or transmitted, a unique message identification, an acknowledgment bit, a validation bit, a timestamp and a pointer to the saved message in a stable storage.

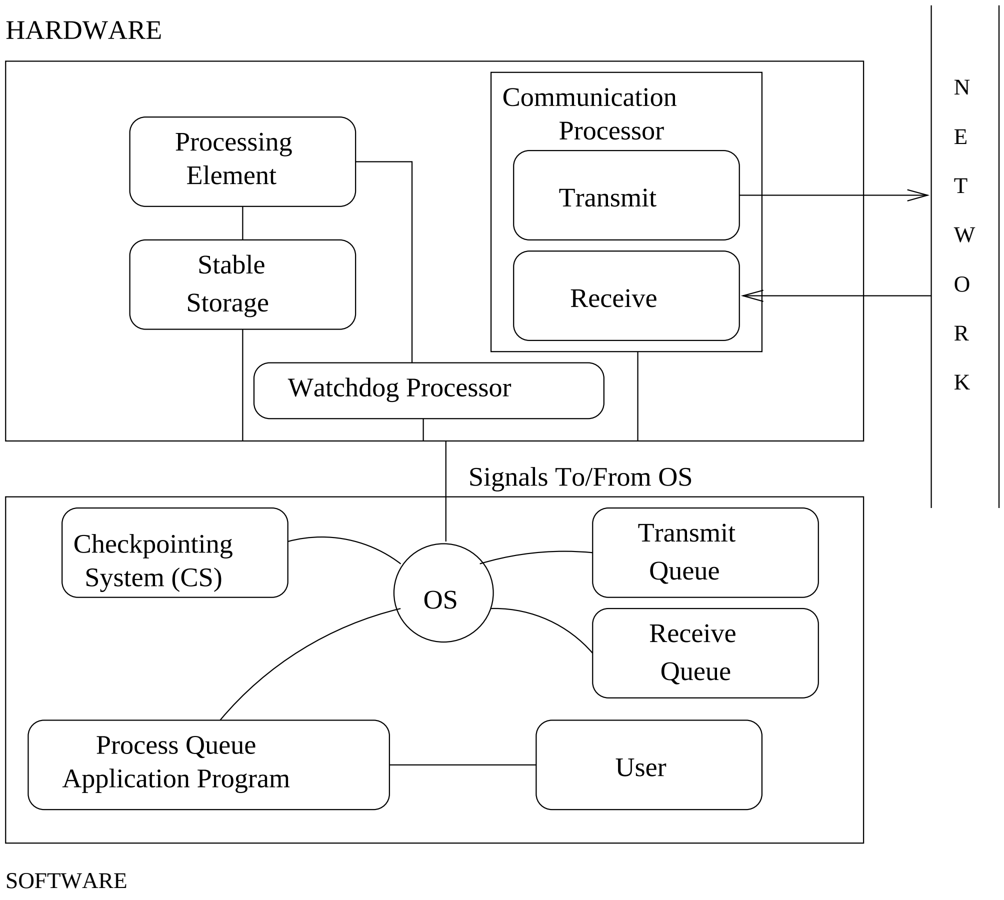


Figure 2: Structure of a Fault Tolerant Node in a Distributed System

Checkpointing involves saving the state of a process in a secondary store or in a special hardware such as a cache [2], [5]. In process control systems where recovery within a short period of time is critical, instruction-level fine-grain checkpointing may be needed. Any policy such as equidistant checkpointing or checkpointing on the basis of runtime characteristics [3] can be used for checkpointing. We assume that a mechanism is in place to discard unwanted checkpoints so that the size of the stable storage can be minimized. In conventional recovery, acceptance tests [13] are performed before actual state saving operation at the checkpoints. In our approach, the signature comparison at the detection points is used as an acceptance test.

3.2.2 Inter-process Activities

The communication among processes is implemented by *sends* and *receives*. In general, *sends* are *non-blocking* and *receives* are *blocking*. This means that a process after sending a message continues execution without waiting. But when a message is required by a process, it blocks itself waiting to receive the message. The different types of messages are:

1. The regular messages generated by the application.
2. The special messages used by the recovery scheme,
   (a) Acknowledgment message $A_p(P_j, P_i)$ following the successful logging of a message by the receiver in the stable storage (discussed in Section 3.3).
   (b) Validation message $V_p(P_i, P_j)$ used to confirm that a message sent earlier is not contaminated by error(s) (discussed in Section 3.4).
   (c) Recovery message $R_p(P_i, P_h)$ to initiate a rollback due to any propagated error p. (discussed in Section 3.7).

A communication processor is assumed to be available within each node (as shown in Figure 2) that handles the inter-processor communication. No assumption is made on the message delivery protocol. However, it is implicitly assumed that a message always arrives at the destination prior to the arrival of its validation signal. Validation and recovery messages are sent as separate messages when the message traffic in the network is low and are piggy-backed to the regular messages when communication traffic is high.

3.3 Message Logging

Message logging is one of the critical factors affecting the performance of error recovery. A variety of methods [14] [15] are available for message logging. Our scheme uses *receiver-based* message logging. The sender continues execution after sending a message to the receiver. The method is pessimistic in the sense that the receiver logs a message as soon it is received. It then sends an acknowledgment to the sender to notify the successful logging of the message. All the messages logged at the receiver are validated periodically using a validation signal from the sender. This information is used for the conditional rollback when an error is detected. A piecewise deterministic model of execution [16] is assumed and messages are replayed from the receiver log during recovery. Atomic message logging required by a pessimistic scheme is eliminated here by the use of the acknowledgment scheme. The disadvantage of regular optimistic logging is the excessive rollback when an error occurs [17]. This problem is alleviated in our approach by using a novel *validation scheme.*

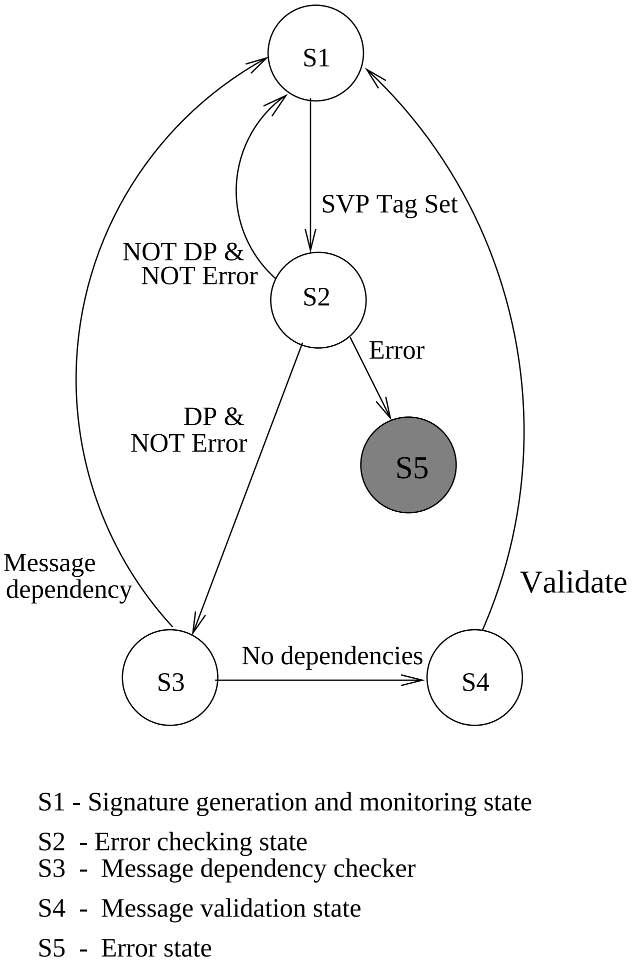


Figure 3: Error Detection and Message Validation

3.4 Message Validation

Message validation is a mechanism for declaring that a set of one or more messages transmitted is not contaminated by an error occurring in a process. If a message is known to have been transmitted before the occurrence of an error in a process, the recipient process need not rollback due to any local error at the sender. This eliminates any undue rollbacks. Moreover, the low latency of detection is instrumental in reducing the rollback distance since an error is detected as soon as it occurs.

The validation process is explained using Figure 3. State $S1$ represents the signature generator that works concurrently with the main process execution. When the SVP indicator tag is set, error checking is done ($S2$). If there is no error and the SVP is a detection point then certain messages can be validated. Following definitions are used in developing the validation method.

Definition 3 *A detection interval for process P_i is the duration between two consecutive detection points $D_k(P_i)$ and $D_{k+1}(P_i)$.*

It is necessary that one detection interval (l SVPs) has to elapse past the occurrence of an error in order to ensure its detection with a predetermined probability of nearly 1.0 (eq. (2)). Thus, if no error is detected at $D_k(P_i)$, one could infer that no error would have occurred anywhere in the interval $\{D_{k-2}(P_i),\ D_{k-1}(P_i)\}$. Therefore, the messages sent in that interval can be validated as free from local errors in node P_i. But there may be dependency on the other processes due to prior communications and received messages.

Definition 4 *A message $M_p(P_i, P_j)$ sent by process P_i is said to be a* dependent message *if there is at least one message $M_q(P_h, P_i)$, which was received prior to sending $M_p(P_i, P_j)$, and is yet to be validated by the sender process P_h.*

Definition 5 *A message $M_p(P_i, P_j)$ sent by a process P_i is said to be* independent *if (a) no message was received prior to sending $M_p(P_i, P_j)$ or (b) all messages received prior to $M_p(P_i, P_j)$ have been validated.*

From Definitions 4 and 5 it is clear that any dependent message will eventually become independent through validation, provided there are no errors. The steps of message validation for a process P_i are given in the following algorithm.

Algorithm 1 *At detection point $D_k(P_i)$, $0 \leq i \leq N-1$,*

1. *Any independent (outgoing) messages in the interval $\{D_{k-2}(P_i),\ D_{k-1}(P_i)\}$ are validated.*
2. *Any messages prior to $D_{k-2}(P_i)$ that have become independent due to validation messages that arrived since $D_{k-1}(P_i)$ are also validated.*

State $S3$ in Figure 3 checks the message dependencies. In state $S4$ the process transmits validation signals if all the message dependencies have been satisfied. If not, the message validation is deferred until the next detection point $D_{k+1}(P_i)$. The validation message contains the sender sequence number of the validated message. In the existing schemes validation process either does not exist or it is carried out only at a checkpoint. By validating messages much more frequently, fewer processes need to rollback when an error occurs, as explained in the next section. This is one of the distinguishing features of our scheme. Although validation checking takes place at every DP, it involves the checking of only a single bit in the event trace and is thus inexpensive.

It is possible that a transmitted message may not be acknowledged by the receiver, and a validation signal may not be received for a message, due to errors at the receiver and sender respectively. In order not to endlessly wait for either an acknowledgment or a validation signal, timeouts can be enforced.

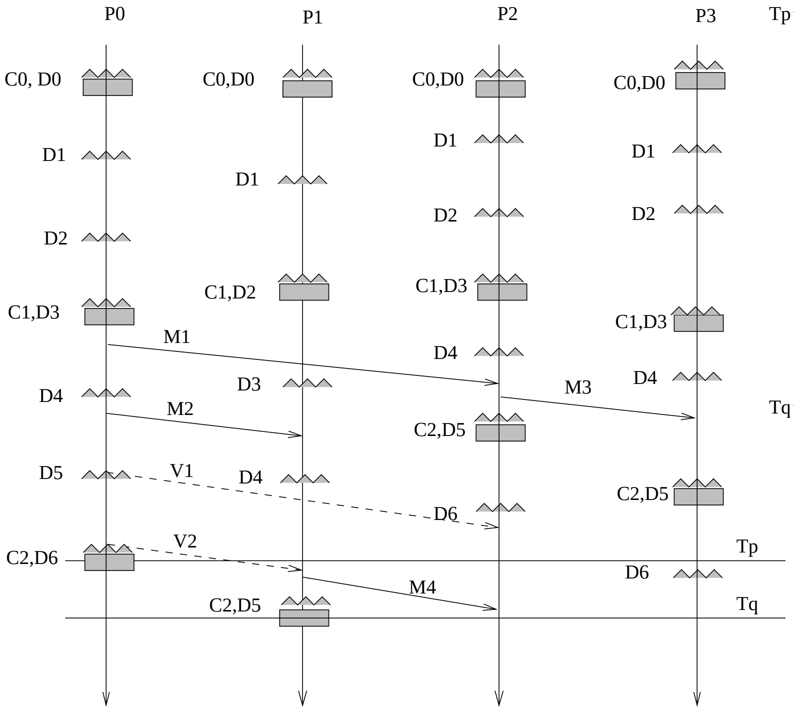
3.5 Global System State

The distributed system state is a snapshot of the system [18] observed from outside. A matrix is used to represent the system state. This matrix contains the current detection points of the processes and the dependencies brought about by the messages, and is saved at a checkpoint as a part of the event trace. It is one of the attributes that is restored during recovery. It is updated at every DP of a process. Message validation is done after this matrix is updated. When an error is detected at a DP of a process, the matrix entries are used to decide the extent of rollback of the process in error and which other processes are to be rolled back. The notion of a safe detection point is introduced prior to defining the global state matrix.

Definition 6 *At a given time T_p, a detection point $D_k(P_i)$ is* safe *with reference to another process P_j, $i \neq j$, if P_j has no message dependency on P_i in the interval $\{D_0(P_i), D_k(P_i)\}$.*

In other words, P_j has not received any messages from P_i during the interval $\{D_0(P_i), D_k(P_i)\}$ or if it has received any, they have been validated. The notion of safe DP indicates the extent of rollback of a process on account of rollback of another process that is dependent on it. Consider the global time T_p in Figure 4(a). For clarity and ease of readability the messages in Figure 4 (and Figure 5) are numbered sequentially using a simpler notation. For instance, $D_4(P_2)$ is safe with reference to all the processors since P_2 has not transmitted any messages before D_4. On the other hand, $D_5(P_2)$ is not safe with reference to P_3 since there is a message $M_3(P_2, P_3)$ indicated as M_3 in the figure which is yet to be validated. Once this message is validated $D_5(P_2)$ will become safe with reference to P_3.

Definition 7 *A global state matrix (GSM) S_p at some global observation time T_p is an $N \times N$ matrix with its diagonal elements made up of current local detection point indices at T_p. An entry $S_p[P_j, P_i]$ is the latest* safe *detection point of P_i with reference to P_j.*



(a) Distributed System Timing Diagram : Scenario 1

Tp	P0	P1	P2	P3
P0	6	4	6	5
P1	4	4	6	5
P2	6	4	6	5
P3	6	4	4	5

(b) Global State Matrix at Tp

Tq	P0	P1	P2	P3
P0	6	5	6	5
P1	4	5	6	5
P2	6	4	6	5
P3	6	5	4	5

(c) Global State Matrix at Tq

Figure 4: Activities and Global State Matrices at T_p and T_q

For simplicity of representation, only the indices of the DPs are entered in the matrix. Figure 4(b) shows the GSM at time T_p. Dependency of a process P_i on other processes is given by row i of the GSM. Consider process P_0. This process has not received any messages from other processes. So, the current DP indices of the respective processes form the row entries corresponding to P_0. Consider process P_1. P_1 is dependent on $D_4(P_0)$ due to the message indicated by M_2 (note that V_2 will be issued only after the updating of the GSM is over). Therefore, the first entry of row P_1 is 4. Since there are no received messages with outstanding validation, the entries of the row corresponding to P_2 are the current DP indices of the respective processes. Finally, consider the row corresponding to process P_3. P_3 is dependent on P_2 because of M_3 which has not been validated yet. Therefore, $S_p[P_3, P_2]$ is 4.

Even though the GSM is shown as one entity, the information contained in it is distributed among the processes. More specifically, column j of the matrix is maintained at P_j in the event trace of the node. The GSM is updated every time a new detection point is reached to reflect the new dependency structure of the system. The dependency structure of the processes in the system is affected by the transmitted messages. The following algorithm gives the activities related to updating the global state matrix.

Algorithm 2 *At each process j, $0 \leq j \leq N-1$, when a detection point $D_k(P_j)$ is reached,*

1. *The j^{th} element of the j^{th} column of the global matrix is updated, $S_p[P_j, P_j] = D_k(P_j)$.*
2. *For every h, $0 \leq h \leq N-1$, $h \neq j$,*
   - (a) *Let $x = S_p[P_h, P_j]$,*
   - (b) *If there are no messages from P_j to P_h in the interval $\{D_x(P_j),\ D_k(P_j)\}$ then $S_p[P_h, P_j] = D_k(P_j)$.*
   - (c) *If there are messages from P_j to P_h in the interval $\{D_x(P_j), D_k(P_j)\}$, and if all the messages in the interval $\{D_x(P_j), D_r(P_j)\}$ where $r \leq k$ have been validated, then global state matrix is updated, $S_p[P_h, P_j] = D_r(P_j)$.*

Consider the time line T_q and the corresponding GSM in Figure 4(c). The current detection point of P_1 has become D_5 and there has been a message from P_1 to P_2 since T_p. Step 1 of Algorithm 2 is applied in updating $S_p[P_1, P_1]$ to D_5. Step 2(b) is used in updating $S_p[P_0, P_1]$ and $S_p[P_3, P_1]$ to D_5, the current detection point. Entry $S_p[P_2, P_1]$ remains unchanged due to the message M_4, which is yet to be validated.

3.6 Response to Validation

When a validation message is received by a process P_i, either directly or piggy-backed on a regular message (i) one or more messages received earlier are marked valid, and (ii) if there are any previous messages dependent on the messages that just got validated, those are also validated. At the next detection point, the GSM is updated to reflect the latest validations. For example, in Figure 4 after the validation V_2 is issued, $S_q[P_1, P_0]$ will be updated to its current DP at the next DP.

3.7 Recovery Model

When an error is detected in a process P_i, its execution is suspended and a rollback is initiated. One or more processes will have to be rolled back to establish consistency. The recovery activity of the erroneous process is termed *primary rollback*. The rollback of process P_j initiated as a result of an error signal $R_l(P_i, P_j)$, is termed an *induced rollback*. Algorithms 3 and 4 describe the primary and induced rollbacks respectively.

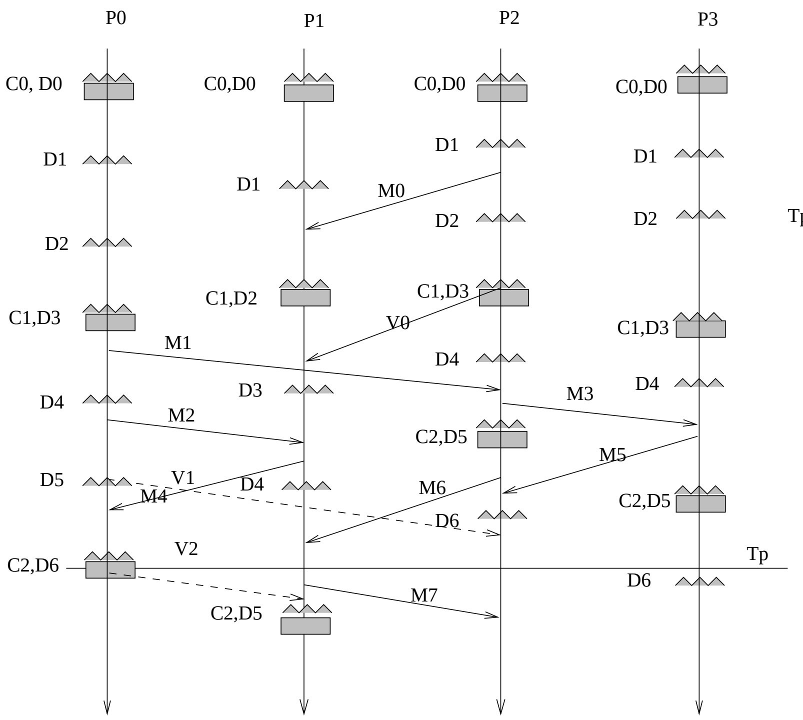
Algorithm 3 *When an error l is detected at $D_k(P_j)$,*

1. *Let the global state matrix be S_p.*
2. *Process P_j rolls back to the most recent checkpoint C_q prior to $D_{k-2}(P_j)$.*
3. *For every i, $(0 \leq i \leq N-1)$ and $(j \neq i)$,*
   (a) *Let $D_x = S_p[P_i, P_j]$, $D_y = S_p[P_j, P_j]$.*
   (b) *If $D_x < D_y$, then a rollback message $R_l(P_j, P_i)$ is sent to P_i.*

Algorithm 4 *When a process P_j receives a rollback message $R_l(P_i, P_j)$*

1. *If (it already received $R_y(P_h, P_j)$, $0 \leq h \leq N-1$, and if $y = l$), then it is ignored, Else*
2. (a) *Let the global state matrix be S_p.*
   (b) *Let $D_x = \min(S_p[P_x, P_j])$, $0 \leq x \leq N-1$.*
3. *Process P_j is rolled back to the most recent checkpoint C_r prior to D_x.*
4. *For every h, $(0 \leq h \leq N-1)$ and $h \neq j$,*
   (a) *Let $D_y = S_p[P_h, P_j]$.*
   (b) *Let $D_z = S_p[P_j, P_j]$.*
   (c) *If $D_y < D_z$, then a rollback message $R_l(P_j, P_h)$ is sent to P_h.*

An important point is that any repeated reception of rollback messages will be ignored since a process rolls back to a safe point with reference to all processes on the first notification of rollback. New errors are allowed to occur during rollback. The rollback messages related to different error scenarios carry different identifiers as indicated in Step 1 of the algorithm. In other words, all rollback messages for a given error have the same rollback identifier and each process will rollback at most once for a given error scenario. Moreover, the chain of rollback messages terminates once every process involved in the rollback gets a rollback message. In the best case, no process except the one where the error is detected needs to be rolled back. The worst case occurs when all the processes involved are sent rollback messages in a daisy-chain fashion.



Tp	P0	P1	P2	P3
P0	6	3	6	5
P1	4	4	5	5
P2	6	4	6	4
P3	6	4	4	5

(b) Global State Matrix

(a) Distributed System Timing Diagram : Scenario 2

Figure 5: Activities and Global State Matrices at T_p

4 Analysis

The concept of safe detection points is extended and used in formalizing the recovery model designed in Section 3.7. The relation that transforms state matrices is defined and its properties analyzed. It is then used to prove that a set of checkpoints that form a system recovery line is consistent. It is also shown that the validation mechanism will not result in a deadlock.

Lemma 1 *If a detection point $D_p(P_i)$ is safe with reference to P_j, then all detection points $D_q(P_i)$, $0 \leq q \leq p-1$, are also safe with reference to P_j.*

Proof: According to Definition 6, when $D_p(P_i)$ is safe, P_j has no dependency on P_i in the interval $\{D_0(P_i),D_p(P_i)\}$. Any interval $\{D_0(P_i),\ D_q(P_i)\}$, $q \leq p$ is included in the interval $\{D_0(P_i),D_p(P_i)\}$. This implies that P_j has no dependency on P_i in the interval $\{D_0(P_i),\ D_q(P_i)\}$. Then according to Definition 6, any detection point $D_q(P_i)$, $q < p$ is also safe.

Consider the scenario depicted in Figure 5. From the GSM in Figure 5 (b), $S_p[1,2]$ is 5, implying that $D_5(P_2)$ is safe with reference to P_1. Consider all D_q's, $q < 5$. The message sent from $\{D_1(P_2), D_2(P_2)\}$ has been validated at D_3. So, D_2 and D_1 are safe. Since no messages were sent, D_3 and D_4 are also safe.

Lemma 2 *Let S_p be the global state matrix at a detection point $D_k(P_j)$. If an entry $S_p[P_i, P_j]$, $i \neq j$, is less than the corresponding diagonal entry $S_p[P_j, P_j]$ then P_i is dependent on P_j at $D_k(P_j)$.*

Proof: According to step 2 of Algorithm 2, an entry $S_p[P_i, P_j]$, $i \neq j$ is updated to the current detection point $D_k(P_j)$ if there are no messages between $D_p(P_j)$ given by the entry $S_p[P_i, P_j]$ and $D_k(P_j)$. $S_p[P_j, P_j]$ holds the current detection point. So if an entry in column P_j is less than $S_p[P_j, P_j]$, it implies that one or more messages have been sent by P_j to P_i in the interval $\{D_p(P_j),\ D_k(P_j)\}$ which are yet to be validated. This means that P_i is dependent on P_j.

Lemma 2 formalizes the algorithms discussed in the recovery model. Only those processes dependent on the process in error need to be rolled back. Consider $S_p[P_0, P_1]$ which is 3. This is less than $S_p[P_1, P_1]$ which is 4. Message M_4 from P_1 to P_0 is yet to be validated at T_p. Thus P_0 is dependent on P_1 at T_p. This dependency is clearly indicated by the inequality $S_p[P_0, P_1] < S_p[P_1, P_1]$.

The global state matrix is updated every time a new DP is encountered in any process. The set of all state matrices that occur during a single execution of an application is identified as S. We next show that it is sufficient to keep only one copy of the GSM to determine the dependency of messages.

Definition 8 *The relation $\trianglelefteq$ over the set of global state matrices S transforms any element S_p into another element S_q such that $\forall i, j,\ 0 \leq i, j \leq N-1$, (i) $S_p[P_i, P_j] = S_q[P_i, P_j]$ or (ii) $S_p[P_i, P_j] < S_q[P_i, P_j]$.*

Theorem 1 *The set of values a global state matrix assumes during a single execution of a process is a totally ordered set where $\trianglelefteq$ is the total ordering relation.*

Proof: In order to prove that $\{S, \trianglelefteq\}$ is a totally ordered set it is required to prove that the relation $\trianglelefteq$ is (i) reflexive, (ii) anti-symmetric, (iii) transitive, and (iv)$\forall S_p, S_q$ in S, $S_p \trianglelefteq S_q$, or $S_q \trianglelefteq S_p$.

(i) For any element S_p of S, $S_p \trianglelefteq S_p$. Therefore, it is reflexive.

(ii) The global state matrix at the origin of execution is all 0's and its entries are incremented as the execution progresses. For any element S_p of S, (a) $S_p[P_i, P_j]$, for $i = j$, is incremented at every D_p and (b) $S_p[P_i, P_j]$, for $i \neq j$ may be incremented when a validation is received. Consider two states S_p and S_q. Assume that S_p happened before S_q. Thus the relation $\trianglelefteq$ holds: $S_p \trianglelefteq S_q$. Suppose that $S_q \trianglelefteq S_p$. This implies that S_q happened before S_p, or $S_q = S_p$. The former contradicts our initial assumption. Therefore, $S_p \trianglelefteq S_q$ and $S_q \trianglelefteq S_p$ can happen only if $S_p = S_q$. This proves anti-symmetry.

(iii) Assume three state matrices S_p, S_q, and S_r of the set S such that $S_p \trianglelefteq S_q$ and $S_q \trianglelefteq S_r$. According to the definition of $\trianglelefteq$, S_p happened before S_q and S_q happened before S_r. Therefore S_p happened before S_r. In other words, $S_p \trianglelefteq S_r$. Thus the relation $\trianglelefteq$ is transitive.

(iv) This follows directly from the definition of the global state matrix and the relation $\trianglelefteq$.

Since $\trianglelefteq$ is reflexive, anti-symmetric, and transitive and since it satisfies the property (iv) over set S, $\{S, \trianglelefteq\}$ is a totally ordered set.

Let S_p be the current global state matrix. Let B be the subset of all global state matrices up to S_p. $B \subset S$.

Lemma 3 *If S is totally ordered by the relation $\trianglelefteq$, any subset B of S is also totally ordered by the same relation.*

Proof: The four properties of totally ordered set are preserved under the restriction of its domain to any subset. So $\{B, \trianglelefteq\}$ is also totally ordered.

Theorem 2 *If S_p is the current state matrix ($S_p \in S$), and B, the set of matrices up to S_p ($B \subset S$), then S_p represents the least upper bound of B.*

Proof: Let S_0 be the initial state matrix. The p^{th} global matrix S_p is obtained from S_{p-1} by updating at least one of its elements:

$\exists i, j,\ 0 \leq i, j \leq N-1 : S_p[P_i, P_j] > S_{p-1}[P_i, P_j]$. Also $S_p > S_q$ where $S_q \in B$, for any q. S_r, $(r > p)$ is obtained by incrementing an element of S_p. So S_p is the least upper bound for B.

Theorems 1 and 2 establish that it is necessary to keep only the latest global state matrix for determining the dependency for rollback recovery.

Theorem 3 *The number of rollbacks for a given system-wide recovery is bounded by the number of processes N.*

Proof: The domino effect is caused by a cascade of induced rollbacks. Induced rollbacks are due to message dependencies. Each process P_j rolls back to a checkpoint C_p prior to the lowest numbered safe detection point represented in the column P_j of the current GSM. Since a process has rolled back to a safe checkpoint on receiving the first rollback message itself, other rollback messages associated with the current error will not result in further rollback. In the worst case all the processes will rollback but each of them will rollback only once. Thus the number of rollbacks is bounded by the number of processes.

Consider Figure 4. Assume that an error is detected at D_6 of P_3. According to Algorithm 3, P_3 will rollback to the nearest checkpoint preceding D_4. Note that this is the maximum extent that P_3 will rollback irrespective of the messages it gets during the current recovery process. P_3 will send an error message to P_2 which will rollback to $C_1(P_2)$ preceding D_4. This is the lowest number in P_2 column of the GSM in Figure 4(b). It will in turn send rollback messages to P_1 and P_3. P_3 has already rolled back, so the message is ignored. P_1 rolls back to $C_1(P_1)$ preceding D_3 and sends a rollback message to P_0. P_0 will rollback to $C_1(P_0)$ at D_3 and will send an error message to P_1. P_1 having rolled back before will ignore this message. In this scenario, all the processes rollback and they do it only once for a given error.

Lemma 4 *Any message $M_p(P_i, P_j)$ during error-free operation of a system will get validated in a finite time.*

Proof: If a message is sent in the interval $\{D_{p-2}(P_i),\ D_{p-1}(P_i)\}$ and if there is no dependency, then it gets validated at $D_p(P_i)$. The worst case scenario is a circular wait involving two or more processes. We now prove that it is not possible. Lamport's *happened-before* relationship among events [18] will be used in this proof. The dependencies or *wait* among the processes arise due to messages which are yet to be validated. Consider three processes P_i, P_j and P_k. P_i sends a message $M_q(P_i, P_j)$ to P_j. After receiving $M_q(P_i, P_j)$, P_j sends a message $M_r(P_j, P_k)$ to P_k and after receiving it, P_k sends message $M_s(P_k, P_i)$. A circular wait will occur if P_i waits for the validation of $M_s(P_k, P_i)$ in order to validate $M_q(P_i, P_j)$. It is necessary to prove that this is not possible. Let the *happened-before* relation be represented by $<$. P_j waiting for validation $V_q(P_i, P_j)$ to validate its outgoing message M_r implies that $M_q(P_i, P_j)$ was sent prior to $M_r(P_j, P_k)$. In other words, $M_q(P_i, P_j) < M_r(P_j, P_k)$. P_k waiting for $V_r(P_j, P_k)$ in order to validate the message $M_s(P_k, P_i)$ implies that the arrival of $M_r(P_j, P_k)$ happened before $M_s(P_k, P_i)$. Applying transitive closure to $M_q(P_i, P_j) < M_r(P_j, P_k)$ and $M_r(P_j, P_k) < M_s(P_k, P_i)$, it can be seen that $M_q(P_i, P_j) < M_s(P_k, P_i)$. So P_i is not waiting for $M_s(P_k, P_i)$ to be validated in order to validate $M_q(P_i, P_j)$. Therefore circular wait is not possible. This implies that an indefinite wait cannot happen.

Lemma 5 *The validation scheme does not result in system deadlock.*

Proof: When there is no error, the messages get validated within a finite time according to Lemma 4 and so there is no possibility of a deadlock. When error occurs there may a circular wait among the processes due to the unavailability of the validation message from the process in error. Since the rollback initiated as a response to the error rolls the processes back to safe detection points, the circular wait is automatically resolved, thus preventing deadlocks. Moreover, the validation scheme is distributed. A validation message determines the dependencies at the sender process which is maintained in the column of the dependency matrix. If the sender process fails, when it rolls back it takes care of the validation. If the receiver process fails before receiving validation, the validation message will be buffered by the underlying communication protocol for delivery after the receiver's recovery.

Definition 9 *A set of checkpoints forms a* recovery line *if there is no message dependencies among the processes prior to each of the checkpoints in the set.*

Traditionally a recovery line is considered consistent if there is no message cutting the recovery line. In other words, a message sent before rollback recovery but received after rollback will result in an inconsistency. The validation scheme, and the definition of safe detection points based on the message dependencies ensure that this inconsistency does not happen when using our scheme. In Figure 5, considering the global state matrix at T_p, the recovery line is given by $\{C_1(P_0), C_1(P_1), C_1(P_2), C_1(P_3)\}$.

Theorem 4 *The recovery line defined by Algorithms 3 and 4 based on the global state matrix is a (i) maximal (ii) consistent and (iii) correct recovery line for the system.*

Proof: (i) It is maximal since current global state matrix S_p is the least upper bound of the chain formed by the current set of global state matrices S.

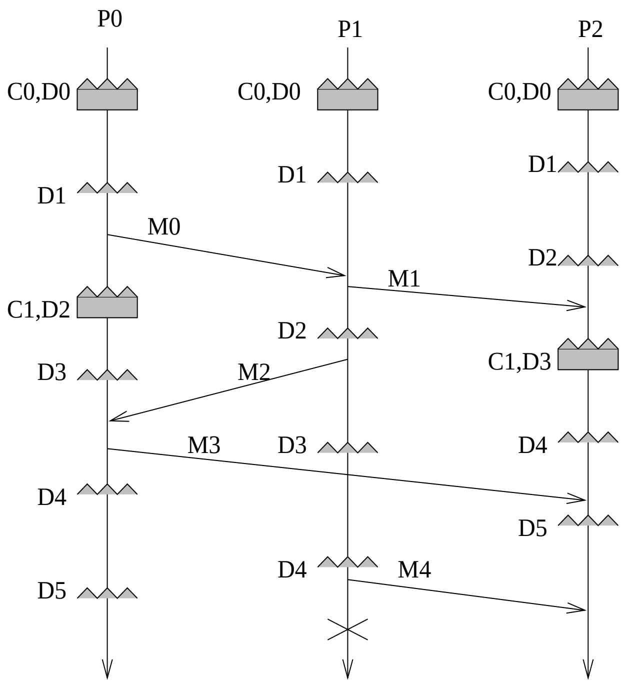


Figure 6: A Scenario with Process Communications, and an Error

(ii) Any entry $S_p[P_i, P_j]$, $i < j$, represents a safe detection point. A safe detection point implies message independence among the processes involved. So the recovery line established by the recovery algorithm is consistent. (iii) The recovery line chosen by the Algorithms is dependent on the message validation which is proven to be deadlock-free by Lemma 5. It is maximal and consistent as proven above. Thus Algorithms 3 and 4 define a correct recovery line.

5 Illustration

The proposed recovery scheme improves performance by reducing the number of induced rollbacks and also the rollback distance. The distinction between the conventional recovery scheme (that does not use hardware-assistance) and this scheme is illustrated using a communication scenario given in Figure 6.

Let an error be detected at $\times$ in process P_1. Then, process P_1 rolls back to the checkpoint $C_0(P_1)$. If error checking were done only at the checkpoints as in most existing schemes, all three outgoing messages M_1, M_2 and M_4 are suspects. This means that P_0 rolls back to $C_1(P_0)$ and process P_2 has to rollback to $C_0(P_2)$.

Now consider the scenario where the message validation decisions are made at the detection points. When the execution reaches $D_3(P_0)$, the message M_0 can be validated because an error detection interval has elapsed since the transmission of this message. Similarly, messages M_1 and M_2 would have been validated at DPs $D_3(P_1)$ and $D_4(P_1)$ respectively. With an error detection at $\times$, process P_1 will undergo a primary rollback. M_4 is yet to be validated. Hence P_1 sends a rollback message to P_2 which executes an induced rollback to $C_1(P_2)$. Note that when using conventional scheme P_2 would have rolled back to $C_0(P_2)$. Using the new scheme P_0 does not have to rollback at all and the messages M_1 and M_2 will not be retransmitted. Thus the number of induced rollbacks (only P_2 instead of P_2 and P_0) as well the extent of these rollbacks (to $C_1(P_2)$ instead of $C_0(P_2)$) have been cut short by our scheme.

The above illustration can be formally studied by developing the global state matrices and going through the steps of the recovery algorithms of Section 3. The savings in the number of induced rollbacks and the extent of rollback is obtained at the cost of four additional validation messages compared to the conventional scheme.

6 Experimental Evaluation

The preliminary simulation of our scheme was conducted using a process-based simulation tool YACSIM [19]. Since our scheme involved fault detection, we needed a tool with a fault-injection capability along with general purpose simulation support. We found such a tool in DEPEND [20] which is a dependability analysis tool based on the robust simulation package CSIM [21]. With DEPEND as the core, we built an object-oriented testbed for simulation of checkpointing and recovery called OTEC [22] to allow for design, development and comparison of different checkpointing and recovery schemes in a common test environment under realistic fault scenarios. OTEC reuses and extends many of the facilities provided by DEPEND and CSIM. OTEC is being used for the simulation reported here.

6.1 Simulation and Comparison

In the simulation study of the new scheme, a distributed application with multiple processes was implemented. In order to allow for concurrent error detection such as signature analysis-based schemes, real code is run on the simulated machine. SPEC'92's *compress.c* yielded five reasonably long traces each with varying interval sizes. Four of these are run on four distributed processes in our experiments. Processes are divided into checkpoint intervals, and checkpoint intervals into detection intervals. Messages among the processes are simulated and are uniformly distributed along the code. Message initiation overhead is counted as one instruction. The message rates used range from 0 to 10 messages per 100 instructions and the error rates range from 0 to 1 per 1000 instructions. The magnitude of the error detection latency and the recovery time are at the instruction cycle-level granularity. Since the error detection is done by a watchdog processor, the detection latency is typically in the order of a few instructions.

Three different schemes are simulated: (a) the new integrated checkpointing scheme (referred as SchemeA) (b) a scheme by Johnson [16] (referred as SchemeB) and (c) conventional recovery, without hardware integration (referred as SchemeC). SchemeB does not use a concurrent error detection tool. It also differs from SchemeA in the way recovery is handled. During recovery it sends rollback messages to all the processes in the system and it is up to the individual processes to determine if they need to rollback. Detailed description of the scheme can be found in [16]. SchemeC is a traditional recovery scheme in which faults are detected through consistency checks performed before taking checkpoints.

The three schemes are simulated on OTEC and the overhead in terms of process completion times with respect to error rates and message rates is monitored. In particular, two experiments (i) fractional time overhead versus message rate at a constant error rate of 0.0005 (5 in 10000 instructions) and (ii) fractional time overhead versus error rate at a constant message rate of 0.1 (1 message per 10 instructions) are presented.

6.2 The Results

The Y-axis in the figures represents the fractional increase in the time of completion of a process with respect to its time of completion under error-free condition. The X-axis is message rate (0.0 to 0.09) in Figure 7, and error rate (0.0 to 0.0009) in Figure 8.

Consider Figure 7. The corresponding experiment studies the effect of messages and the resultant overhead at low error rate. When the error rate is kept low and constant, the effect of messages on the overhead is determined. For a nominal message rate of 0.05 (5 messages per 100 instructions), the overheads of the various schemes are: SchemeA 12%, SchemeB 21%, and SchemeC 24%. SchemeA incurs only half the overhead as SchemeC, whereas SchemeB provides only a marginal improvement of 3% over SchemeC. This is attributable to the elaborate message-dependent commit routine to be executed in SchemeB. SchemeA, on the other hand, handles

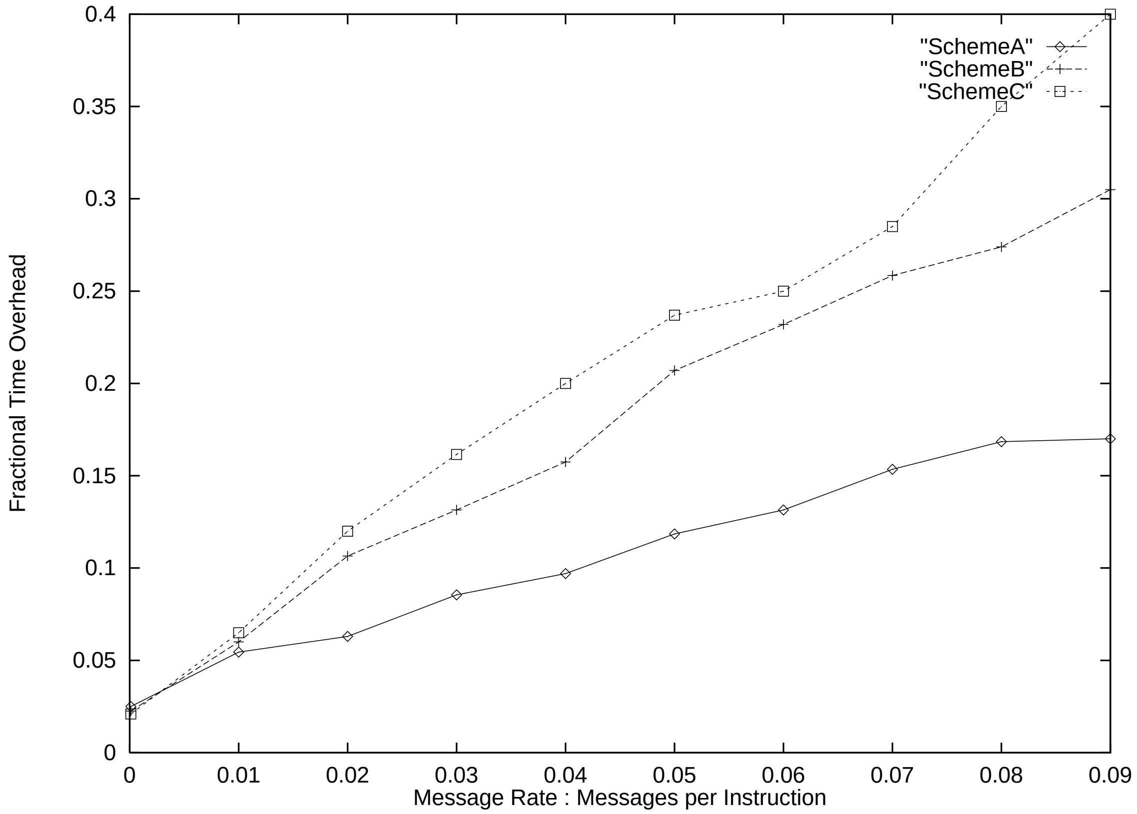


Figure 7: Completion Time Overhead Versus Message Rate – (Error Rate 0.0005)

a message by a simple validation between the communicating processes. Another significant observation is that as the messages rate increases, the overhead in SchemeA rises much slower than the other two.

Consider Figure 8. For a nominal error rate of 0.0005 (5 errors per 10000 instructions), the overheads of the various schemes are: SchemeA 21%, SchemeB 33%, and SchemeC 45%. Special rollback recovery method of both SchemeA and SchemeB helps to flatten the overhead at high error rates. In this case too, SchemeA's overhead is consistently lower than that of SchemeB. The conventional scheme with no special recovery assistance performs rather poorly at high error rates.

The generation of acknowledgment and validation messages in our method may potentially incur communication overhead. However, the overhead is not significant when the system-related communication intensity is low since messages can utilize the unused bandwidth.

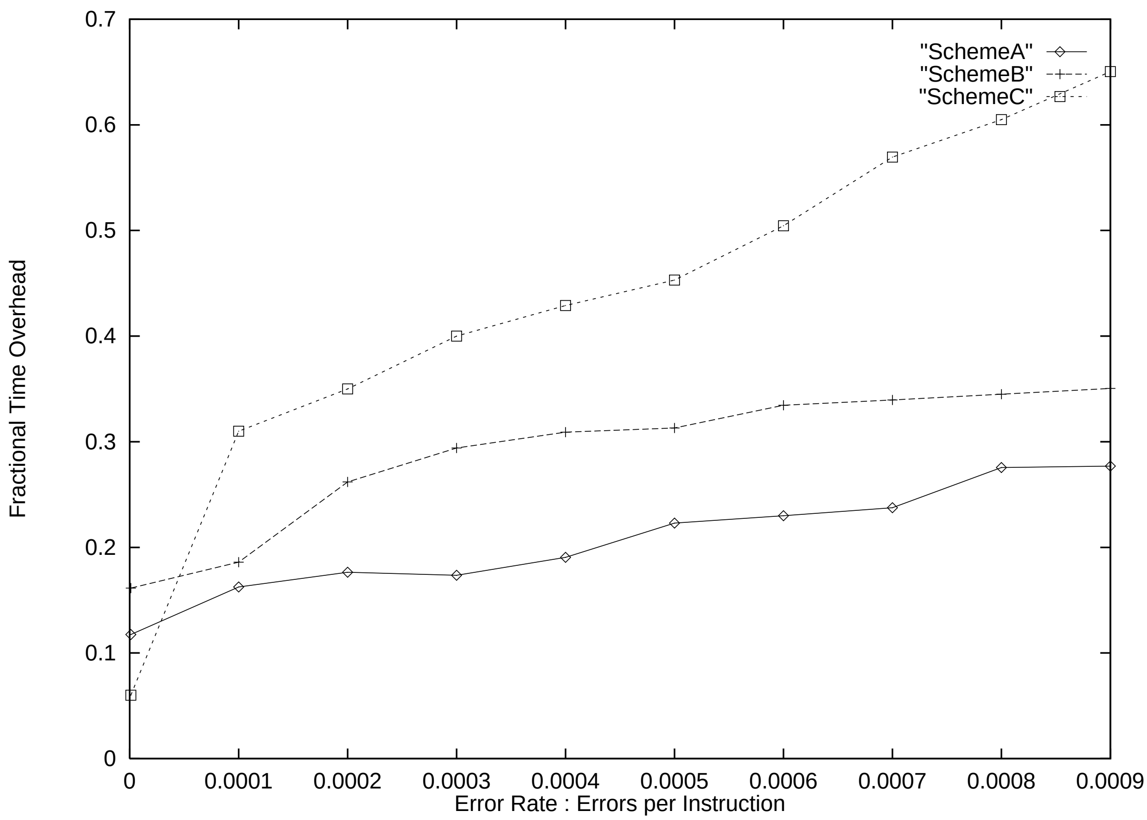


Figure 8: Completion Time Overhead Versus Error Rate – (Message Rate 0.01)

7 Comparison of Related Work

Most of the existing analysis in the literature are concerned with checkpointing and recovery schemes which do not include message validation as an integral part. We give a qualitative comparison of our scheme with schemes in [16] and [23] since these schemes use somewhat similar approaches to checkpointing and recovery as ours.

The most significant difference between our work and [16] is in the recovery process. In [16], when a process rolls back, this message is sent to all the processes. It is up to the individual process to decide whether the rollback message is relevant to it. In our scheme only the processes that need to rollback get the messages. The main advantage here is for systems that do not support broadcasting. The method in [16] also maintains a matrix for the states and updating the matrix requires the transmission of vectors every time a state change occurs in a process due to message reception. The global state matrix introduced in this paper is distinct from that of [16] in that it maintains both the error detection information as well as message dependencies.

In [23], the checkpointing scheme allows out of order reception whereas [16] requires FIFO order for the messages. Since messages are timestamped and every logged message is accompanied by an acknowledgment, the order in which messages are received is not critical in our method. However, in all our illustrations we assumed FIFO order of delivery for the sake of the ease of presentation. The work in [23] discusses in detail the checkpointing process but only a simple timeout mechanism is used for error detection. The scheme of [24] has similar goals as ours, viz., fast recovery and output commit. However, unlike using concurrent error detection as in ours, their scheme uses vector time to handle failures in distributed systems.

The global state matrix introduced in this paper provides a mechanism to systematically and efficiently keep track of the dependencies among the processes. It provides a basis for the analysis as well as for implementation of our scheme. Since each process is in charge of keeping track of dependencies due to it, updating and using a column corresponding to process P_i is done at process P_i. This structure yields conveniently to a distributed implementation of the scheme. There is no need for vectors getting passed around as in [16] or large antecendent graphs as in [7]. Though the validation messages represent an overhead, they are single bit messages and can be easily piggy-backed to regular messages to save bandwidth when communication intensity is high. Thus, our model based on a global state matrix and a validation mechanism simplifies the implementation and the operation of the checkpointing and recovery in distributed systems.

8 Summary

A comprehensive non-replication-based fault tolerant scheme applicable to distributed computing systems has been presented and the architecture, attributes and activities of its components are described. Algorithms that implement the various functionalities of the scheme are also given. Our approach is unique in that the error detection characteristics are exploited to provide fast recovery. Speedy recovery has been made possible by the integration of concurrent error detection into the recovery algorithm. In this paper, we used a signature-based concurrent error detection scheme, although any other extensive error detection mechanism with guaranteed low latency of detection can be used.

Several illustrations are used to show how the integration of detection points into the recovery scheme helps in controlling the primary and induced rollbacks. Analytical results that establish the correctness and completeness of the integrated recovery scheme are presented and the benefits of using this scheme under the transient fault model are demonstrated using computer simulation. The simulation has also brought out the overheads involved in concurrent error detection-based checkpointing. In addition to transient faults, our scheme can also be used to detect transients that may last long enough to be treated as a permanent fault. The analysis presented in this paper is expected to provide a vehicle for its implementation. The data structures and algorithms discussed in this paper can be readily used to implement the scheme in practice.

There are some issues which need further investigation. For example, the determination of bounds for the validation distance and average rollback distance. It is necessary to address task deadlines in the context of checkpoint placement in order to guarantee timely recovery. In this sense, our scheme is not applicable to real-time systems subject to hard deadlines, although it offers speedy recovery. The evaluation of scalability and further parameterization of the experiments to include the effect of multiple correlated errors, checkpointing strategy, optimal checkpoint interval and communication overhead will be a part of our future work. Such experiments can be done using the simulation support of OTEC.

The integration of concurrent error detection generally entails a preanalysis to signature the instruction stream and application code modification to embed the signatures. The signature generation can be automated by building a post-compiler. The preanalysis cost is incurred only once for a given application and this cost can be amortized by running the application a large number of times. We believe that our scheme is applicable to the domain of embedded control systems which employ tightly-coupled distributed processing and run applications repetitively. The low latency offered by the scheme is highly desirable in embedded systems where quick response is deemed critical. At the same time embedded systems provide convenient accessibility to internal structure necessary for the implementation of the scheme.

Acknowledgements

We thank Professor Ravi Iyer for making DEPEND available for our experiments.

References

[1] K. Chandy and C. Ramamoorthy, "Rollback and recovery strategies for computer programs," *IEEE Transactions on Computers*, vol. C-21, pp. 546–556, June 1972.

[2] D. Hunt and P. Marinos, "A general purpose cache-aided rollback error recovery (CARER) technique," *IEEE Int. Symposium on Fault Tolerant Computing*, pp. 170–175, July 1987.

[3] S. Upadhyaya and K. Saluja, "An experimental study to determine task size for rollback recovery systems," *IEEE Transactions on Computers*, vol. 37, pp. 872–877, July 1988.

[4] Y. Tamir, M. Liang, T. Lai, and M. Trembley, "The UCLA mirror processor: Al building block for self-checking self-repairing computer nodes," *21st Int. Symposium on Fault Tolerant Computing*, pp. 178–185, 1991.

[5] A. Mendelson and N. Suri, "Cache based fault recovery for distributed systems," *IEEE Int. Conference on Engineering of Complex Computer Systems*, pp. 119–129, Sept. 1997.

[6] B. Ramamurthy and S. Upadhyaya, "Watchdog processor-assisted fast recovery in distributed systems," *Fifth IEEE International Working Conference on Dependable Computing for Critical Applications*, pp. 125–134, September 1995.

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

[8] R. Iyer and D. Rossetti, "A measurement based model for workload dependence of CPU errors," *IEEE Transactions on Computers*, vol. C-35, pp. 511–519, June 1986.

[9] M. Namjoo, "Techniques for concurrent testing of VLSI processor operation," *Proc. International Test Conference*, pp. 461–468, November 1982.

[10] M. Schutte and J. Shen, "Exploiting instruction-level parallelism for integrated control-flow monitoring," *IEEE Transactions on Computers*, vol. 43, pp. 129–140, February 1994.

[11] S. Upadhyaya and B. Ramamurthy, "Concurrent process monitoring with no reference signatures," *IEEE Transactions on Computers*, vol. 43, pp. 475–480, April 1994.

[12] B. Ramamurthy and S. Upadhyaya, "Concurrent process monitoring in RISC based system," *IASTED Conference on Reliability, Quality and Risk Assessment*, pp. 117–120, October 1993.

[13] D. Pradhan, *Fault-Tolerant Computer System Design.* Upper Saddle River, NJ: Prentice-Hall Inc., 1996.

[14] L. Alvisi, B. Hoppe, and K. Marzullo, "Nonblocking and orphan-free message logging protocols," *Proceedings of IEEE Fault Tolerant Computing Symposium*, pp. 145–154, 1993.

[15] L. Alvisi and K. Marzullo, "Message logging: Pessimistic,, optimistic, causal, and optimal," *IEEE Transactions on Software Engineering*, vol. 24, no. 2, pp. 149–159, 1998.

[16] D. Johnson, "Efficient transparent optimistic rollback recovery for distributed application programs," *Proc. 12th Symposium on Reliable Distributed systems*, pp. 86–95, 1993.

[17] D. Johnson and W. Zwaenepoel, "Recovery in distributed systems using optimistic message logging," *Proceedings of the 7th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, August 1988.

[18] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, July 1978.

[19] J. R. Jump, *YACSIM Reference Manual.* ECE Dept, Rice University, 1993.

[20] K. K. Goswami, R. K. Iyer, and L. Young, "Depend: A simulation-based environment for system level dependability analysis," *IEEE Transactions on Computers*, vol. 46, pp. 60–74, January 1997.

[21] H. Schewetman, "CSIM: A C-based process-oriented simulation language," in *Proceedings Winter Simulation Conference*, 1986.

[22] B. Ramamurthy, S. Upadhyaya, and R. K. Iyer, "An object-oriented testbed for the evaluation of checkpointing and reco very systems," *Dig. Papers 27th Annual IEEE International Symposium on Fault Tolerant Com puting (FTCS)*, pp. 194–203, June 1997.

[23] B. Bhargava and S. R. Lian, "Independent checkpointing and concurrent rollback for recovery – An optimistic approach," in *Proceedings of the Seventh IEEE Symposium on Reliable Distributed systems*, pp. 3–12, October 1988.

[24] G. Richard and M. Singhal, "Using vector time to handle multiple failures in distributed systems," *IEEE Concurrency*, vol. 5, pp. 50–59, April 1997.