Use of Common Time Base for Checkpointing and Rollback Recovery in a Distributed System

Parameswaran Ramanathan, *Member, IEEE,* and Kang G. Shin, *Fellow, IEEE*

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 points to develop a checkpointing algorithm that has the following advantages: 1) reduced wait for commitment for establishing recovery lines, 2) fewer messages to be exchanged, and 3) less memory requirement. These advantages are assessed quantitatively by developing a probabilistic model.**

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

I. 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. It would be difficult to meet the deadlines 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 [3], [12] 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 a 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 the entire computation done prior to the detection of the error. In order to avoid the domino effect, Randell [12], [13] proposed a *conversation scheme.* Kim [5] also proposed a similar scheme but with more flexibility.

In the conversation scheme, cooperating processes enter a *conversation* before interacting with each other. Within a conversation, processes save their states before they interact. During a conversation, a process is not allowed to interact with any other process that does not belong to the conversation. 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 processes in the conversation roll back to the saved state and redo the computation using an alternative algorithm.

Although this scheme results in a good abstraction for the programmer, there are several disadvantages. First, the faster processes in a conversation have to wait for the slower processes to complete their acceptance test. Second, a process outside the conversation will have to wait for the end of the conversation if it wants to interact with a process within the conversation. Third, processes have to exchange messages solely for checkpointing purposes. For instance, in order to ensure that all processes in a conversation have passed their acceptance tests, each process has to broadcast a message to all other processes in the conversation to indicate that it has successfully passed its acceptance test. The end result of these three factors is an underutilization of the processors on which the processes are executing.

To overcome this underutilization problem, Shin and Lee [14] proposed a *pseudo-recovery block* (PRB) approach. 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 asking them to save their states immediately. On receiving such a message, a process completes its current instruction and then saves its states without an acceptance test. This ensures that all processes save their states almost at the same time. However, the saved state may be potentially erroneous because most of the processes did not execute an acceptance test. Shin and Lee [14] refer to this coordinated (but potentially erroneous) set of states as a *pseudo-recovery point* (PRP). Upon detection of an error, processes roll back to a PRP that has been validated later by an acceptance test in each of the processes. Although there is no waiting for commitment in this scheme, substantial overheads in time and space are introduced because each process has to retain many PRP's in order to ensure a successful rollback in case of error. In addition, each process has to send a large number of messages solely for checkpointing purposes.

Another approach for coordinating the establishment of the checkpoints was proposed by Koo and Toueg [6]. In their approach, a process that wishes to establish a checkpoint executes a two-phase commit protocol. In the first phase, the process

Manuscript received July 29, 1990. This work was supported in part by NASA Grant NAG-1-296 and ONR Contract N00014-85-K-0122. 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. An earlier version of this paper appeared in Proceedings of the Symposium on Reliable Distributed Systems, 1988. Recommended by D. Reed.

P. Ramanathan is with the Department of Electrical and Computer Engineering, University of Wisconsin–Madison, Madison, WI 53706-1691.

K. G. Shin is with the Real-Time Computing Laboratory, Computer Science and Engineering Division, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI 48109-2122.

IEEE Log Number 9202418.

tentatively saves its state and requests all other processes to do the same. The process then waits for the response from other processes. If all the "relevant" processes indicate that they are also willing to establish a checkpoint, then the tentative checkpoint is made permanent. Otherwise, the checkpoint is discarded. In the second phase, the decision to commit or discard the tentative checkpoint is conveyed to the other processes and they immediately carry out the decision. Since all or none of the processes commit to a checkpoint, a process has to retain a maximum of two saved states. However, the worst-case time for completing the protocol is proportional to the total number of processes in the system and to the upper bound on the communication delay between any two processes. Since a process cannot send any noncheckpointing message while it is waiting for the protocol to terminate, the waiting overhead of this approach can be fairly large.

In contrast to the foregoing approaches, a new scheme is proposed in this paper to prevent the domino effect with little time and space overhead. This scheme is intended for systems that have a *common time base*. A common time base can be established by synchronizing the clocks of all the processors[1] in the system using a hardware synchronization algorithm [4], [7], [16]. This entails additional hardware on each processor (see [16] for an analysis of additional hardware needed) but imposes almost no time overhead on the system performance.

Unlike conventional algorithms, individual processes are coordinated in saving their states by using the common time base. The expected time for processes to reach their acceptance tests is first estimated;[2] these estimated times are then used to determine the *times* at which all the cooperating processes are asked to save their states. If the times for saving the states are selected properly, then we show that the overheads imposed by the checkpointing algorithm can be reduced considerably. In particular, each process has to retain only *two* sets of states to be able to restart after an error without a cascade of rollbacks. This should be contrasted to $n-1$ set of states each process has to retain in the PRB scheme where n is the total number of processes concurrently running on the system. Moreover, as shown later in the paper, the probability of a process sending a message solely for checkpointing purposes and the expected waiting time at each checkpoint is very small as compared with other algorithms. For a typical numerical example, the expected waiting time for the proposed algorithm is only about 10–15% of the expected wait time in the conversation scheme [12].

The paper is organized as follows. Section II presents an informal description of the proposed scheme. The necessary assumptions, notation, and terms are presented in Section III. A checkpointing scheme that couples the idea of a PRB approach with the presence of common time base is proposed in Section IV. Then, in Section V, the checkpointing scheme is modeled probabilistically to analyze the various overheads involved. Based on this analysis, a numerical method for determining the optimal (in the sense to be defined) times for establishing the PRP's is also presented in this section. In Section VI, the expected overheads are evaluated numerically for some known distributions in the model. For that example, it is shown that the overheads in the proposed approach are much smaller than the other schemes. Finally, Section VII concludes the paper by briefly describing the merits and demerits of the proposed scheme.

[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 loops, recursions, synchronization delays, etc.

II. Informal Description of the Proposed Scheme

Since the proposed checkpointing scheme is based on the PRB approach, we begin by briefly describing the PRB scheme as presented in [14].

A. PRB Scheme

Unlike a conversation-based scheme, each process in the PRB scheme is allowed to interact with any other process without any restrictions. In addition, each process is allowed to execute an acceptance test as, and when, it is appropriate. This is significant because the programmer can independently choose suitable points in each process to test for correctness. However, in order to ensure a consistent rollback in case of an error, each process sends a message to all other processes whenever it successfully completes an acceptance test. On receiving this message, the other processes save their states immediately without executing any acceptance test. In case of an error, all processes roll back to a state that has later been shown to be error free by an acceptance test in each process.

For example, consider a system of three processes, P_1, P_2, and P_3 shown in Fig. 1. In this figure, the acceptance tests by the individual processes are shown by the rectangular boxes. Corresponding to an acceptance test in each process, a dotted line represents the coordinated saving of states among all processes. However, this coordinated set of states is not guaranteed to be error free because all but one process have not executed an acceptance test before saving their states. Therefore, if P_2 detects an error at a time indicated by $\times$ in the figure, the processes cannot roll back to the most recently saved state because no other process except P_3 has validated that state. Hence, all processes roll back to the state corresponding to the acceptance test by process P_1. This state can be assumed to be error free because all processes have executed at least one acceptance test since saving that state.

It is clear from the foregoing example that a PRB scheme requires processes to save their states quite often. Furthermore, each process has to retain a large number of states so that it can roll back to an error-free state when an error is detected. The proposed checkpointing scheme eliminates these two drawbacks by using the common time base.

B. Proposed Checkpointing Scheme

The basic idea of the proposed checkpointing scheme is to have the programmer first estimate the expected time for processes to reach each of their acceptance tests. This can possibly be done by observing the processes during program development. It is only an estimate because the programmer

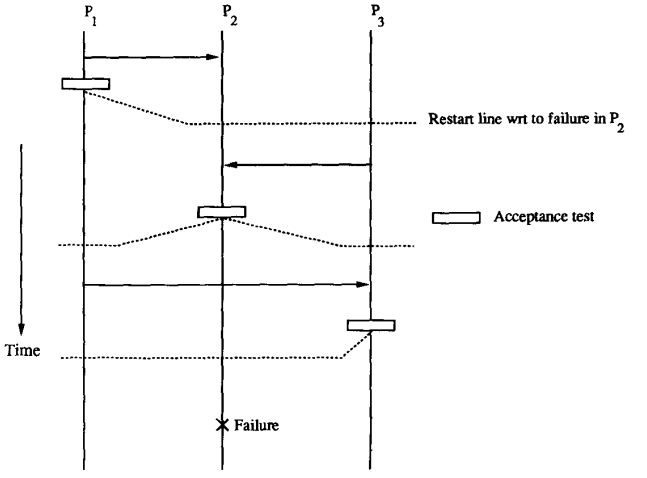

Fig. 1. Example of a PRB checkpointing scheme.

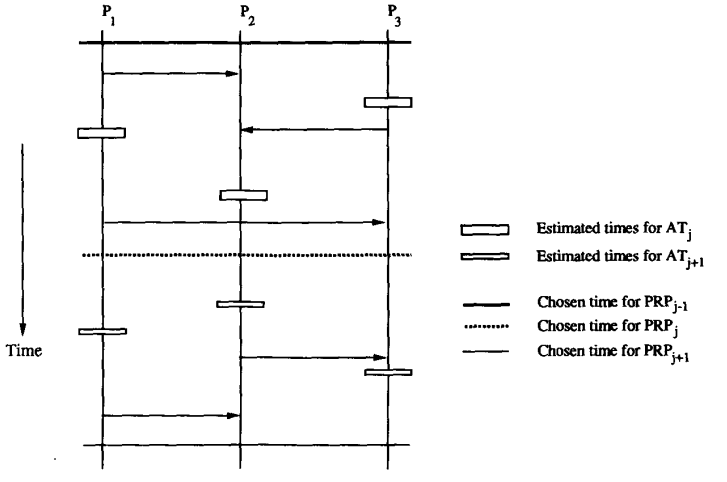

Fig. 2. Example of a system with three cooperating processes.

has no way of determining *a priori* the effect of the actual environment on the number of loops, recursion, synchronization overheads, resource contention, etc. Based on these estimated times, the programmer selects the *times* at which all processes establish a pseudo-recovery point (PRP). When the local clock of a process reaches one of the selected times, the process saves its states without executing an acceptance test. Due to the presence of a common time base, the local clocks of all processes are tightly synchronized and therefore all processes establish a PRP at almost the same time.

The time for establishing the jth PRP is chosen in such a way that all processes are expected to have completed their jth acceptance test by that time. If a process has not completed its jth acceptance test by that time, then all other processes wait for that process to complete its jth acceptance test. As a result, no process establishes two consecutive PRP's without an acceptance test between them. Similarly, no process is allowed to execute two consecutive acceptance tests without a PRP in between them. That is, if a process reaches the $(j+1)$th acceptance test before the time for the jth PRP, the process just waits until it is time for the next PRP and then saves its state before proceeding with its normal execution. The end result of these two conditions is that in each process an acceptance test alternates with a PRP. If the time for establishing the PRP is chosen as described later in this paper, then it is shown that in most cases there will be no need for a process to wait before establishing a PRP. Furthermore, in most cases there will be no need for processes to exchange messages for checkpointing purposes.

For example, consider a system of three processes shown in Fig. 2. For each process the figure shows the estimated times for the jth and the $(j+1)$th acceptance tests (denoted by AT_j and AT_{j+1}, respectively) and the times selected by the programmer for establishing the $(j-1)$th, jth, and $(j+1)$th PRP (denoted by PRP_{j-1}, PRP_j, and PRP_{j+1}, respectively). Figs. 3–6 show various possible scenarios that can arise

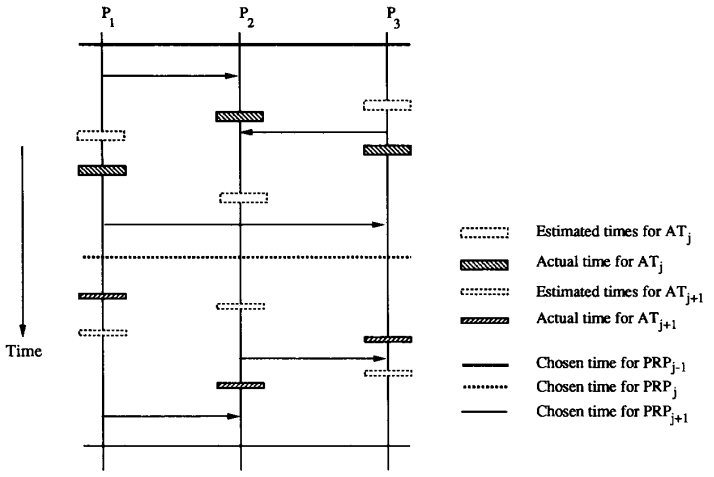

Fig. 3. Most likely scenario in the proposed scheme.

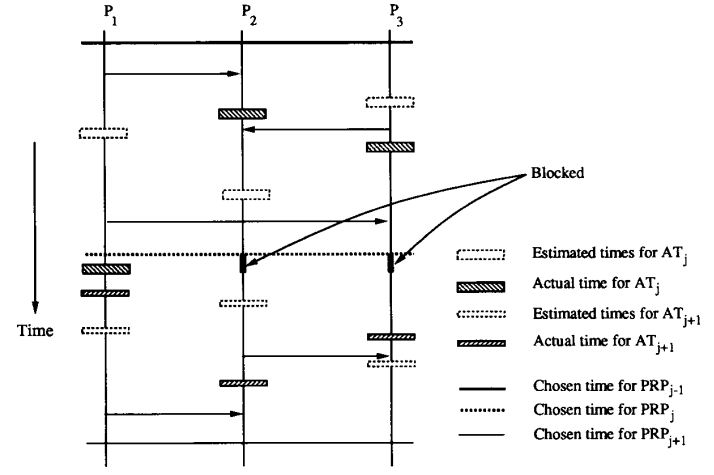

Fig. 4. Scenario in which a process is slow.

during an actual execution of these three processes. The scenarios arise because the time at which a process reaches its acceptance tests differs from the estimated times.

Fig. 3 shows the most likely scenario in which all processes have completed their jth acceptance test prior to the time for the jth PRP. In this scenario, no messages are exchanged for checkpointing purposes and no process waits for another process to establish a PRP. Fig. 4 shows the scenario in which P_1 is slow and therefore has not completed its jth acceptance test by the time for the jth PRP. In this scenario, P_1 sends a message to other processes asking them to wait for it to catch up. As soon as P_1 completes its jth acceptance test all the processes establish their jth PRP and proceed with their normal execution. Note that this requires other fault-tolerance mechanisms such as timeout to ensure that a fault in P_1 does not permanently hold up other nonfaulty processes. Such fault-tolerance mechanisms are not unique to our scheme; they are also needed in other checkpointing algorithms such as a conversation scheme because each process in a conversation has to make sure other processes in that conversation have passed their acceptance test.

Fig. 5 shows the scenario in which P_3 is so fast that it reaches the $(j+1)$th acceptance test before the time for the jth PRP. As a result, P_3 waits until it is time for the jth PRP, saves its state, and then proceeds with its normal execution. The other processes are unaffected by this. As mentioned earlier, if the programmer selects the time for establishing the jth PRP appropriately, then the first scenario will occur more frequently than the latter two.

Fig. 6 shows the behavior of the processes when a failure is detected by P_3. Processes cannot roll back to the most recently saved PRP because it has not necessarily been validated by an acceptance test in all processes. However, by the very nature of the algorithm, the second to last PRP is guaranteed to have been validated by an acceptance test in all processes. Therefore, all processes roll back to the second to last PRP and

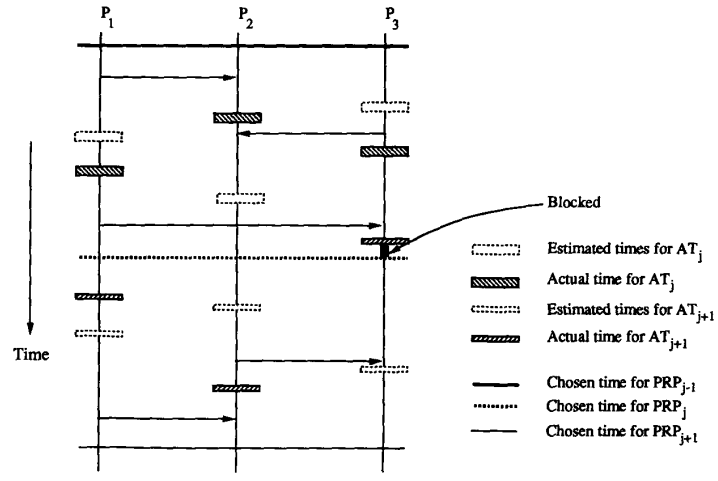

Fig. 5. Scenario in which a process is fast.

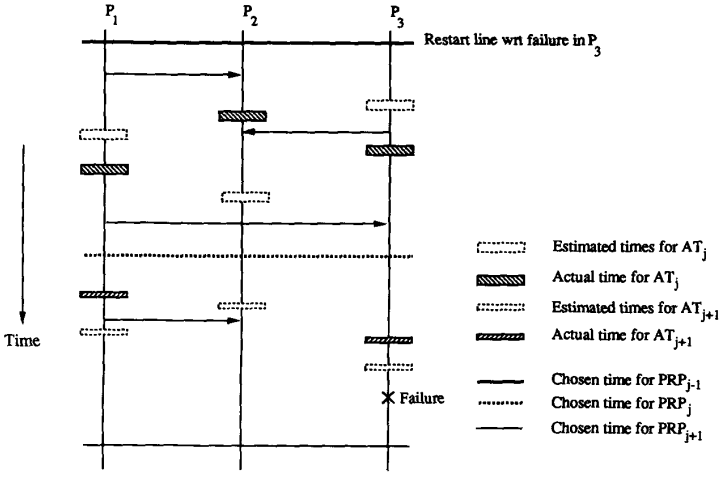

Fig. 6. Scenario in which a process detects a failure.

redo their computation. Observe that processes do not have to exchange messages to determine the state to which they roll back. Furthermore, each process has to retain only *two* PRP's to be able to successfully roll back in case of an error.

III. 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 (as is usually the case) and is provided with a clock synchronization circuitry [4], [7], [16]. The circuitry is composed 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 [16] 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 asynchronously 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.

In order to coordinate the establishment of PRP's, 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 we describe 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.

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 previous values, all the other pseudo-clocks also do the same.

Each process is also provided with four interrupts: *Pseudo-Recovery Interrupt* (PRI), *Acceptance Test Interrupt* (ATI), *Error Interrupt* (EI), and *Timer Interrupt* (TI). A process receives a PRI when its pseudo-clock reaches a value R_j for some $j \in \{1, 2, \cdots, n\}$, 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, 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 during the execution of an acceptance test. It is generated in the process that detects the error and passed on to the other processes by sending messages. A TI is generated when an alarm set by a process expires before the process cancels the alarm.

To preserve process autonomy, PRI and ATI are generated locally in each process. As a result, they are not generated at the same time in all the cooperating 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 pseudo-clocks is thus 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 requiring only a loose synchronization in the execution of the acceptance tests.

IV. Formal Description of the Proposed Scheme

The main goal of the scheme proposed in this paper is to reduce the amount of waiting and the number of messages exchanged between the processes for checkpointing purposes. Presented as follows is a formal description of the proposed scheme.

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

A1) The processes are synchronous in the sense that there is an integer constant $\Phi \geq 1$ such that in any time interval in which some nonfaulty process takes $\Phi + 1$ "steps," every nonfaulty process must take at least one step.

A2) There is a known upper bound U on the communication delays between any two processes.

A3) The processes are executing on reliable processors.

A1) is based on the availability of the common time base. In this assumption, a "step" is defined by the granularity of our abstraction. For instance, a step can either be an instruction or a procedure or a set of procedures depending on the level of our abstraction. A2) is required to distinguish between a failed process not sending a message and unusually large communication delays. It is well-known that any form of synchronization is difficult to achieve, if not impossible, in the presence of both faulty processes and unbounded communication delays [2]. A1) and A2) are consistent with definition of processor and communication synchronism in [1]. A3) is made for convenience of presentation. Over and above the proposed checkpointing scheme, one would need other fault-tolerant mechanisms such as timeouts, redundancy, and the like. A checkpointing-and-rollback recovery scheme does not obviate the need for redundant or spare processors since a rollback recovery might need a redundant or spare processor to execute alternate algorithms in case of a permanent hardware failure.

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. However, it is only expected because the presence of loops and recursions and waiting times for shared resources, the overhead due to interrupts and the like 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 jth PRP is chosen so that all processes are expected to have completed their jth 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, a process checks whether it has passed 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 passed an acceptance test since the last PRP. In the rare instances when it has not passed an acceptance test, the process broadcasts a message to all other processes in the system indicating that. On receiving such a message, every process stops incrementing its pseudo-clock and waits for those processes that have not yet passed their acceptance test. After they pass their acceptance test, messages are once again sent to all processes indicating it. After all processes have passed their acceptance tests, processes start incrementing their pseudo-clocks and resume their normal operation. 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 our 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.
- *Alarm* (*interval*): Cancel any previously-set alarm and generate a timer interrupt after *interval* time units have elapsed. If *interval* has value 0, then do not generate any interrupt but cancel all previously-set alarms.

The issue of implementing these three primitives in the presence of faults is nontrivial. There are several papers in literature that have addressed this issue [8], [9]. Here we assume the existence of such an implementation and concentrate on developing and analyzing a checkpointing algorithm using these primitives.

```
procedure pseudorecovery_interrupt;
  δ: Maximum skew between pseudo-clocks of
     nonfaulty processes;
  U: Upper bound on communication delay;
  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
      wait(alarm(δ + U));
      for k=1 to N, k ≠ i
        if receive ("not completed AT", k)
        then
          begin
            receive_flag := true;
            count := count + 1;
            alarm (Max_Willing_To_Wait);
          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
                begin
                  receive_flag := false;
                  alarm (0);
                end
            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;
  Invalidate checkpoint_new;
end /* error_interrupt */

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

The procedures used in the foregoing 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 "not completed AT" message 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 that all other processes have passed their acceptance test. A simple, but slightly inefficient, way is to halt all processing for $\delta + U$ time units, where δ is the maximum skew between pseudo-clocks of any two nonfaulty processes and U is an upper bound on communication delay between any two nonfaulty processes. This wait ensures receipt of all "not completed AT" messages because the corresponding PRI's in all processes will occur within δ time units of each other and U is the maximum time required to deliver a "not completed AT" message from one process to all other processes. A more efficient way of checking for "not completed AT" messages is to set an alarm for $\delta+U$ time units and let each process proceed with its computation as long as there is no (noncheckpointing) message to be sent. When the alarm expires, each process can independently check their incoming buffers for "not completed AT" messages from other processes.

If a process receives a "not completed AT" message, it rolls back its pseudo-clock to the time of the PRI and disable its increment. It also stops all execution until it receives "completed AT" message from all the slow processes. To prevent a failed process from permanently blocking a nonfaulty process, each process sets an alarm for a duration of Max_Willing_to_Wait on receiving the "not completed AT" message. If a "completed AT" message is not received within this duration, then slow process is considered to have failed and a recovery action such as a rollback is undertaken.

It is not easy to determine an optimal value for Max_Willing_to_Wait. Choosing a small value would cause more (slow) nonfaulty processes to be considered faulty, whereas choosing a large value will cause nonfaulty processes to wait for a long time in case a process is faulty. However, it is not difficult in practice to choose reasonably good values depending on the exact nature of the cooperating processes.

At each PRP, the processes must have a consistent view of messages in the system. A message viewed as "sent" by one process should be viewed as "received" by all processes that are supposed to receive that message. Similarly, a message viewed as "received" by one process should be viewed as "sent" by the process that sent the message. If these two conditions are not satisfied at a PRP, the system will be in an inconsistent state if a rollback occurs to that PRP. To ensure this consistency, all messages sent by a process between its jth and $(j + 1)$th PRP are tagged with j to indicate the interval in which they are sent. Processes use this tag to determine whether or not a message should be part of their jth PRP. Only messages with tag $(j - 1)$ are included in the jth PRP. This technique guarantees the latter consistency condition because a message is viewed as "received" in the jth PRP only if it has a tag of $j - 1$, which means that the message was sent prior to the establishment of the jth PRP in the sending process. Moreover, since processes wait for $\delta + U$ time units after receiving a PRI before establishing a PRP, this technique also guarantees the first consistency condition.

Procedure AT_Interrupt: In order to prevent a process from running ahead of all the others, a process is allowed to establish only one acceptance test between any two successive pseudo-recovery lines. Thus, 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, otherwise it waits for others to finish.

Procedure Error_Interrupt: This procedure is executed whenever an error occurs in the system. Errors are detected during the execution of an acceptance test.[3] The procedure rolls back the processes to a valid state and then allows each of the processes to proceed from that point on by using an alternative path [12], [13]. If an error is detected somewhere in between the jth and $(j + 1)$th PRI, the processes roll back to the states corresponding to the $(j - 1)$th PRI because the proposed algorithm guarantees an acceptance test between the $(j - 1)$th and the jth PRI. Consequently, there is no need to interact with other processes to determine the state to which a process has to roll back. This simplifies the rollback procedure to a great extent.

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

V. Analysis of a Checkpointing Scheme

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

n	Number of checkpoints
AT_j	The jth acceptance test
PRP_j	The jth pseudo-recovery point.
W_{ij}	Random variable representing the uncertainty in the expected time for P_i to reach AT_j
T_{ij} (A_{ij})	Time required by P_i to reach AT_j without (with) the checkpointing overhead
S_j	$\max_i W_i j$
O_{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 jth PRI
f_X (F_X)	Density (distribution) of a random variable X
M_j	$\max_i W_i j$
m_j	$\min_i W_i j$
R_j	Pseudo-clock time at which all processes receive their jth PRI.

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

A. 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 or she will usually have a desired point (time) in the process where each test could be inserted. The desired point will be based on the number of acceptance tests the programmer wants to insert and the estimated total execution time for the processes [15]. 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). Thus 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 uncertainty in the expected time to reach an acceptance test as a random variable distributed around the desired time for inserting that acceptance test. Section VI illustrates this aspect by using a specific example.

MA1) states that the random variable representing the uncertainty 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 $k \neq i$. MA2) states that due to unpredictable waits for shared resources and other overheads, the actual time for P_i to reach its jth 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 jth acceptance test and not the $j+1$th, it is reasonable to assume that the time for the jth 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 two factors that contribute to the checkpointing overhead: overhead of saving states, and overhead of waiting at each pseudo-recovery point. The overhead of saving 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 [10]. A process waits at the jth pseudo-recovery point either if one of the n processes does not complete its jth acceptance test before the jth PRI or if the process receives the $j+1$th ATI before the jth PRI. In the first case, a process waits because some process is slower than expected, whereas in the second case the process waits because it is much faster than expected. We henceforth refer to the first case as a *slow_wait* and the second case as the *fast_wait*.

Since PRI's are based on time rather than points in execution whereas ATI's correspond to points in execution, the probability of a *slow_wait* depends on the time interval between the start of the process and the jth PRI. When the overhead due to the checkpointing scheme is zero, P_i executes for R_j time units before the jth PRI. But in practice, due to the waiting times at the previous checkpoints, P_i gets $A^a_{ij} - \sum_{k=1}^{j-1} O_{ik}$ time units for execution before the jth PRI. Since the processes prevent their pseudo-clocks from incrementing while waiting at the PRP's, $A^a_{ij} - \sum_{k=1}^{j-1} O_{ik} \geq R_j$. This result is proved 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.

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 jth PRI. Also let O_{ik} be the P_i's waiting time at the kth PRP. Then, $A^a_{ij} - \sum_{k=1}^{j-1} O_{ik} \geq R_j$.

Proof: Since a process might have to wait for some other processes (including itself) to complete the kth acceptance test before establishing the kth 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 kth 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 between the establishment of the kth PRP and the receiving of the $k+1$th PRI, the pseudo-clocks of all processes keep up with real time. As a result,

$$C_i(A^a_{ik+1}) - C_i(A^d_{ik}) = A^a_{ik+1} - A^d_{ik} \text{ for all } i, k.$$

Also, since a process that has not completed its kth acceptance test when it received the kth PRI continues to run while the others are waiting for it to complete the acceptance test, $O_{ik} \leq A^d_{ik} - A^a_{ik}$. From these observations, the theorem can be proved as follows.

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

In other words, $A^a_{ij} - \sum_{k=1}^{j-1} O_{ik} \geq R_j$.

The foregoing 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 jth 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 probability of P_i sending a message can be estimated by using T_{ij} instead of A_{ij}.

B. Estimation of the Overhead

The overhead due to *slow_wait* occurs when a process does not complete its jth acceptance test before the time R_j while the overhead due to *fast_wait* occurs when 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.

In terms of the notation introduced earlier, the wait time at the jth PRP of process P_i can be expressed as

$$O_{ij} = \begin{cases} S_j - T_{ij} & \text{if } T_{ij} > R_j \\ S_j - R_j & \text{if } S_j > R_j, T_{ij} \le R_j \text{ and } T_{ij+1} > R_j \\ S_j - T_{ij+1} & \text{if } S_j > R_j \text{ and } T_{ij+1} \le R_j \\ R_j - T_{ij+1} & \text{if } S_j \le R_j \text{ and } T_{ij+1} \le R_j \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

The first case, namely $T_{ij} > R_j$, corresponds to the situation where P_i does not complete its AT_j before receiving the jth PRI. P_i therefore waits from the time it completes its AT_j till the slowest process completes. The second case corresponds to situation where P_i completes its jth acceptance test before the jth PRI, but some other process does not. So P_i waits for the slowest to complete from the time it receives the j PRI. The third and the fourth cases occur when P_i is so fast that it completes both AT_j and AT_{j+1} before receiving the jth PRI. In the third case, a process other than P_i does not complete its AT_j before the jth PRI. In this case, P_i waits from the time it receives the $j+1$th ATI till the slowest process completes its jth acceptance test. In the fourth case, all processes complete their AT_j by R_j and therefore P_i resumes its normal execution at the time R_j. Finally, if none of the above situation occurs, then P_i does not wait at the jth pseudo-recovery point. The goal is to select a R_j such that the final case occurs more often than the other four cases.

We can then prove the following result about the expected value of O_{ij}, i.e., the expected wait time of P_i at the jth PRP.

Theorem 2:

$$E[O_{ij}] \le E[T_{ij} - R_j | T_{ij} > R_j] + E[R_j - T_{ij+1} | R_j > T_{ij+1}].$$

Proof: See Appendix.

Although exact analytic expressions for the expected overhead can be derived, evaluating those expressions to within an acceptable accuracy is very complicated. Thus, we derive analytic expressions for the upper bound in Theorem 1 instead of attempting to derive the exact expressions.

$E[T_{ij} - R_j | T_{ij} > R_j]$ can be evaluated from the joint distribution of T_{ij} and R_j as follows.

$$P\{T_{ij} \le t; R_j \le z\} = \int_{t_1=0}^{z} P\{T_{ij} \le t; m_{j+1} \le \frac{z - (1-k_j)t_1}{k_j} | M_j = t_1\} f_{M_j}(t_1) dt_1. \tag{5.2}$$

Equation (5.2) 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

$$P\{T_{ij} \le t; M_j \le z_1; m_{j+1} > z_2\} = P\{T_{ij} \le t; W_{ij} \le z_1; W_{ij+1} > z_2\} \cdot P\{\forall l \neq i \; W_{lj} \le z_1; W_{lj+1} > z_2\}$$

where

$$P\{\forall l \neq i \; W_{lj} \le z_1; W_{lj+1} > z_2\} = \prod_{\substack{l=1 \\ l \neq i}}^{N} \int_{t_1=0}^{z_1} P\{W_{lj+1} > z_2 | W_{lj} = t_1\} f_{W_{lj}}(t_1) dt_1$$

and

$$P\{T_{ij} \le t; W_{ij} \le z_1; W_{ij+1} > z_2\} = \int_{t_1=0}^{z_1} P\{T_{ij} \le t | W_{ij} = t_1\} P\{W_{ij+1} > z_2 | W_{ij} = t_1\} \cdot f_{W_{ij}}(t_1) dt_1.$$

Similarly, it is possible to evaluate the $E[R_j - T_{ij+1} | R_j > T_{ij+1}]$ from the joint distribution of T_{ij+1} and R_j.

$$P\{T_{ij+1} \le t; R_j \le z\} = \int_{t_1=0}^{z} P\{T_{ij+1} \le t; m_{j+1} \le \frac{z - (1-k_j)t_1}{k_j} | M_j = t_1\} \cdot f_{M_j}(t_1) dt_1. \tag{5.3}$$

Equation (5.3) can be evaluated from the joint distribution of T_{ij+1}, 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

$$P\{T_{ij+1} \le t; M_j \le z_1; m_{j+1} > z_2\} = P\{T_{ij+1} \le t; W_{ij} \le z_1; W_{ij+1} > z_2\} \cdot P\{\forall l \neq i \; W_{lj} \le z_1; W_{lj+1} > z_2\}$$

where,

$$P\{\forall l \neq i \; W_{lj} \le z_1; W_{lj+1} > z_2\} = \prod_{\substack{l=1 \\ l \neq i}}^{N} \int_{t_1=0}^{z_2} P\{W_{lj} \le z_1 | W_{lj+1} = t_1\} f_{W_{lj+1}}(t_1) dt_1$$

$$P\{T_{ij+1} \le t; W_{ij} \le z_1; W_{ij+1} > z_2\} = \int_{t_1=0}^{z_2} P\{T_{ij+1} \le t | W_{ij+1} = t_1\} \cdot P\{W_{ij} \le z_1 | W_{ij+1} = t_1\} f_{W_{ij+1}}(t_1) dt_1.$$

The overhead due to saving of states depends on: 1) the number of times a process has to save its state and 2) 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 [14], where N is the number of cooperating processes in the system, this overhead is substantially less in the proposed algorithm than in the PRB approach.

The foregoing equations can be used to determine a good value for the design parameter k_j. This value of k_j would have been optimal if we use the exact expressions (instead of the upper bound) for the $E[O_{ij}]$ and if our objective is to minimize the expected wait time. On the other hand, if our objective is to minimize a different objective like the probability of a long wait or the probability of a process sending a message, then we need to derive and minimize the appropriate analytic expressions. Although the basic probabilistic model used in this paper is general enough to accommodate almost any objective, we will presume that the objective is to select a k_j that minimizes the expected wait time at the jth PRP. Since evaluating the exact expressions for the expected wait time is very complicated even for some simple distributions, we will minimize the upper bound specified in Theorem 1 to obtain a sub-optimal solution. In other words,

Minimize
$E[S_j - R_j | S_j > R_j] + E[R_j - T_{ij+1} | T_{ij+1} \leq R_j]$ **with respect to** R_j
Subject to:

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

where k_j is a design parameter. Since k_j is the only design parameter in R_j, choosing a value for R_j is equivalent to choosing a value for k_j. The value of k_j that minimizes the foregoing objective can be determined numerically using iterative optimization techniques as in [11].

VI. 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 jth acceptance test was assumed to be uniformly distributed over the interval $[j * inter\_at - a_j,\ j * inter\_at + a_j]$, where a_j is a known parameter. Given the expected time for a process to reach its jth acceptance test, the actual time (without the checkpointing overhead) was assumed to be uniformly distributed around the expected time with parameter b_j, i.e., $f_{T_{ij}|W_{ij}}(t|W_{ij} = t_1)$ is uniformly distributed over the interval $[t_1 - b_j,\ t_1 + b_j]$. This accounted for the variation in number of times certain loops were executed, waiting times for shared resources, interrupt service overhead, etc.

The expected waiting time at the sixth PRP for the best value of k_j (obtained by solving the minimization problem in Section V-B) is shown in Table I. The values in this table correspond to a $inter\_at$ of 25. The minimization problem for determining the best k_j was solved using the Fibonacci descent method. This method would lead to the optimal solution if the objective being minimized is unimodal. Otherwise, the results would be upper bounds to the actual value.

TABLE I
Expected Overheads in the Proposed Checkpointing Scheme

a_6	b_6	$E[O_{ij}]$ (Prop.)	$E[O_{ij}]$ (Rand.)	$\frac{\text{Prop.}}{\text{Rand.}}$ %	Prob. of mess. %
6.0	12.0	1.32	15.80	13.87	4.14
8.0	12.0	1.95	17.25	11.30	4.86
9.0	12.0	2.79	18.03	15.47	5.95
10.0	12.0	4.42	18.82	23.48	5.76
12.0	12.0	6.56	20.49	32.01	5.04
8.0	10.0	1.22	15.41	7.90	2.61
8.0	12.0	1.95	17.25	11.29	4.86
8.0	14.0	2.62	19.14	13.71	7.70
8.0	16.0	2.92	21.07	13.87	7.65
8.0	18.0	4.06	23.03	17.64	13.53

The table also shows the variation in the expected waiting times with changes in the parameters a_6 and b_6. It is clear from the table that the expected waiting time increases with a_6 for a constant b_6. This is because an increase in a_6 corresponds to a greater variance between the processes, and hence it is more difficult to coordinate the completion of acceptance tests by the processes. Similarly, an increase in b_6 results in an increase in the expected waiting time. This is because an increase in b_6 implies that our estimate of the execution time to reach the jth differs more from the actual execution time. For the purpose of comparison, the table also contains the expected wait time in Randell's checkpointing scheme [12]. Even for large values of a_6 and b_6 the expected wait times in the proposed checkpointing scheme is much less than in Randell's scheme. For example, $a_6 = 12$, $b_6 = 12$ and $inter\_at = 25$ corresponds to the case where the actual execution time to reach the sixth acceptance could vary between 126 time units to 174 time units. Even for this severe variation in the actual execution time, the expected wait time is only 32% of the expected wait time in Randell's scheme.

In addition to the reduced wait times, the proposed scheme also has fewer message exchanges for checkpointing purposes. Column 6 in Table I shows the probability of process sending a message at the sixth PRP for the best value of k_j. These values should be contrasted to a 100% probability of message exchange in Randell's scheme.

VII. 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 expected waiting time and the probability of exchanging messages for checkpointing purposes. For a typical numerical example the expected waiting times and the

probability of a process exchanging messages are shown to be much less than Randell's checkpointing scheme [12].

The additional overheads in this scheme as compared to others are 1) the need for a common time base and 2) 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 [16] for an analysis of hardware cost) is easily compensated by the reduced overhead in the checkpointing algorithm. Moreover, as was pointed out in [8], this common time base can be used to efficiently handle problems other than checkpointing.

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 usually 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.

APPENDIX

Theorem 2:
$E[O_{ij}] \leq E[T_{ij} - R_j | T_{ij} > R_j] + E[R_j - T_{ij+1} | R_j > T_{ij+1}]$.

Proof: From (5.1),

$$\begin{aligned} E[O_{ij}] = {} & E[S_j - T_{ij} | T_{ij} > R_j] P\{T_{ij} > R_j\} \\ & + E[S_j - R_j | S_j > R_j, T_{ij} \leq R_j, T_{ij+1} > R_j] \\ & \quad \cdot P\{S_j > R_j, T_{ij} \leq R_j, T_{ij+1} > R_j\} \\ & + E[S_j - T_{ij+1} | S_j > R_j, T_{ij+1} \leq R_j] \\ & \quad \cdot P\{S_j > R_j, T_{ij+1} \leq R_j\} \\ & + E[R_j - T_{ij+1} | S_j \leq R_j, T_{ij+1} \leq R_j] \\ & \quad \cdot P\{S_j \leq R_j, T_{ij+1} \leq R_j\}. \end{aligned}$$

Since

$$E[S_j - T_{ij} | T_{ij} > R_j] \leq E[S_j - R_j | T_{ij} > R_j]$$

and

$$\begin{aligned} E[S_j - T_{ij+1} | S_j > R_j, T_{ij+1} \leq R_j] = {} & \\ E[S_j - R_j | S_j > R_j, T_{ij+1} \leq R_j] & \\ + E[R_j - T_{ij+1} | S_j > R_j, T_{ij+1} \leq R_j] & \end{aligned}$$

we get,

$$\begin{aligned} E[O_{ij}] \leq {} & E[S_j - R_j | T_{ij} > R_j] P\{T_{ij} > R_j\} \\ & + E[S_j - R_j | S_j > R_j, T_{ij} \leq R_j, T_{ij+1} > R_j] \\ & \quad \cdot P\{S_j > R_j, T_{ij} \leq R_j, T_{ij+1} > R_j\} \\ & + E[S_j - R_j | S_j > R_j, T_{ij+1} \leq R_j] \\ & \quad \cdot P\{S_j > R_j, T_{ij+1} \leq R_j\} \\ & + E[R_j - T_{ij+1} | S_j > R_j, T_{ij+1} \leq R_j] \\ & \quad \cdot P\{S_j > R_j, T_{ij+1} \leq R_j\} \\ & + E[R_j - T_{ij+1} | S_j \leq R_j, T_{ij+1} \leq R_j] \\ & \quad \cdot P\{S_j \leq R_j, T_{ij+1} \leq R_j\}. \end{aligned}$$

But $T_{ij} > R_j$ implies $S_j > R_j$ and $T_{ij+1} > R_j$. Therefore,

$$\begin{aligned} E[O_{ij}] \leq {} & E[S_j - R_j | T_{ij} > R_j] \\ & \quad \cdot P\{S_j > R_j, T_{ij} > R_j, T_{ij+1} > R_j\} \\ & + E[S_j - R_j | S_j > R_j, T_{ij} \leq R_j, T_{ij+1} > R_j] \\ & \quad \cdot P\{S_j > R_j, T_{ij} \leq R_j, T_{ij+1} > R_j\} \\ & + E[S_j - R_j | S_j > R_j, T_{ij+1} \leq R_j] \\ & \quad \cdot P\{S_j > R_j, T_{ij} \leq R_j, T_{ij+1} \leq R_j\} \\ & + E[R_j - T_{ij+1} | S_j > R_j, T_{ij+1} \leq R_j] \\ & \quad \cdot P\{S_j > R_j, T_{ij+1} \leq R_j\} \\ & + E[R_j - T_{ij+1} | T_{ij+1} \leq R_j] \\ & \quad \cdot P\{S_j \leq R_j, T_{ij} \leq R_j, T_{ij+1} \leq R_j\}. \end{aligned}$$

Furthermore,

$$\begin{aligned} E[S_j - R_j | S_j > R_j] = {} & \\ E[S_j - R_j | T_{ij} > R_j] & \\ \cdot P\{S_j > R_j, T_{ij} > R_j, T_{ij+1} > R_j\} & \\ + E[S_j - R_j | S_j > R_j, T_{ij} \leq R_j, T_{ij+1} > R_j] & \\ \cdot P\{S_j > R_j, T_{ij} \leq R_j, T_{ij+1} > R_j\} & \\ + E[S_j - R_j | S_j > R_j, T_{ij+1} \leq R_j] & \\ \cdot P\{S_j > R_j, T_{ij} \leq R_j, T_{ij+1} \leq R_j\} & \end{aligned}$$

$$\begin{aligned} E[R_j - T_{ij+1} | T_{ij+1} \leq R_j] = {} & \\ E[R_j - T_{ij+1} | S_j > R_j, T_{ij+1} \leq R_j] & \\ P\{S_j > R_j, T_{ij+1} \leq R_j\} & \\ + E[R_j - T_{ij+1} | T_{ij+1} \leq R_j] & \\ P\{S_j \leq R_j, T_{ij} \leq R_j, T_{ij+1} \leq R_j\}. & \end{aligned}$$

Hence,

$$\begin{aligned} E[O_{ij}] \leq {} & E[S_j - R_j | S_j > R_j] \\ & + E[R_j - T_{ij+1} | R_j > T_{ij+1}]. \end{aligned}$$

Simplifying further,

$$\begin{aligned} E[S_j - R_j | S_j > R_j] = {} & \\ \sum_{i=1}^{N} E[T_{ij} - R_j | T_{ij} > R_j, T_{ij} \equiv S_j] P\{T_{ij} \equiv S_j\}. & \end{aligned}$$

Since all processes are equally probable of being the slowest $P\{T_{ij} \equiv S_j\} = 1/N$. In other words,

$$\begin{aligned} E[O_{ij}] \leq {} & E[T_{ij} - R_j | T_{ij} > R_j] \\ & + E[R_j - T_{ij+1} | R_j > T_{ij+1}]. \end{aligned}$$

REFERENCES

[1] D. Dolev, C. Dwork, and L. Stockmeyer, "On the minimal synchronism needed for distributed consensus," *J. ACM*, vol. 34, no. 1, pp. 77–97, Jan. 1987.

[2] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *J. ACM*, vol. 32, no. 2, pp. 374–382, Apr. 1985.

[3] J. Horning, H. C. Lauer, P. M. Melliar-Smith, and B. Randell, "A program structure for error detection and recovery," *Lecture Notes in Computer Science*, vol. 16. New York: Springer-Verlag, 1974, pp. 171–187.

[4] J. L. W. Kessels, "Two designs of a fault-tolerant clocking system," *IEEE Trans. Comput.*, vol. C-33, no. 10, pp. 912–919, Oct. 1984.
[5] K. H. Kim, "Approaches to mechanization of conversation scheme based on monitors," *IEEE Trans. Software Eng.*, vol. SE-8, no. 3, pp. 189–197, May 1982.
[6] R. Koo and S. Toueg, "Checkpointing and rollback-recovery for distributed systems," *IEEE Trans. Software Eng.*, vol. SE-13, no. 1, pp. 23–31, Jan. 1987.
[7] 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, Aug. 1985.
[8] L. Lamport, "Using time instead of timeout for fault-tolerant distributed systems," *ACM Trans. Programming Languages Syst.*, vol. 6, no. 2, pp. 254–280, Apr. 1984.
[9] I. Lee and S. B. Davidson, "Adding time to synchronous process communications," *IEEE Trans. Comput.*, vol. C-36, no. 8, pp. 941–948, Aug. 1987.
[10] 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, Feb. 1984.
[11] D. G. Luenberger, *Linear and Non-linear Programming*, 2nd ed. Reading, MA: Addison Wesley, 1984.
[12] B. Randell, "System structure for software fault tolerance," *IEEE Trans. Software Eng.*, vol. SE-1, no. 2, pp. 220–232, June 1975.
[13] 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.
[14] 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, Nov. 1984.
[15] 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, Nov. 1987.
[16] 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, Jan. 1987.

Parameswaran Ramanathan (S'85–M'89) received the B.Tech degree from the Indian Institute of Technology, Bombay, India, in 1984, and the M.S.E. and Ph.D. degrees from the University of Michigan, Ann Arbor, in 1986 and 1989, respectively.

He is an Assistant Professor in the Department of Electrical and Computer Engineering, Department of Computer Sciences, University of Wisconsin, Madison. He has been active in the area of fault-tolerant computing, distributed systems, real-time computing, VLSI design and computer architecture. From 1984 to 1989 he was a Research Assistant in the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor.

Kang G. Shin (S'75–M'78–SM'83–F'92) received the B.S. degree in electronics engineering from Seoul National University, Seoul, Korea, in 1970, and both the M.S. and Ph.D degrees in electrical engineering from Cornell University, Ithaca, NY, in 1976 and 1978, respectively.

He is Professor and Chair of Computer Science and Engineering Division, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor. He has authored/coauthored over 180 technical papers (more than 85 of these in archival journals) and several book chapters in the areas of distributed real-time computing and control, fault-tolerant computing, computer architecture, and robotics and automation. In 1985, he founded the Real-Time Computing Laboratory, where he and his colleagues are currently building a 19-node hexagonal mesh multicomputer, called HARTS, to validate various architectures and analytic results in the area of distributed real-time computing. From 1978 to 1982 he was on the faculty of Rensselaer Polytechnic Institute, Troy, NY. He has held visiting positions at the U.S. Airforce Flight Dynamics Laboratory, AT&T Bell Laboratories, Computer Science Division within the Department of Electrical Engineering and Computer Science at U.C. Berkeley, and International Computer Science Institute, Berkeley, CA.

In 1987, he received the Outstanding IEEE TRANSACTIONS ON AUTOMATIC CONTROL Paper Award for a paper on robot trajectory planning. In 1989, he also received the Research Excellence Award from The University of Michigan. He was the Program Chairman of the 1986 IEEE Real-Time Systems Symposium (RTSS), the General Chairman of the 1987 RTSS, the Guest Editor of the 1987 August special issue of IEEE TRANSACTIONS ON COMPUTERS on Real-Time Systems, and is a Program Co-Chair for the 1992 International Conference on Parallel Processing. He currently chairs the IEEE Technical Committee on Real-Time Systems, is a Distinguished Visitor of the Computer Society of the IEEE, an Editor of IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, and an Area Editor of *International Journal of Time-Critical Computing Systems*.