Checkpointing and Rollback Recovery in a Distributed System Using Common Time Base*

P. Ramanathan Kang G. Shin

Real-Time Computing Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, Michigan 48109-2122.

ABSTRACT

A new approach for checkpointing and rollback recovery in a distributed computing system using a common time base is proposed in this paper. First, a common time base is established in the system using a hardware clock synchronization algorithm. This common time base is coupled with the idea of pseudo-recovery block approach to develop a checkpointing algorithm that has the following advantages: (i) maximum process autonomy, (ii) no wait for commitment for establishing recovery lines, (iii) fewer messages to be exchanged, and (iv) less memory requirement.

Index Terms — Checkpointing, recovery block, rollback error recovery, domino effect, pseudo-recovery points, fault-tolerant clock synchronization, real-time systems.

1 Introduction

Concurrent processes in a critical real-time system are required to complete their execution prior to an imposed deadline. Failing to meet such a deadline could lead to catastrophic results. The deadlines cannot be easily met if the processes always have to restart their execution in case of failure. Hence, one of the issues in the design of a critical real-time system is to provide the capability of error recovery without having to restart the processes from the beginning.

The *recovery block* (RB) approach proposed in [1,2] is one way of recovering from an error without a restart. In this approach, concurrent processes save their states several times during their execution so that they can roll back to the most recently saved state and resume their execution in case of an error. Unfortunately, the rollback of a process can result in a cascade of rollbacks that can push the processes back to their beginnings, i.e., a *domino effect*. This results in the loss of entire computation done prior to the detection of the error. In order to avoid the domino effect, Randell proposed a *conversation scheme* in [2,3]. Kim also proposed a similar scheme but with more flexibility [4]. More recently, Koo and Toueg proposed a similar scheme that requires much less overhead as compared to the Randell's scheme [5].

In the conversation scheme, cooperating processes enter a *conversation* in order to interact with each other. Within a conversation, processes coordinate to save their states before they interact. Processes exit a conversation only after every process in the conversation has passed its *acceptance test*. If a process does not pass its acceptance test, then all the processes in the conversation roll back to the saved state and redo the computation using an alternative process. This rollback or checkpointing scheme usually results in an additional wait for the faster processes, thus degrading the utilization of processors on which the processes are executing. Furthermore, in order to indicate the passing of an acceptance test, each process has to broadcast a message to all the other processes in the conversation, thereby introducing additional overheads.

*The work reported here is supported in part by the NASA grant NAG-1-296 and the ONR contract N00014-85-K-0531. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

To overcome the under-utilization problem, a *pseudo-recovery block* (PRB) approach was proposed in [6]. In this approach, processes do not wait for each other to coordinate the saving of states. Instead, after passing an acceptance test, a process broadcasts a message to all other processes to establish a pseudo-recovery point. A *pseudo-recovery point* (PRP) is defined as a recovery point established *without* a preceding acceptance test. So, when a process receives a message to establish a PRP, it completes its current instruction and then saves its state without an acceptance test. Consequently, a *pseudo-recovery line* is created whenever a process establishes a recovery point. On detecting an error, processes roll back to a pseudo-recovery line that has been validated later by an acceptance test in each of the processes. Though there is no waiting for commitment in this scheme, substantial overheads in time and space are introduced because of the large number of PRP's each process has to establish.

A new approach is proposed in this paper to prevent the domino effect with little time and space overhead. The main idea in this approach is to establish and use a *common time base* in the system. The common time base is established by synchronizing the clocks of all the processors[1] in the system using a hardware synchronization algorithm[7,8,9]. This entails some additional hardware on each processor (see [9] for an analysis of additional hardware need) but imposes almost no time overhead on the system performance.

Unlike the conventional algorithms, the proposed approach does not require the programmer to coordinate the interaction between the cooperating processes. Instead, the common time base is used to coordinate the establishment of the PRP's. The execution of each process is first synchronized to the clock of the processor on which it is executing. An immediate consequence of this synchronous execution is that each instruction in a process takes a known number of clock cycles. This fact is coupled with the existence of the common time base to predict the relative behavior of the cooperating processes. The expected time for processes to reach their acceptance tests is first estimated[2] and then these estimated times are used to determine the *times* at which all the cooperating processes are asked to establish the pseudo-recovery points.

In between two PRP's each process is required to pass an acceptance test. A PRP is considered valid only after all the processes pass their acceptance test. If a process fails an acceptance test, then all processes roll back to a validated PRP. If a process does not execute an acceptance test before the time for the next PRP, then all the other processes must wait for the process to execute its acceptance test. Since the clocks of all processors are tightly synchronized and since the processes are synchronous to the clocks of the processors on which they are executing, the times for establishing the PRP's can be so chosen that a process will *almost always* execute an acceptance test before the next PRP. This fact is also used to reduce the number of messages processes have to exchange to establish a pseudo-recovery line. Note that this approach of coordinating the establishment of the PRP's is not feasible in the absence of a common time base since the times for processes to reach their acceptance tests in an asynchronous system cannot be estimated as accurately as in a synchronous system. Since it is not

[1] Each processor in the system is assumed to have its own clock.

[2] The actual time for a process to reach a given point in its execution could be substantially different from the estimated time due to the presence of loops, recursions, synchronization delays, etc.

difficult to equip the system with a common time base[7,8,9], use of a common time in checkpointing is an attractive alternative.

Since an acceptance test validates a PRP, each process has to establish only one PRP per pseudo-recovery line and preserve only two PRP's to be able to recover from an error without a cascade of rollbacks. This should be contrasted to the $n-1$ PRP's per recovery lines in the PRB approach, where n is the number of cooperating processes running concurrently on the system. The paper also evaluates the the average probability of a process sending a message or waiting for other processes to establish a checkpoint based on a general distribution for error in estimated times. Then, using a typical example, it is shown that even in the presence of a large error in the estimated times, the average probability of a process sending a message or waiting for other processes to establish a checkpoint is less than 3% (see Section 5). This should be contrasted to a 100% probability of sending a message and an almost 100% probability of wait in both the conversation scheme and the PRB approach.

This paper is organized as follows. The necessary assumptions, notation and terms are presented in Section 2. A checkpointing scheme that couples the idea of PRB approach with the presence of common time base is proposed in Section 3. Then, in Section 4, the checkpointing scheme is modeled probabilistically to analyze the various overheads involved. Based on this analysis, a numerical method for determining the optimum (in the sense to be defined) times for establishing the PRP's is also presented in Section 4. In Section 5, the expected overheads are evaluated numerically for some known distributions in the model to show that the overheads in this scheme are quite small as compared to the other checkpointing schemes. Finally, the paper concludes with a brief description of the merits and demerits of the proposed scheme in Section 6.

2 Basic System Architecture

This section describes the architectural support assumed in the rest of the paper. First, the proposed algorithm is based on the availability of a common time base. The common time base is established by synchronizing the clocks of all the processors in the system. For this purpose, each processor is assumed to have its own clock and is provided with a clock synchronization circuitry[7,8,9]. The circuitry is comprised of a reference signal generator, phase detector and a voltage controlled oscillator. The reference signal generator receives some of the other clocks in the system[9] and generates a reference signal depending on the hardware synchronization algorithm being used. The phase detector compares the phase of the reference signal with the phase of the oscillator output and outputs a voltage proportional to the phase error between the two signals. This output voltage is fed through a filter to the voltage controlled oscillator which then adjusts the frequency of operation depending on the magnitude of the error. The output of the oscillator serves as the clock of the processor.

The existence of a common time base is used to predict the relative behavior of the processes. Since the execution of a process is controlled by the clock of the processor in which it is executing and since the clocks of all the processors are kept in lock-step synchronization, the relative behavior of the cooperating processes can be predicted more easily and accurately than when the processors operate completely asynchronous of each other. This additional ability to predict the relative behavior of the processes is used as a key element to reduce time and space overhead of the proposed algorithm.

Unlike the conventional checkpointing algorithms, the proposed algorithm does not require the programmer to coordinate the interaction between the processes. Instead, the establishment of the pseudo-recovery points is coordinated using the common time base. For this purpose, each process is provided with a *pseudo-clock*. The pseudo-clock of each process can be thought of as a counter that normally increments at every pulse of the corresponding real clock. However, as will be described later, there are situations where the pseudo-clocks do not increment. In fact, there are situations in which the pseudo-clocks are forced to roll back instead of keeping up with real time.

The pseudo-clocks of all the processes are always kept tightly synchronized with respect to each other, because the clocks of all the processors are in lock-step synchronization. In addition, the checkpointing algorithm is such that whenever a pseudo-clock stops incrementing or whenever it rolls back to some previous values, all the other pseudo-clocks also do the same.

Each process is also provided with three interrupts: *Pseudo-Recovery Interrupt* (PRI), *Acceptance Test Interrupt* (ATI) and *Error Interrupt* (EI). A process receives a PRI when its pseudo-clock reaches a value R_j for some $j \in \{1, 2, ..., n\}$, where R_j's are clock values provided to the cooperating processes prior to their execution. It can thus be a hardware interrupt implemented with the help of a timer. On the other hand, the ATI is a software interrupt triggered when a process enters an acceptance test whereas EI is either a hardware or a software interrupt triggered whenever an error is detected in the system. It is generated in the process that detects the error and passed on to the other processes by sending messages.

To preserve process autonomy, PRI and ATI are generated locally in each of the process. As a result, they are not generated at the same time in all the processes. However, since the pseudo-clocks of all the processes are kept in tight synchrony by the checkpointing algorithm, the PRI's will be generated within a short time interval determined by the small skews between the synchronized clocks. That is, the time interval is equal to the maximum skew that exists between the pseudo-clocks of all the processes. The maximum skew between the pseudo-clocks is in turn determined by the maximum skews that exist between the real clocks and the maximum number of cycles it takes to execute an instruction in a process.

Unlike the PRI's, the ATI's in different processes do not necessarily occur within a short time interval. They are governed by the presence of loops, recursions, waiting times for shared resources, and other overheads in each of the processes. Hence, the proposed algorithm coordinates the establishment of the pseudo-recovery lines while retaining the asynchrony in the execution of the acceptance tests.

3 Proposed Checkpointing Scheme

Let P_1, P_2, ..., and P_N be the N cooperating processes in the system. They satisfy the following assumptions.

A1: Each process executes independently of others except when accessing shared resources.

A2: The processes are executing on reliable processors.

A3: The processes are synchronous to the clock of the processor on which they are executing.

A4: The programmer makes no effort to coordinate the interaction between the processes.

A1 is a requirement rather than an assumption. A2 is made for convienience of presentation. The proposed scheme tolerates software design faults and not hardware operational faults. Hardware operational faults like failure of processors on which the processes are executing can be tolerated by using either redundant or spare processors. Since the emphasis of this paper is not on tolerating such hardware failures, we assume that the processes are executing on reliable processors. A3 and A4 are based on the availability of the common time base and distinguish our approach from the other checkpointing algorithms. Let n_i be the number of acceptance tests in process P_i and let $n = \min_i n_i$. P_i chooses n out of its n_i acceptance tests for the checkpointing algorithm. Even though failure to pass any one of the n_i acceptance tests will trigger a rollback, only the chosen n acceptance tests will be considered by the checkpointing algorithm to establish the PRP's. The choice of these n acceptance tests does not have to be necessarily based on any criteria. However, the overheads involved in the checkpointing algorithm will be less if the processes choose their acceptance tests in such a manner that all processes execute a chosen acceptance test at the same time. So, to minimize the overheads, P_i should choose an acceptance test that is closest (in estimated time) to an acceptance test of a process P_j that has only n acceptance tests. For clarity of presentation, we will henceforth assume that all processes have the same number of acceptance tests, namely, the chosen n acceptance tests. In particular, we will assume that an ATI occurs only when a process enters one of the chosen acceptance tests.

The expected time for a process to reach each of its acceptance tests is estimated prior to its execution. This is made possible by the existence of a common time base and the fact that the processes are synchronous to the real-time clock. However, it is only "expected" because: (i) of the presence of loops and recursions, and (ii) the waiting times for shared resources, the overhead due to interrupts, etc., are not known *a priori*. Using these expected times, the time at which all processes establish their PRP's is calculated. The time for the j^{th} PRP is so chosen that all processes are expected to have completed their j^{th} acceptance test but not their $j+1^{th}$. Each process establishes its PRP when its pseudo-clock reaches the determined time (indicated by the arrival of a PRI).

However, before establishing a PRP, it checks whether it has executed an acceptance test since the last PRP. Since the times for establishing the PRP are appropriately chosen in accordance with the predicted behavior of the process, the process would have almost always executed an acceptance test since the last PRP. In the rare instances when it has not executed an acceptance test, the process broadcasts a message to all other processes in the system indicating it. On receiving such a message every process stops incrementing its pseudo-clock and waits for those processes that have not yet executed their acceptance test. Once they execute their acceptance test, messages are once again sent to all processes indicating it. After all processes have passed their acceptance test, processes start incrementing their pseudo-clocks and resume their normal operation. If a process fails its acceptance test, then all processes roll back to a PRP that has been later validated by an acceptance test. Similarly, before executing an acceptance test, each process checks whether or not it has established a PRP since the last acceptance test. If it has not established one, it waits till it receives the next PRI, establishes a PRP and then starts executing the acceptance test.

In order to describe this algorithm more formally, we define the following primitives:

receive(text, process_id): Receives the message *text* from the process whose identity is *process_id*.

broadcast(text, process_id): Process *process_id* broadcasts the message *text* to all processes.

wait(condition): Process halts until the condition becomes true.

The issue of implementing these three primitives in the presence of faults is non-trivial. There are several papers in literature that have addressed this issue [10,11]. In this paper we will assume the existence of such an implementation and concenterate on developing and analyzing a checkpointing algorithm using these primitives.

```
procedure pseudorecovery_interrupt;
    begin
        pseudo_clock_backup := pseudo_clock;
        if (not AT_flag) then
            begin
                slow := true;
                broadcast ("not completed AT", my_id);
                disable_clock := true;
                pseudo_clock := pseudo_clock_backup;
            end;
        else begin
            for k=1 to N, $k \neq i$
                if receive ("not completed AT", k) then
                    begin
                        receive_flag := true;
                        count := count + 1;
                    end;
            if (not receive_flag) then
                begin
                    checkpoint_valid := checkpoint_new;
                    checkpoint_new := current_state;
                end
            else begin
                disable_clock := true;
                pseudo_clock := pseudo_clock_backup;
                while receive_flag do
                    if receive ("completed AT", k) then
                        begin
                            count := count - 1;
                            if count=0 then receive_flag := false;
                        end;
                disable_clock := false;
                pseudo_clock := pseudo_clock_backup;
                end;
            end;
        end;
    AT_flag := false;
    end; /* pseudorecovery_interrupt */


procedure at_interrupt;
    begin
        if AT_flag then
            begin
                wait (pseudorecovery interrupt)
                execute (acceptance test)
            end
            else begin
                execute (acceptance test)
                if slow then
                    begin
                        broadcast ("completed AT", my_id);
                        AT_flag := true;
                        slow := false;
                        pseudorecovery_interrupt;
                    end;
                end;
    end; /* at_interrupt */


procedure error_interrupt;
    current_state := checkpoint_valid;
end /* error_interrupt */

procedure clock;
    if (not disable_clock) then
        increment ($C_i$);
end /* clock */
```

The procedures used in the above algorithm can be described informally as follows.

Procedure Pseudorecovery_Interrupt: This procedure is executed when a process receives a PRI, i.e., when its pseudo-clock reaches a time to set up a PRP. *AT_flag* indicates whether it has passed an acceptance test since the last PRP. If it has not passed an acceptance test, it sets the *slow* flag and broadcasts a message "not completed AT" to all the other processes. Otherwise, it checks the incoming messages to ensure that all other processes have also passed an acceptance test since the last PRP.

There are several ways of checking the incoming messages to ensure all other processes have passed their acceptance test. One easy way is to assume that all processes would have passed their acceptance test and proceed with normal execution. If there is at least one process that has not passed an acceptance test since the last PRP, then a message "not completed AT" will be sent to all other processes. On receipt of such a message, the processes roll back to the state at the time of last PRI. While doing so, they also roll back their pseudo-clocks to the time of the last PRI and disable its increment until it gets a message "completed AT" from all the slow processes.

Procedure AT_Interrupt: In order to prevent a process from running ahead of all the other processes, a process is allowed to establish only one acceptance test between any two successive pseudo-recovery lines. So if a process gets two ATI's before getting a PRI, then it has to wait for a PRI. This is ensured by checking the AT_flag variable whenever an ATI occurs.

If it is the first ATI after a PRI, then the process checks whether it is running slower than the others, i.e., are there some processes waiting for it to complete this acceptance (indicated by *slow*). If so, it broadcasts a "completed AT" message to all other processes. If all processes have finished their acceptance tests, then it proceeds with the normal execution, else it waits for others to finish. The detection of an error during the execution of an AT results in an Error Interrupt.

Procedure Error_Interrupt: This is executed whenever an error occurs in the system. The errors are detected during the execution of an acceptance test[3]. The procedure rolls back the processes to a valid state, i.e., a PRP that has been validated by an acceptance test, and then allows each of the processes to proceed from that point on by using an alternative path [2,3].

Procedure Clock: This procedure just describes the incrementing of pseudo-clocks. When the disable_clock is true (happens when one of the processes is waiting for some other process(es) to finish their acceptance test), the pseudo clock does not increment.

Figure 1 illustrates the algorithm for three processes P_1, P_2 and P_3. The thick (thin) lines indicate the estimated (actual) time for establishing the j^{th} and $j+1^{th}$ acceptance tests and the dotted line indicates the time at which the j^{th} PRI was issued. In Figure 1a, all processes complete their j^{th} acceptance test before receiving the j^{th} PRI and no process reaches the $j+1^{th}$ acceptance test before receiving the j^{th} PRI. So, no messages are sent and no process waits for another process to establish a checkpoint. In Figure 1b, P_1 does not complete its j^{th} acceptance test before receiving the j^{th} PRI. So it sends a message to the other two processes when it receives the j^{th} PRI. In Figure 1c, P_3 reaches its $j+1^{th}$ acceptance test before it receives the j^{th} PRI. It, therefore, waits till it receives the j^{th} PRI before executing the acceptance test. There are no messages exchanged in this situation.

The highlight of our approach lies in that the need of message exchange and waits in the above algorithm is minimized by the prediction of process execution behavior with a common time base. A detailed analysis of its performance is the subject of the next section.

4 Analysis of Checkpointing Scheme

In this section, a probabilistic model is developed to characterize the checkpointing scheme described in Section 3. This model will be used to analyze the expected overhead of the proposed scheme. To facilitate the analysis, we shall use the following notation.

n	Number of checkpoints.
AT_j	The j^{th} acceptance test.
PRP_j	The j^{th} pseudo-recovery point.
W_{ij}	Random variable representing the uncertainity in the expected time for P_i to reach AT_j.
X_{ij}	$W_{ij} - W_{ij-1}$.
T_{ij} (A_{ij})	Time required by P_i to reach AT_j without (with) the checkpointing overhead.
H_{ij}	Waiting time by P_i at PRP_j.
A^a_{ij} (A^d_{ij})	The real-time at which P_i receives (establishes)the j^{th} PRI.
fx (F_X)	Density (distribution) of a random variable X.
M_j	$\max_i \; W_{ij}$
m_j	$\min_i \; W_{ij}$
R_j	The pseudo-clock time at which all processes receive their j^{th} PRI.

[3]The chosen as well as the additional acceptance tests can trigger an EI.

4.1 Probabilistic Model

To model the performance of the proposed checkpointing scheme, we make the following assumptions.

MA1: W_{ij} and W_{kl}, $k \neq i$, are independent of each other for all j, l.

MA2: Given W_{ij}, the time required by P_i to reach AT_j without waiting for other processes during the checkpointing, denoted by, T_{ij}, has a density $f_{T_{ij}|W_{ij}}$.

MA3:

$$R_j = \begin{cases} M_j & \text{if } M_j > m_{j+1} \\ M_j + k_j * (m_{j+1} - M_j) & \text{if } m_{j+1} \geq M_j \end{cases}$$

where k_j is a design parameter.

MA4: $R_0 = A^d_{i0} = 0$ for all i.

The programmer inserts the acceptance tests within the processes. For each acceptance test, he will usually have a desired point (time) in the process where he would like to insert it. The desired point will be based on the number of acceptance tests he wants to insert and the estimated total execution time for the processes [12]. The acceptance tests cannot, however, be inserted at any arbitrary point in the process since one may not be exactly aware of the state a process should be in at every point in its execution (for verification by an acceptance test). So the acceptance tests will be inserted at a feasible point closest to the desired point. After insertion the expected time to reach an acceptance test will therefore differ from the desired time. We can model this uncertainity in the expected time to reach an acceptance test as a random variable distributed around the desired time for inserting that acceptance test. Section 5 illustrates this aspect by using a specific example.

MA1 states that the random variable representing the uncertainity in the expected time for P_i to reach AT_j is independent of that for P_k to reach the AT_l for all $k \neq i$. This arises from the assumption that each process executes independently of others except when accessing shared resources. MA2 states that due to unpredictable waits for shared resources and other overheads the actual time for P_i to reach its j^{th} acceptance test will differ from the expected time. The distribution of the actual time of an acceptance test around the expected time is defined by MA2. MA3 defines the class of functions considered in the analysis for determining the times to establish the PRP's. Since our checkpointing algorithm requires every process to have completed the j^{th} acceptance test and not the $j+1^{th}$, it is reasonable to assume that the time for the j^{th} PRI should be somewhere in between M_j and m_{j+1}. This particular class of functions is chosen to make the analysis tractable. MA4 specifies the initializing conditions.

There are three factors that contribute to the checkpointing overhead: (i) the overhead to save states, (ii) the overhead due to messages, and (iii) overhead due to waiting. The overhead due to saving of states depends directly on the number of times a process has to save its state. It can be reduced substantially at the cost of additional hardware as shown in [13]. The overhead due to messages occurs when a process does not complete its j^{th} acceptance test before it receives its j^{th} PRI, while the overhead due to waiting occurs when a process receives its $j+1^{th}$ ATI before it receives its j^{th} PRI. Since PRI's are based on time rather than points in execution whereas ATI's correspond to points in execution, the probability of a message being sent at the j^{th} PRP depends on the time interval between the start of the process and the j^{th} PRI. When the overhead due to the checkpointing scheme is zero, P_i executes for R_j time units before the j^{th} PRI. But in practice, due to the waiting times at the previous checkpoints, P_i gets $A^a_{ij} - \sum_{k=1}^{j-1} H_{ik}$ time units for execution before the j^{th} PRI. Since the processes prevent their pseudo-clocks from incrementing while waiting at the PRP's, $A^a_{ij} - \sum_{k=1}^{j-1} H_{ik} \geq R_j$. This result is proved more formally in Theorem 1. For proving the theorem it is convenient to define the following function.

Definition: Let C_i be a mapping from real-time to the pseudo-time on process P_i such that $C_i(t) = T$ means that the pseudo-time on P_i at real-time t is T.

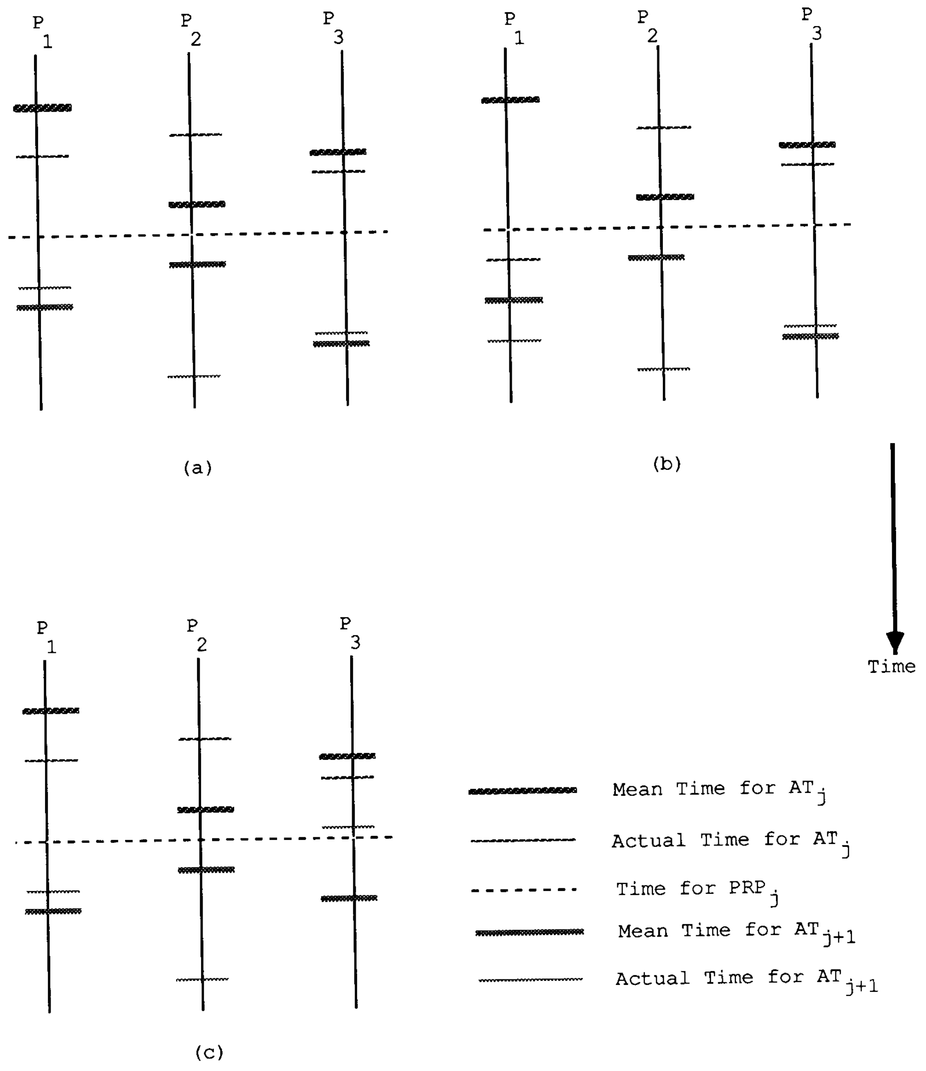

Figure 1: Illustration of checkpointing algorithm.

Theorem 1 *Let A^a_{ij} and R_j be the respective real-time and the pseudo-time at which process P_i receives the j^{th} PRI. Also let H_{ik} be the waiting time at the k^{th} PRP. Then, $A^a_{ij} - \sum_{k=1}^{j-1} H_{ik} \geq R_j$.*

Proof: Since a process might have to wait for some other processes (including itself) to complete the k^{th} acceptance test before establishing the k^{th} PRP, A^a_{ik} is not necessarily equal to A^d_{ik}, where A^d_{ik} is the real-time at which P_i establishes the k^{th} PRP. However, since the processes do not increment their pseudo-clocks while some process is waiting, $C_i(A^a_{ik}) = C_i(A^d_{ik})$ for all i.

In the time interval between the establishment of the k^{th} PRP and the receiving of the $k+1^{th}$ PRI, the pseudo-clocks of all processes keep up with real-time. As a result, $C_i(A^a_{ik+1}) - C_i(A^d_{ik}) = A^a_{ik+1} - A^d_{ik}$ for all i, k. Also, since a process that has not completed its k^{th} acceptance test when it received the k^{th} PRI continues to run while the others are waiting for it to complete the acceptance test, $H_{ik} \leq A^d_{ik} - A^a_{ik}$. From these observations, the theorem can be proved as follows.

$$\begin{aligned} A^a_{ij} &= \sum_{k=1}^{j}(A^a_{ik} - A^d_{ik-1}) + \sum_{k=1}^{j-1}(A^d_{ik} - A^a_{ik}) \\ &= \sum_{k=1}^{j}(R_k - R_{k-1}) + \sum_{k=1}^{j-1}(A^d_{ik} - A^a_{ik}) \\ &= R_j + \sum_{k=1}^{j-1}(A^d_{ik} - A^a_{ik}) \\ &\geq R_j + \sum_{k=1}^{j-1} H_{ik}. \end{aligned}$$

In other words, $A^a_{ij} - \sum_{k=1}^{j-1} H_{ik} \geq R_j$. **Q.E.D.**

The above theorem implies that the time interval between two successive PRI's does not depend on the waiting times at the previous PRI's. Consequently, the total time that a process gets to execute before receiving the j^{th} PRI does not depend on the waiting times at the previous PRI's. So the probability of a message being sent does not depend on the waiting times at each PRP. The probability of a message being sent can therefore be estimated by considering the situation in which there is no checkpointing overhead, i.e., the estimation of the probability of P_i sending a message can be done by using T_{ij} instead of A_{ij}.

4.2 Estimation of the Overhead

The overhead due to messages occurs when a process does not complete its j^{th} acceptance test before the time R_j while the overhead due to waiting occurs a process reaches its $j+1^{th}$ acceptance test before R_j. The overhead due to saving of states occurs each time a process has to establish a pseudo-recovery point.

The probability that P_i does not complete its j^{th} acceptance test before R_j can be expressed as $P\{T_{ij} > R_j\}$.

$$\begin{aligned} P\{T_{ij} > R_j\} &= \int_{z=0}^{\infty} P\{T_{ij} > z; R_j = z\}dz \\ &= \int_{z=0}^{\infty} \frac{\partial}{\partial u} P\{T_{ij} > z; R_j \leq u\}|_{u=z}dz, \end{aligned} \quad (1)$$

where

$$\begin{aligned} &P\{T_{ij} > t; R_j \leq z\} \\ &= \int_{t_1=0}^{\infty} P\left\{T_{ij} > t; M_j = t_1; m_{j+1} \leq \frac{z-(1-k_j)t_1}{k_j}\right\} dt_1 \\ &= \int_{t_1=0}^{\infty} \frac{\partial}{\partial u} P\left\{T_{ij} > t; M_j \leq u; m_{j+1} \leq \frac{z-(1-k_j)t_1}{k_j}\right\} |_{u=t_1} dt_1. \end{aligned} \quad (2)$$

Eq. (2) can be evaluated from the joint distribution of T_{ij}, M_j and m_{j+1}.

Since W_{ij} and W_{kl} are assumed to be independent for all $k \neq i$, the joint distribution of T_{ij}, M_j and m_{j+1} can be expressed as

$$\begin{aligned} &P\{T_{ij} \leq t; M_j \leq z_1; m_{j+1} \leq z_2\} \\ &= P\{EA_1A_2(B_1 + \overline{B_1B_2})\} \\ &= P\{EA_1B_1\}P\{A_2\}P\{EA_1\overline{B_1}\}P\{A_2\overline{B_2}\} \end{aligned} \quad (3)$$

where, $A_1 \equiv W_{ij} \leq z_1$, $A_2 \equiv (\forall\, l \neq i \;\; W_{lj} \leq z_1)$, $B_1 \equiv W_{ij+1} \leq z_2$, $B_2 \equiv (\forall\, l \neq i \;\; W_{lj} > z_2)$, and $E \equiv T_{ij} \leq t$. Each term in the right-hand side of Eq. (3) can be evaluated as follows:

$$\begin{aligned} &P\{EA_1B_1\} \\ &= \int_{t_1=0}^{z_1} P\{T_{ij} \leq t | W_{ij} = t_1\} P\{X_{ij+1} \leq z_2 - t_1\} P\{W_{ij} = t_1\} dt_1 \\ &= \int_{t_1=0}^{z_1} F_{T_{ij}|W_{ij}}(t|W_{ij} = t_1) F_{X_{ij+1}}(z_2 - t_1) f_{W_{ij}}(t_1) dt_1 \end{aligned} \quad (4)$$

$$P\{A_2\} = \prod_{\substack{l=1 \\ l\neq i}}^{N} (1 - F_{W_{lj}}(z_2)) \quad (5)$$

$$\begin{aligned} &P\{EA_1\overline{B_1}\} \\ &= \int_{t_1=0}^{z_1} P\{T_{ij} \leq t | W_{ij} = t_1\} P\{X_{ij+1} > z_2 - t_1\} P\{W_{ij} = t_1\} dt_1 \\ &= \int_{t_1=0}^{z_1} F_{T_{ij}|W_{ij}}(t|W_{ij} = t_1)(1 - F_{X_{ij+1}}(z_2 - t_1)) f_{W_{ij}}(t_1) dt_1 \end{aligned} \quad (6)$$

$$\begin{aligned} P\{A_2\overline{B_2}\} &= \prod_{\substack{l=1 \\ l\neq i}}^{N} F_{W_{lj}}(z_1) - \\ &\prod_{\substack{l=1 \\ l\neq i}}^{N} \left\{ \int_{t_1=0}^{z_1} F_{T_{lj}|W_{lj}}(t|t_1)(1 - F_{X_{lj+1}}(z_2 - t_1)) f_{W_{lj}}(t_1) dt_1 \right\}. \end{aligned} \quad (7)$$

Similarly, it is also possible to evaluate the probability of a process waiting at the $j+1^{th}$ acceptance test because it has not established the j^{th} PRP, i.e., $P\{\, T_{ij+1} \leq R_j \,\}$.

$$\begin{aligned} P\{T_{ij+1} \leq R_j\} &= \int_{z=0}^{\infty} P\{T_{ij+1} \leq z; R_j = z\}dz \\ &= \int_{z=0}^{\infty} \frac{\partial}{\partial u} P\{T_{ij+1} \leq z; R_j \leq u\}|_{u=z}dz, \end{aligned} \quad (8)$$

where

$$\begin{aligned} &P\{T_{ij+1} \leq t; \;\; R_j \leq z\} \\ &= \int_{t_1=0}^{\infty} P\{T_{ij+1} \leq t; M_j = t_1; m_{j+1} \leq \frac{z-(1-k_j)t_1}{k_j}\} dt_1 \\ &= \int_{t_1=0}^{\infty} \frac{\partial}{\partial u} P\{T_{ij+1} \leq t; M_j \leq u; m_{j+1} \leq \frac{z-(1-k_j)t_1}{k_j}\}|_{u=t_1} dt_1. \end{aligned} \quad (9)$$

Since W_{ij} and W_{kl} are independent for all $k \neq i$, the joint distribution of T_{ij+1}, M_j and m_{j+1} can be expressed as

$$\begin{aligned} &P\{T_{ij+1} \leq t; M_j \leq z_1; m_{j+1} \leq z_2\} \\ &= P\{E'A_1A_2(B_1 + \overline{B_1B_2})\} \\ &= P\{E'A_1B_1\}P\{A_2\} + P\{E'A_1\overline{B_1}\}P\{A_2\overline{B_2}\} \end{aligned} \quad (10)$$

where, $E' \equiv T_{ij+1} \leq t$. Among the four terms on the right-hand side of Eq. (10), $P\{A_2\}$ and $P\{A_2\overline{B_2}\}$ can be evaluated from Eqs. (5) and (7), respectively. The remaining two terms can be evaluated as follows:

$$\begin{aligned} P\{E'A_1B_1\} &= \int_{t_1=0}^{z_1}\int_{t_2=0}^{z_2-t_1} [P\{T_{ij+1} \le t | W_{ij} = t_1+t_2\} P\{X_{ij+1} = t_2\} \\ &\qquad P\{W_{ij} = t_1\} dt_2 dt_1] \\ &= \int_{t_1=0}^{z_1}\int_{t_2=0}^{z_2-t_1} [F_{T_{ij+1}|W_{ij}}(t|W_{ij} = t_1+t_2) f_{X_{ij+1}}(t_2) \\ &\qquad f_{W_{ij}}(t_1) dt_2 dt_1] \end{aligned} \tag{11}$$

$$\begin{aligned} P\{E'A_1\overline{B_1}\} &= \int_{0}^{z_1}\int_{z_2-t_1}^{\infty} [P\{T_{ij+1} \le t | W_{ij} = t_1+t_2\} P\{X_{ij+1} = t_2\} \\ &\qquad P\{W_{ij} = t_1\} dt_2 dt_1] \\ &= \int_{t_1=0}^{z_1}\int_{z_2-t_1}^{\infty} [F_{T_{ij+1}|W_{ij}}(t|W_{ij} = t_1+t_2) f_{X_{ij+1}}(t_2)) \\ &\qquad f_{W_{ij}}(t_1) dt_2 dt_1] \,. \end{aligned} \tag{12}$$

The overhead due to saving of states depends on: (i) the number of times a process has to save its state and (ii) the architecture of the system. In particular, it does not depend on R_j or any other parameter specific to the proposed algorithm. Since the proposed algorithm requires a process to save its states only once every pseudo-recovery line as compared to $N-1$ in the PRB approach (where N is the number of cooperating processes in the system), this overhead is substantially less in the proposed algorithm when compared with the PRB approach.

The above equations for various overheads can be used to determine an "optimal" (to be defined below) value for the design parameter k_j. Ideally, the optimal value for k_j should be such that the expected time overhead as a result of using the proposed checkpointing algorithm is minimum. However, since the time overhead due to the additional messages cannot be easily evaluated, we will use the weighted average of the probability of sending a message and probability of waiting as the objective function to be minimized. The design problem can thus be stated as:
Minimize

$$w_m * P\{T_{ij} > R_j\} + (1-w_m) * P\{T_{ij+1} \le R_j\} \quad \textbf{with respect to } R_j$$

Subject to:

$$R_j = \begin{cases} M_j & \text{if } M_j > m_{j+1} \\ M_j + k_j * (m_{j+1} - M_j) & \text{if } m_{j+1} \ge M_j \end{cases}$$

where k_j is a design parameter and $0 \le w_m \le 1$ is a weighting factor. Since k_j is the only design parameter in R_j, choosing value for R_j is equivalent to choosing a value for k_j. However, this problem is not analytically tractable even for simple distributions. Therefore, an optimal value for k_j has to be determined numerically as follows.

Partition the interval between the M_j and m_{j+1} into a finite number of intervals. A simple regular grid can be used for this partitioning. (One can also use more complicated partitioning schemes for better optimization [14].) Choose a value of k_j from each of these intervals. Compute the objective function for each of these chosen values of k_j. The value of k_j that gives rise to the minimum objective function can be approximated as the optimal value of k_j. This algorithm can be stated more formally as:

```
function objective($R_j$);
    return $w_m * P\{T_{ij} > R_j\} + (1-w_m) * P\{T_{ij+1} \le R_j\}$;


procedure optimal_$k_j$();
    begin
        if ($M_j < m_{j+1}$) then
            partition [$M_j$, $m_{j+1}$] into intervals $I_1, I_2, \ldots, I_l$
        else
            partition [$m_{j+1}$, $M_j$] into intervals $I_1, I_2, \ldots, I_l$;
    for some $k_j \in I_i, 1 \le i \le l$
        begin
            if ($M_j < m_{j+1}$) then
                $R_j := M_j$;
            else
                $R_j := M_j + k_j * (m_{j+1} - M_j)$;
            optk [i] := objective($R_j$);
        end
    return $min_i$ optk [i];
end.
```

5 Numerical Examples

The overheads described in the previous section were evaluated using numerical integration techniques for some known distribution and the following results were obtained. To account for differences in the processes under consideration, the expected time for processes to reach the j^{th} acceptance test was assumed to be normally distributed around $j * inter\_at$ with variance var_at, where $inter\_at$ and var_at are known parameters, i.e., $f_{W_{ij}}(t)$ is normal with mean $j * inter\_at$ and variance var_at. Given the expected time for a process to reach its j^{th} acceptance test, the actual time (without the checkpointing overhead) was also assumed to be normally distributed around the expected time with variance $j * var\_exe$, i.e., $f_{T_{ij}|W_{ij}}(t|W_{ij} = t_1)$ is normal with mean t_1 and variance $j * var\_exe$. This accounted for the variation in number of times certain loops were executed, waiting times for shared resources, interrupt service overhead, etc. This variance increases linearly with the number of checkpoint because longer the execution time, more the resources it has to access and hence greater the uncertainity.

The probability of a process sending a message at the sixth PRP for different values of k_j is shown in Figure 2. The figure clearly indicates the variation in this probability with changes in the parameters var_at and var_exe. As expected, the probability of a process sending a message increases with the var_exe for constant $inter\_at$ and var_at. However, the probability of a process sending a message does not always increase with var_at for constant $inter\_at$ and var_exe. As seen from Figure 2, when $k_j \approx 0$, the probability of sending a message decreases as the variance is increased whereas when $k_j \approx 1$ the probability increases with increasing variance. The reason for this behavior can be explained as follows.

Consider two variances var_at_1 and var_at_2 such that $var\_at_1 > var\_at_2$. For clarity of presentation, let us refer to the situation with var_at_1 and var_at_2 as S1 and S2, respectively, and let $k_j \approx 0$. This implies $R_j \approx M_j$. Since the spread of W_{ij}'s around $j * inter\_at$ is larger in S1 than in S2, more processes are likely to have their W_{ij}'s much smaller than M_j in S1 than in S2. Since

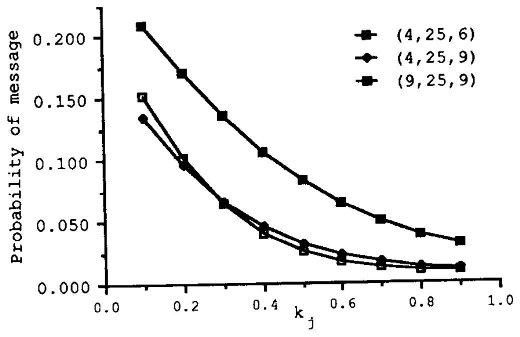

Figure 2: Probability of message vs. k_j.

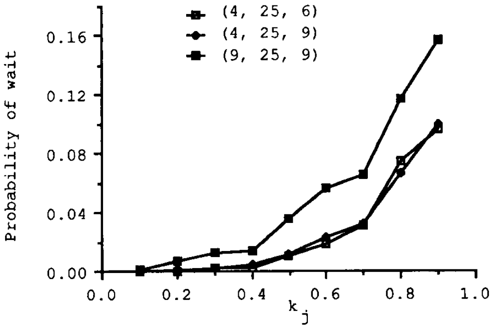

Figure 3: Probability of wait vs. k_j.

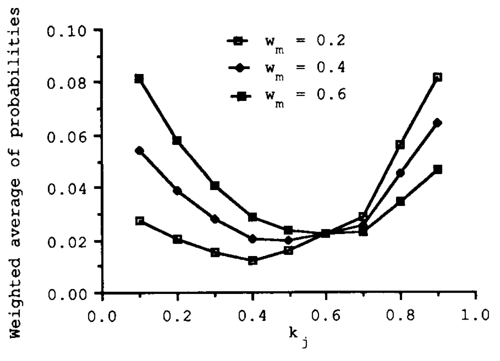

Figure 4: Weighted average of probabilities of message and wait vs. k_j.

var_exe is the same in both cases, and since smaller W_{ij} implies smaller $P\{T_{ij} > R_j\}$, the probability of a process sending a message is smaller in S1 than in S2. On the other hand, if $k_j \approx 1$, then $R_j \approx m_{j+1}$. Also m_{j+1} is more likely to be smaller in S1 than in S2. Similarly, M_j is more likely to be larger in S1 than in S2. So R_j will be closer to M_j in S1 than in S2. Hence, if $W_{ij} = M_j$, then P_i is more likely to send a message in S1 than in S2. When $R_j \approx m_{j+1}$, the probability of processes with $W_{ij} < M_j$'s is insignificant irrespective of *var_at*. Consequently, the probability of a process sending a message increases with *var_at* when $k_j \approx 1$.

The probability of a process waiting at the seventh acceptance test, because it has established its sixth PRP, is also shown in Figure 3. The figure clearly shows the variation in this probability with changes in the parameters *var_at* and *var_exe*. Here again, the probability of wait always increases with *var_exe* but increases with *var_at* when $k_j \approx 0$ and decreases with *var_at* when $k_j \approx 1$. The reasons are similar but the roles of $k_j \approx 0$ and $k_j \approx 1$ are reversed.

To illustrate the determination of the optimal values for k_j, the plots of weighted average of the probability of message and probability of wait for different weighting factors (w_m) are shown in Figure 4. The plot of the weighted average for optimum values of k_j against the number of checkpoints is shown in Figure 5. Figures 4 and 5 correspond to a *var_exe*, *inter_at* and *var_at* of 4, 25 and 9, respectively. From these plots it is clear that the weighted average of the probabilities of message and wait is only 2% in this example. This should be contrasted to the almost 100% probability of wait and 100% probability of message in the conversation

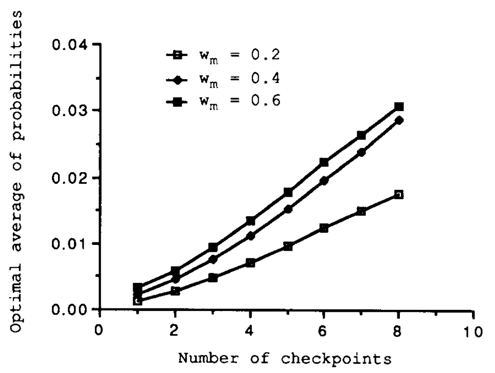

Figure 5: Weighted average of probabilities of message and wait vs. k_j.

scheme. Hence, the overheads involved in the checkpointing scheme proposed in this paper is far less than the other checkpointing schemes.

6 Conclusion

The checkpointing algorithm proposed in this paper has all the desirable features with little time and space overhead. Processes have to establish only one PRP per pseudo-recovery line and preserve only two PRP's. The paper also presented a model to evaluate the probabilities of exchanging messages or waiting for some other process (for checkpointing purposes). The average probability of exchanging messages or waiting for some other process was evaluated for a typical example and was shown to be only 2% for this example. A low probability of sending a message or waiting some other process is expected in most cases, although it may vary somewhat with the nature of cooperating processes.

The additional overheads in this scheme as compared to other schemes are (i) the need for a common time base and (ii) the need to know the expected times for reaching the acceptance tests *a priori*. If a hardware synchronization algorithm is used to establish the common time base, then the time overhead on the system is almost minimal. The cost of the additional hardware (see [9] for an analysis of hardware cost) is easily compensated by the reduced overhead in the checkpointing algorithm.

The expected times for reaching acceptance tests have to be estimated only once for every process. This can be easily done by executing the process repeatedly prior to their actual execution (mission). Since processes are repeatedly executed prior to the mission to ensure that there are no bugs in the program, these estimates can be obtained at no extra cost. Hence, the checkpointing algorithm proposed in this paper has high potential use for real-time applications.

Acknowledgement

The authors would like to thank the anonymous referees for their useful comments.

References

[1] J. Horning, H. C. Lauer, P. M. Melliar-Smith, and B. Randell, *A Program Structure for Error Detection and Recovery*, pp. 171–187, vol. 16 of *Lecture Notes in Computer Science*, New York: Springer-Verlag, 1974.

[2] B. Randell, "System Structure for Software Fault Tolerance", *IEEE Trans. Software Eng.*, vol. SE-1, no. 2, pp. 220–232, June 1975.

[3] B. Randell, P. A. Lee, and P. C. Treleaven, "Reliability Issues in Computing System Design", *ACM Comput. Surveys*, vol. 10, no. 2, pp. 123–165, June 1978.

[4] K. H. Kim, "Approaches to Mechanization of the Conversation Scheme Based on Monitors", *IEEE Trans. Software Eng.*, vol. SE-8, no. 3, pp. 189–197, May 1982.

[5] R. Koo, and S. Toueg, "Checkpointing and Rollback-Recovery for Distributed Systems", *IEEE Trans. Software Eng.*, vol. SE-13, no. 1, pp. 23–31, January 1987.

[6] K. G. Shin and Y.- H. Lee, "Evaluation of Error Recovery Blocks Used for Cooperating Processes", *IEEE Trans. Software Eng.*, vol. SE-10, no. 6, pp. 692–700, November 1984.

[7] J. L. W. Kessels, "Two Designs of a Fault-Tolerant Clocking System", *IEEE Trans. Comput.*, vol. C-33, no. 10, pp. 912–919, October 1984.

[8] C. M. Krishna, K. G. Shin, and R. W. Butler, "Ensuring Fault Tolerance of Phase-Locked Clocks", *IEEE Trans. Comput.*, vol. C-34, no. 8, pp. 752–756, August 1985.

[9] K. G. Shin and P. Ramanathan, "Clock Synchronization of a Large Multiprocessor System in the Presence of Malicious Faults", *IEEE Trans. Comput.*, vol. C-36, no. 1, pp. 2–12, January 1987.

[10] L. Lamport, "Using time instead of timeout for fault-tolerant distributed systems", *ACM Trans. Programming Languages and Systems*, vol. 6, no. 2, pp. 254–280, April 1984.

[11] I. Lee and S. B. Davidson, "Adding time to synchronous process communication", *IEEE Trans. Comput.*, vol. C-36, no. 8, pp. 113–124, August 1987.

[12] K. G. Shin, T.-H. Lin, and Y.-H. Lee, "Optimal Checkpointing of Real-Time Tasks", *IEEE Trans. Comput.*, vol. C-36, no. 11, pp. 1328–1341, November 1987.

[13] Y.- H. Lee and K. G. Shin, "Design and Evaluation of a Fault-Tolerant Multiprocessor Using Hardware Recovery Blocks", *IEEE Trans. Comput.*, vol. C-33, no. 2, pp. 113–124, February 1984.

[14] D. G. Luenberger, *Linear and Nonlinear Programming*, second ed., Addison Wesley, 1984.