CHECKPOINTING WITH SYNCHRONIZED CLOCKS IN DISTRIBUTED SYSTEMS

S. Neogy[1] **A. Sinha[1]** **P. K. Das[2]**

[1]Department of Computer Science & Engg., Jadavpur University, India
sarmisthaneogy@gmail.com
[2] Faculty of Engg. & Tech., Mody Institute of Technology & Science, India

ABSTRACT

The processes of the distributed system considered in this paper use loosely synchronized clocks. The paper describes a method of taking checkpoints by such processes in a truly distributed manner, that is, in the absence of a global checkpoint coordinator. The constituent processes take checkpoints according to their own clocks at predetermined checkpoint instants. A global consistent set of such asynchronous checkpoints needs to be formed to avoid the domino effect. This is achieved by adding suitable information to the existing clock synchronization messages looking at which the processes synchronize their checkpoints to form a global consistent checkpoint. Communication in this system is synchronous, so, processes may be blocked for communication at checkpointing instants. The blocked processes save the state they were in just before being blocked. It is shown here that the set of such i-th checkpoints is consistent and hence the rollback required by the system in case of failure is only up to the last saved state.

KEYWORDS

Fault tolerance, Checkpointing, Rollback recovery, Synchronized clock,Clock synchronization message

1. INTRODUCTION

The distributed system considered here consists of several processes executing on different nodes that communicate with each other via synchronous message passing. Each process has its own logical clock as explained by Lamport in [4]. It is assumed that the system uses a fault-tolerant hardware platform as described in Neogy [9]. A synchronization layer guarantees the synchronization of the individual clocks by sending suitable messages (henceforth referred to as *clock synchronization message*) at the end of each resynchronization interval as shown by Sinha [15]. There exists a constant D_{max} such that in the kth resynchronization interval ($k \geq 0$) for all correct clocks i and j, if the logical clocks of processes P_i and P_j be denoted by C_i^k and C_j^k, then as given in Srikanth [16],

$$| C_i^k(t) - C_j^k(t)| \leq D_{max} \quad (1)$$

One of the attractive approaches for providing fault tolerance to such distributed systems is the checkpoint/rollback recovery mechanism mentioned in Koo [3]. As is widely known, checkpointing is the method of periodically recording the state of a system in stable storage. Thus, a checkpoint will include the computational messages transferred by different processes. Any such periodically saved state is called a *checkpoint* of the process. As observed by Manivannan [5], a global state is a set of individual process states, one per process. The state contains a snapshot at some instant during the execution of a process. The major drawback of implementing rollback recovery technique is the *domino effect* dealt with by Tsai [19].

In a truly distributed system, where there is no central checkpoint coordinator, each process takes its own checkpoint individually. However, at any given point of time it has to be ensured that the set of most recent checkpoints taken by the processes provides a consistent picture of the system. The constituent processes of our system take individual (local) checkpoints at predetermined time instants according to their own logical clock. The set of all such local k-th checkpoints ($k \geq 0$) form the global k-th checkpoint. Since communication in the system is only through messages, it is to be guaranteed that no message gets lost in case a failure of any system component occurs. This implies that some sort of synchronization must exist among the set of otherwise asynchronous local checkpoints. The easiest, as also the much-practised means adopted in such situation is the introduction of *special message* as shown in Kalaiselvi [2]. But messages meant for checkpointing purposes only increase system overhead. To avoid this we have utilized the clock synchronization message itself for checking the consistency of the set of local checkpoints. The additional information required for this is appended at the end of the clock synchronization message. This work shows that any global checkpoint taken in the above-mentioned fashion in our system is consistent and the system has to roll back only to the last saved state in case of a failure as described in Neogy [7]. The rest of the paper is organized as follows. Section 2 gives the basic ideas about consistent checkpoints. Section 3 discusses some related works. Section 4 describes our system model, Section 5 discusses in detail the checkpointing algorithm with study of various inconsistent cases along with a proof of correctness of the algorithm. Section 6 presents the simulation results and Section 7 draws concluding remarks.

2. BASIC CONCEPTS AND IDEAS

In the present discussion we regard consistency of a checkpoint, according to Chandy [1] and Tsai [18], as the constraint that if a sender 'S' sends a message 'm' before it has taken its i-th checkpoint, then message 'm' must be received by a receiver 'R' before the receiver has taken its i-th checkpoint. A message will be termed missing in the i-th global checkpoint if its sending is recorded by S but its receipt is not recorded by R. Again, if the sending of a message is not recorded whereas its receipt is recorded then it is termed an orphan as described in Tong [17]. If a system can ensure that there is no missing or orphan message in the i-th checkpoint, then the set of all the i-th checkpoints taken by its constituent processes is bound to be consistent. Maintaining consistency is necessary to avoid the domino effect during rollback recovery in case any process fails after taking its i-th checkpoint. If the set of the i-th checkpoints can be proved to be consistent, then in case of failure the system has to roll back only up to the i-th checkpoint.

It is assumed here that a constituent process may be either blocked for communication or executing or ready for execution. Since communication is assumed to be synchronous, processes get blocked during communication. The synchronous transfer of a message m between two processes P_i and P_j involves three significant events $E_1(m)$, $E_2(m)$ and $E_3(m)$ as described below (Figure 1) where $T_i(m)$ is real time instant. Since processes in this system execute asynchronously, so, any one of the communication partners may try to execute its communication statement earlier than the other while the other may not have arrived at that communication statement yet. So, one of the partners has to wait for the other.

Event $E_1(m)$: At $T_1(m)$ the process executing its communication statement earlier begins wait for message m to be sent or received.

Event $E_2(m)$: At $T_2(m)$ the transfer of message m actually commences. At this point, the process that reaches its communication statement later has executed it.

Event $E_3(m)$: At $T_3(m)$ the transfer of m terminates.

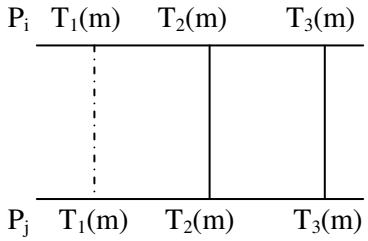

Figure 1: Significant time instants in the synchronous message passing between P_i and P_j

Blocking interval: The blocking interval $[T_x(m), T_3(m)]$ where x = 1 for the process executing its communication statement earlier and x = 2 for the process executing its communication statement later. For instance, if P_i executes its communication statement earlier, then P_i is blocked in the interval $[T_1(m), T_3(m)]$ while P_j is blocked in the interval $[T_2(m), T_3(m)]$.

It is assumed here that the state recorded for a blocked process is the state it was in just before it blocked even if its checkpointing instant occurs during its blocking period. When a particular process is 'ready' or 'running' and a checkpointing instant occurs according to its logical clock then its checkpoint is taken immediately. After taking the checkpoint it however does not resume its task but just freezes (that is, it stops executing) till the end of the current clock resynchronization interval. The idea behind the freezing is that the processes would first check their corresponding checkpoints for consistency and then proceed further. During this check a process may find out that there is a possibility of having missing/orphan message(s) in the concerned checkpointing interval that has given rise to inconsistency. Since logical clocks of the processes do not match exactly with each other there may be a skew between any two clocks whose upper bound is D_{max} as already mentioned in (1). The checkpointing instants will also suffer from that skew thus giving rise to the possibility of inconsistency. The list of computational messages exchanged in the current checkpoint interval (checkpoint interval is the time between two consecutive checkpoints) is appended to the clock synchronization message thus avoiding the overhead of sending separate messages for consistency checking. These ideas have been incorporated in designing the present checkpointing algorithm. The algorithm described in Section 5 provides corrective measure that removes any inconsistency. An alternative method of taking checkpoints by a blocked process is described in Neogy [7] where the blocked process takes checkpoint after it unblocks even if the checkpointing instant had already occurred.

3. RELATED WORKS

Unlike the approach that should exist in a distributed system Kalaiselvi [2] have a checkpoint coordinator that matches the message log it gets from all the processes at each checkpointing time. The present work does not have a checkpoint coordinator. Due to disparity in speed or congestion in the network, a message belonging to the (i+1)th checkpointing interval may reach the receiver who has not yet taken its i-th checkpoint. Such a message is not acknowledged in Tong [17] and the sender retransmits it. In Neves [10] a sender is prevented from sending for a certain time. In Neves [11] unacknowledged messages are made a part of the next checkpoint. The very design of our system prevents the occurrence of such a message. Distributed systems that use the recovery block approach of Randell [14] and have a common time base may estimate a time by which the participating processes would take acceptance tests. These estimated instants form the pseudo recovery point times as described in Ramanathan [13]. The disadvantages of such a scheme are that the fast processes have to wait for slow processes and there is a need for time-out to ensure that fault in some process does not permanently hold up other non-faulty processes. In the present work any process takes checkpoints according to its own logical clock and does not have to wait for others.

The work presented in [20] tries to minimize the number of checkpoints that are not "useful". A checkpoint is "useful" if it belongs to some consistent global checkpoint. It can be proved that a checkpoint is "useful" if and only if it is not involved in any zigzag cycle. This seems to incur a substantial overhead. The authors do not give any measure of this overhead. On the contrary, our algorithm does not incur this overhead as we do not need to detect zigzag cycles.
Furthermore, Xu and Netzer's work does not attempt to eliminate rollback propagation. It does not guarantee that every checkpoint will belong to some consistent checkpoint. Instead, it reduces rollback propagation to less than one checkpoint interval. In contrast, our algorithm always ensures that every checkpoint is globally consistent. We assume a synchronous communication model and there are no "in-transit" messages. Rollback propagation is irrelevant in our algorithm and rollback is always to the last (globally consistent) checkpoint.
The work presented in [21] defines a function $\rightsquigarrow$. Two sets of checkpoints R and S are related R $\rightsquigarrow$ S iff there is a zigzag path from some member of R to some member of S. The authors show how a consistent set of checkpoints can be obtained by starting from a set S satisfying S $\not\rightsquigarrow$ S.
In our algorithm, this is not an issue since every global checkpoint is consistent and only the last one needs to be saved.
The work presented in [22] is a communication-induced checkpointing protocol that ensures the RDT (Rollback Dependency Trackability) property. Each process maintains a Transitive Dependency Vector TDV of integers. It also maintains a Boolean array simple and a Boolean matrix causal. When a message m is sent, the variable tdv, causal and simple are piggybacked on the message. On receiving 'm', the destination process updates its own variables and takes a forced checkpoint if required. The "per-message overhead" of this algorithm can be significant for fine-grain applications, i.e., applications which communicate frequently. In contrast, our algorithm does not involve any "per-message overhead". The overhead that we incur is only on a per-checkpoint basis. That overhead is (i) the bookkeeping of the number of messages exchanged between different processes during the last checkpointing interval. This information is piggybacked on clock synchronization messages that are periodically exchanged among processes; and (ii) the freezing time, i.e., the time for which a process remains suspended after taking a checkpoint.
The work in [23] presents a more formal and generalized framework in which the works of [22] exists. The author defines Elimentary-Prime-Simple-Causal-Message Z-paths (EPSCM). It is then established that EPSCM property is RDT-compliant and that this property can not be implied by any other RDT-compliant property with respect to the set of Z-paths. A protocol based on minimal EPSCM-path subset has also been presented. This again suffers from the per-message overhead already discussed.

4. SYSTEM MODEL

Let us now consider a system of 'n' processes, P_0, P_1, …. P_{n-1} each of which takes individual checkpoints at predetermined instants of its logical clock. Let these instants be denoted by C_k^i, where i denotes the checkpoint index ($i \geq 0$) and k denotes the process id ($0 \leq k \leq n-1$). Let the checkpoints be denoted as the initial checkpoint CP_k^0, first checkpoint CP_k^1, second checkpoint CP_k^2 and so on. The initial checkpoint will be taken when the system is initialized. Let us now state our assumptions regarding the system:
Logical clocks of any two processes are synchronized periodically by exchanging messages [15, 16] (referred to as clock synchronization messages) and are at most D_{max} apart from each other [Section1]
Communication is synchronous and performs the steps mentioned in Section 2 (figure 1) where $T_k(m)$ denotes instant of time for a message m throughout this paper:
Each process maintains information about messages sent and received during each checkpointing interval by storing the (sender id/ receiver id, message id) pair.

Checking of consistency between a pair of kth checkpoints CP_i^k and CP_j^k of processes P_i and P_j respectively involve checking of whether message/s recorded sent by one (P_i or P_j) in its corresponding checkpoint is/are recorded received by the other (P_j or P_i) in its corresponding checkpoint and vice versa.
Processes blocked in communication are able to take checkpoint if checkpointing instant t occurs during that interval that is, $T_x(m) < t < T_z(m)$. The state recorded is the one the process was in prior to $T_x(m)$.
Assumption 5 above tells that if a process is in the midst of receiving (at time instant t where $T_x(m) < t < T_3(m)$) while its checkpointing instant occurs, then the checkpoint will not include the receiving buffer. Also, if a process is in the midst of sending (at time instant t where $T_x(m) < t < T_3(m)$) while its checkpointing instant occurs, then the checkpoint will not include the sending buffer. Communication is taken care of by the existing protocols for synchronous communication.

5. THE CHECKPOINTING ALGORITHM

Before formally presenting the algorithm we describe its working by explaining various situations under which a process P_i may have to take its checkpoint.
We have used the Clock Synchronization Algorithm of Srikanth [16]. Let the clock resynchronization period be P. Resynchronization is done periodically and the resynchronization interval is designated K (=1,2,,….). Let 'f' be the maximum number of faulty nodes in the system. When the logical clock of a node reaches K.., it broadcasts a "Ready-K" message to all other nodes, signifying that it is ready to resynchronize. This message is "signed" to provide Byzantine Resilience. On receiving such a messagem every node relays it to the other nodes. When a node receives (f+1) "Ready-K" messages originated from distinct nodes, it resynchronizes its logical clock to KP+α where α is a constant. It then increments K, the resynchronization interval number. This algorithm ensures that the logical clocks of all non-faulty nodes are synchronized to within a predefined constant D_{max}. The obvious constraints on D_{max} are the maximum clock drift ρ and the maximum node-to-node message transit time (single-hop) t_{del}. When any process resynchronizes its clock, every other process is guaranteed to do so within a time bound D_{max} [Axiom 7]. For a special D_{max}, the constant α is given by Axiom 2. The upper and lower bounds on P are specified by Axiom 3 and 4 respectively. The formal specification of the physical clock drift ρ is given by Axiom 6. The axioms are described in Section 5.3.
From Srikanth [16] we know that a process expects its kth resynchronization at time kP on its logical clock where P is the logical time between resynchronizations. Resynchronization takes place by setting the clock of the process to kP+α where α is a constant. In the present work we have chosen the predetermined checkpointing instants at kP+0.5P+α for specific values of k.

5.1 WORKING OF THE ALGORITHM

Here we describe in details how a process P_i would take its kth checkpoint according to the present algorithm. Each process maintains a list of computational messages in the form of mlist(P_i, k) which is essentially a list of message transfers in which P_i participated during the k-th checkpointing interval. So, mlist(P_i, k) = msent(P_i, k) ∧ mrecd(P_i, k) where msent(P_i, k) and mrecd(P_i, k) denote respectively the list of messages sent and received by P_i during the k-th checkpointing interval. Let us denote by sender(m) the process that sent message m and by receiver(m) the process that receives message m. Let the logical clock of P_i determine the checkpoint instant for taking k-th checkpoint. The take_ckpt() algorithm is invoked periodically by the system when the logical clock reaches predefined values [Rule 1, Section 5.3]. It is a time-

triggered system process independent of the application process. At that time P_i may be in one of the following states:

Case 1: Executing computation: P_i is interrupted and immediately saves its current state. This represents step 2.1 of take_ckpt(). Rather than resuming its computation P_i freezes, that is, stops its activities and waits for the time-out of the current resynchronization interval (step 2.2). At the end of the interval P_i sends out messages for resynchronizing logical clock to other processes in the system as elaborated in Sinha [15] and Srikanth [16]. P_i appends information of computation messages that it has sent and received (according to assumption 3 in Section 4) in the current checkpointing interval to the message of clock synchronization. P_i now resumes whatever it was doing before the kth checkpointing instant occurred.

Case 2: In the midst of communication: Since communication is assumed to be synchronous (Section 2) P_i is either blocked or transferring data. The checkpoint state-saving can proceed concurrently without interrupting the communication. In this case, the state saved is the one just prior to the beginning of execution of communication statement. However, when P_i finishes communication it has to be suspended, i.e., it cannot reschedule its next activity until the following clock synchronization instant.

If P_i remains blocked in communication when the following clock synchronization occurs, the system still works consistently. The information that needs to be exchanged on occurrence of event1 (time-out of current clock synchronization interval) is exchanged in the background by the system without any involvement on the part of P_i. The message transfer in which P_i participates is NOT recorded either at the sender-end or at the receiver-end since the state saved is the one at the beginning of the communication. Since communication is synchronous, P_i's partner cannot record the (termination of) message transfer until P_i does so.

P_i checks each such information for possible existence of any one of the following:

Case 2a: A message has been recorded to be received by some P_j (in CP_j^k) whose sender is P_i though P_i's local checkpoint (CP_i^k) does not record the send of the same message, that is,

$\exists m$: mrecd(P_j, k) and $m \notin$ ((msent(P_i, k)) $\wedge$ (sender(m) = P_i))

In this case P_i's checkpoint is modified to record the corresponding send. Then P_i resumes its next task.

Case 2b: A message has been recorded to be sent by P_j (in CP_j^k) whose destination is P_i though P_i's local checkpoint (CP_i^k) does not record the receipt of the same message, that is,

$\exists m$: msent(P_j, k) and $m \notin$ ((mrecd(P_i, k)) $\wedge$ (sender(m) = P_j))

In this case P_i takes another checkpoint called a forced checkpoint to include the corresponding receipt. Then P_i resumes its task.

Before going into the algorithm let us describe the procedures, events and date structures used in the algorithm for checkpointing in a system of n processes. The kth checkpoint in process P_i is denoted by CP_i^k.

Events:

event1: Time-out of current clock synchronization interval

event2: Receiving by P_i of clock resynchronization messages along with other information (sp_mess) (described later) from P_j where $0 \le j \le n-1$ and $j \ne i$

Procedures:

save_state(): procedure that writes current process state in stable storage, that is it writes checkpoint CP_i^k

save_block_state(): procedure that writes in stable storage the state of a blocked process prior to blocking, that is it writes checkpoint CP_i^k.

save_mod_state(): procedure that accesses CP_i^k from stable storage and appends updated mess_sent_own[] array (defined below)

wait(event): procedure that suspends current activities until event occurs

send(source, destination, message): procedure for sending message in synchronous mode

receive(source, message): procedure for receiving messages in synchronous mode

Data structures:
mess_sent_own[]: an n-element vector maintained by each process P_i that indicates the number of messages P_i has sent to the process with vector index as process id during the current checkpointing interval, that is, mess_sent_own[j] = x means that P_i has sent x messages in the current checkpointing interval to P_j.
mess_recd_own[]: an n-element vector maintained by each process P_i that indicates the number of messages P_i has received from the process with vector index as process id during the current checkpointing interval, that is, mess_recd_own[j] = x means that P_i has received x messages in the current checkpointing interval from P_j.
clock_resynch_mess: message for clock resynchronization as described in Sinha [15] and Srikanth [16]
sp_message: special message carrying mess_sent_own[] and mess_recd_own[] with clock_resynch_mess
mess_sent_other[i][]: This array is compiled by P_i after P_i receives mess_sent_own[] of each P_j from all P_j ($0 \le j \le n-1$ and $j \ne i$) in the following way:
for(all j = 0,1,2,..,n-1 and j≠i)
    mess_sent_other[i][j] = mess_sent_own[j]
mess_recd_other[i][]: This array is compiled by P_i after P_i receives mess_recd_own[] of each P_j from all P_j ($0 \le j \le n-1$ and $j \ne i$) in the following way:
for(all j = 0,1,2,..,n-1 and j≠i)
    mess_recd_other[i][j] = mess_recd_own[j]
flag_new, flag_mod: boolean variables which when set indicate that a new checkpoint has to be taken or the current checkpoint has to be modified respectively
All array elements are initialized to –1 and all boolean variables to false at the beginning of each checkpointing interval.
Symbols:
: concatenation
Algorithm: take_ckpt()
(For any process P_i whose kth checkpointing instant C_i^k has occurred)
flag_new, flag_mod = false
If P_i is executing computation, then
P_i calls save_state()
P_i calls wait(event1)
For (all j = 0,1,2,…,n-1 and j ≠ i)
sp_message = clock_resynch_mess : mess_sent_own[] : mess_recd_own[]
send(P_i, P_j, sp_message)
3. else // P_i is blocked in communication //
P_i calls save_block_state() from its blocked state
P_i calls wait(event1)
For (all j = 0,1,2,…,n-1 and j ≠ i)
sp_message = clock_resynch_mess : mess_sent_own[] : mess_recd_own[]
send(P_i, P_j, sp_message)
P_i calls wait(event2)
For (all j = 0,1,2,..,n-1 and j≠i)
if (mess_sent_own[j] < mess_recd_other[i][j])
  mess_sent_own[j] := mess_recd_other[i][j]
  flag_mod = true
if (mess_recd_own[j] < mess_sent_other[i][j])
  flag_new = true
  break

If (flag_new) then
P_i calls save_state()
else if (flag_mod) then
P_i calls save_mod_state()

5.2 Analysis of the Algorithm

In this section we analyze various scenarios that might occur in such a distributed system. We have used certain notations in the figures that are indicated below. In the following discussion event means communication (send and receive) between two processes.

: event that would have happened : event that actually happens

: exchange of clock synchronization messages : process blocked at C^k

Case 1: P_i and P_j are both *not blocked* during their respective checkpointing instants.

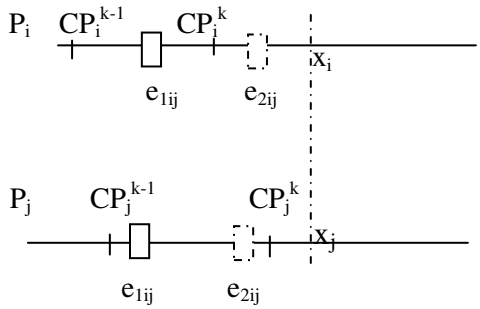

Figure 2

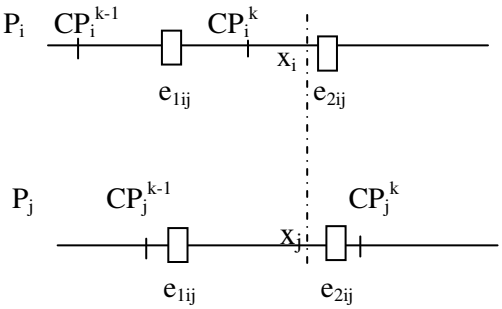

Figure 3

In both the above figures (figure 2, figure 3) P_i is assumed to be faster than P_j and hence takes its CP_i^k earlier than P_j.

Case i: Inconsistency can occur if event e_{2ij} is not recorded in the checkpoint CP_i^k of P_i but recorded in the checkpoint CP_j^k of P_j in figure2. However, by virtue of step 2.2 of the algorithm, P_i remains suspended in the interval (CP_i^k, x_i) and hence can not participate in event e_{2ij}. Since e_{2ij} is a synchronous event, it can not be executed in P_j either. Hence no inconsistency can occur. Thus CP_i^k and CP_j^k are mutually consistent.

Case ii: Let us consider figure3. Checkpointing instant C_j^k of P_j occurs after x_j. This might lead to inconsistency due to events between (x_j, CP_j^k) as shown by event e_{2ij} in the figure. Theorem 1 in Section 5.3 shows that checkpointing instant of the process with the slower clock will occur earlier than the next clock synchronization instant of the process with the faster clock. Hence the case described in figure 3 would never happen in reality.

Case 2: P_i is blocked but P_j is not blocked during their respective checkpointing instants

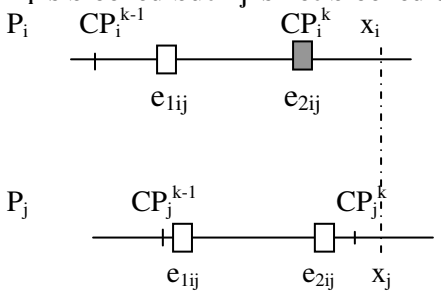

Figure 4

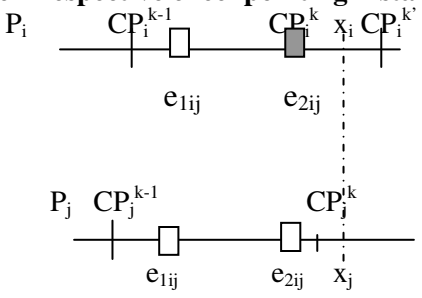

Figure 5

Now we consider processes that are blocked for communication.

Case i: Let P_i be faster than P_j as depicted in figure 4. The kth checkpointing instant C_i^k occurs when P_i is blocked on event e_{2ij}. This is shown as shaded box in figure 4. Let P_i be the sender in e_{2ij}. P_i does not record the send of e_{2ij} in CP_i^k since the event has not yet finished. Inconsistency will occur if P_j records the receipt of e_{2ij} in CP_j^k. Since P_j has finished receiving when its checkpointing instant C_j^k occurs, so it records e_{2ij} in CP_j^k leading to inconsistency. It is evident from the figure that P_j has advanced during the interval (time of occurrence of e_{2ij}, CP_j^k). At x_i P_i notes the inconsistency (step 3.5.1 of the algorithm). Since P_i has not executed anything during the interval (CP_i^k, x_i) (step 3.2 of algorithm) so, the algorithm (steps 3.7, 3.7.1) modifies the CP_i^k only to the effect that it now records the send of e_{2ij} and thus CP_i^k and CP_j^k become mutually consistent.
Case ii: Let P_i be faster than P_j as depicted in figure 5. The k-th checkpointing instant C_i^k occurs when P_i is blocked on event e_{2ij}. This is shown as shaded box in figure 5. Let P_i be the receiver in e_{2ij}. P_i does not record the receipt of e_{2ij} in CP_i^k since the event has not yet finished. But, checkpointing instant C_j^k occurs in P_j after P_j has finished "sending". So, P_j records event e_{2ij} in CP_j^k. This leads to inconsistency. P_i notes the inconsistency at x_i (step 3.5.2 of the algorithm) and takes a forced checkpoint $CP_i^{k'}$ (steps 3.6, 3.6.1. of the algorithm) to become mutually consistent with P_i.

Case 3: P_i and P_j are both *blocked* during their respective checkpointing instants

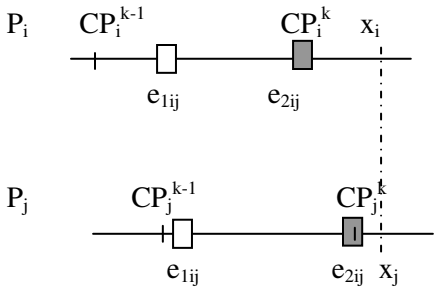

Figure 6

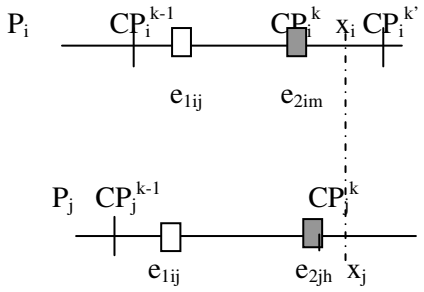

Figure 7

Case 3a: Let P_i and P_j be blocked over the same communication event e_{2ij}. Hence according to the algorithm none of them would record the communication event in their respective checkpoints CP_i^k and CP_j^k. Hence no inconsistency would arise.
Case 3b: Let P_i and P_j be blocked over different communication events e_{2im} and e_{2jh}. Hence according to the algorithm none of them would record the corresponding communication events in their respective checkpoints CP_i^k and CP_j^k. If the communication partners P_m (of P_i) and P_h (of P_j) record the corresponding events in their respective checkpoints CP_m^k and CP_h^k, then inconsistency would be dealt with as already described in Case2. Otherwise no inconsistency would arise.

5.3. PROOF OF CORRECTNESS

Axiom 1: The clock drift rate ρ satisfies $0<\rho<1$ by Srikanth [16].
Axiom2: For a user-selected maximum clock deviation D_{max}, and maximum message transit time t_{del}, the clock correction α has a lower bound given by Srikanth [16]

$$\alpha \geq [(1+\rho)D_{max} + t_{del}] * (1+\rho)$$

Axiom 3: The lower bound on the clock resynchronization interval P is given by Srikanth [16]

$$P \geq d_{min}(1+\rho) + \alpha$$

Axiom 4: The upper bound on P is (Srikanth [16])

$$P \leq [((D_{max} - d_{min}(1+\rho)) / dr) - t_{del}] / (1+\rho)$$

Axiom 5: The maximum drift rate dr between clocks is (Srikanth [16])

$$dr = (1+\rho) - (1+\rho)^{-1}$$

Axiom 6: A straight line envelope of the logical clock of a process has a slope m satisfying
$(1+\rho)^{-1} \leq m \leq (1+\rho)$
Axiom 7: The maximum time difference between resynchronization of two processes is
$d_{min} = t_{del} > 0$ (Srikanth[15])
Rule 1: The logical clock time of checkpointing for a process x is chosen by the following rule
$C_x^{k+1}(T_x^k) = (k+0.5)P + \alpha$
where T_x^k is the corresponding real time.
This rule is obviously a design issue. It sets the checkpointing instant halfway (on the logical clock axis) during the (k+1)th resynchronization period (k = 1,2,……). However, a process takes checkpoint for some specified values of k.
Lemma 1: (i) $dr < 2\rho$ and (ii) $(1+\rho)^{-1} > (1-\rho)$
Lemma 2: If $z > 0$ then (a) $x>y \leftrightarrow xz>yz$ and (b) $x>y$ $x/z>y/z$
Lemma 3: If $x > y > 0$, then $1/x<1/y$
Lemma 4: $dr > 0$
Lemma 5: The product of positive monotonic increasing functions is a positive monotonic increasing function.
Lemma 6: The function $f_n(\rho) = \rho^n$ where n is a positive integer is monotone increasing.
Lemma 7: The sum of two monotone increasing functions is monotone increasing.
Lemma 8: The function $e(\rho) = k\rho^n$ where k is a positive constant and n is a positive integer is monotone increasing.
Lemma 9: If $x \geq x_{min}$ and $y \leq y_{max}$ then $x_{min} - y_{max} > 0 \Rightarrow (x - y > 0)$
Lemma 10: If $0<x\leq x_{max}$ and $y\geq y_{min}>0$ then $x/y \leq x_{max}/y_{min}$
Lemma 11: For any two processes P_s and P_r, the respective kth checkpointing instants T_s^k and T_r^k during the kth resynchronization interval are selected by RULE-1. Without loss of generality, we assume that P_r has the faster clock. Then $T_r^c - T_s^k = (P-\alpha)(1/m_1 - 0.5/m_2) - (t_2 - t_1)$ where T_r^c is the next clock resynchronization time of P_r.
Proof: Without loss of generality, we assume that P_r has the faster clock, that is,
$m_1>m_2$ and $t_1<t_2$
Figure 8 shows the logical clocks C_r^{k+1} and C_s^{k+1} of P_r and P_s during the (k+1)th resynchronization interval. They have slopes m_1 and m_2. By Rule-1,

$$C_r^{k+1}(T_r^k) = C_s^{k+1}(T_s^k) = (k+0.5)P+0.5\alpha \quad (11.1)$$

Since $(T_r^c, C_r^{k+1}(T_r^c))$ and $(t_1, C_r^{k+1}(t_1))$ are two points on straight line C_r^{k+1} having slope m_1,

$$[C_r^{k+1}(T_r^c) - C_r^{k+1}(t_1)] / (T_r^c - t_1) = m_1 \quad (11.2)$$

By Srikanth and Toueg's algorithm [15],

$$C_r^{k+1}(T_r^c) = (k+1)P \quad (11.3a)$$
$$C_s^{k+1}(T_s^c) = (k+1)P \quad (11.3b)$$

and
$$C_r^{k+1}(t_1) = kP+\alpha \quad (11.4a)$$
$$C_s^{k+1}(t_2) = kP+\alpha \quad (11.4b)$$

Substituting (11.3a) and (11.4a) in (11.2) we get,

$$T_r^c - t_1 = (P-\alpha)/m_1 \quad (11.5)$$

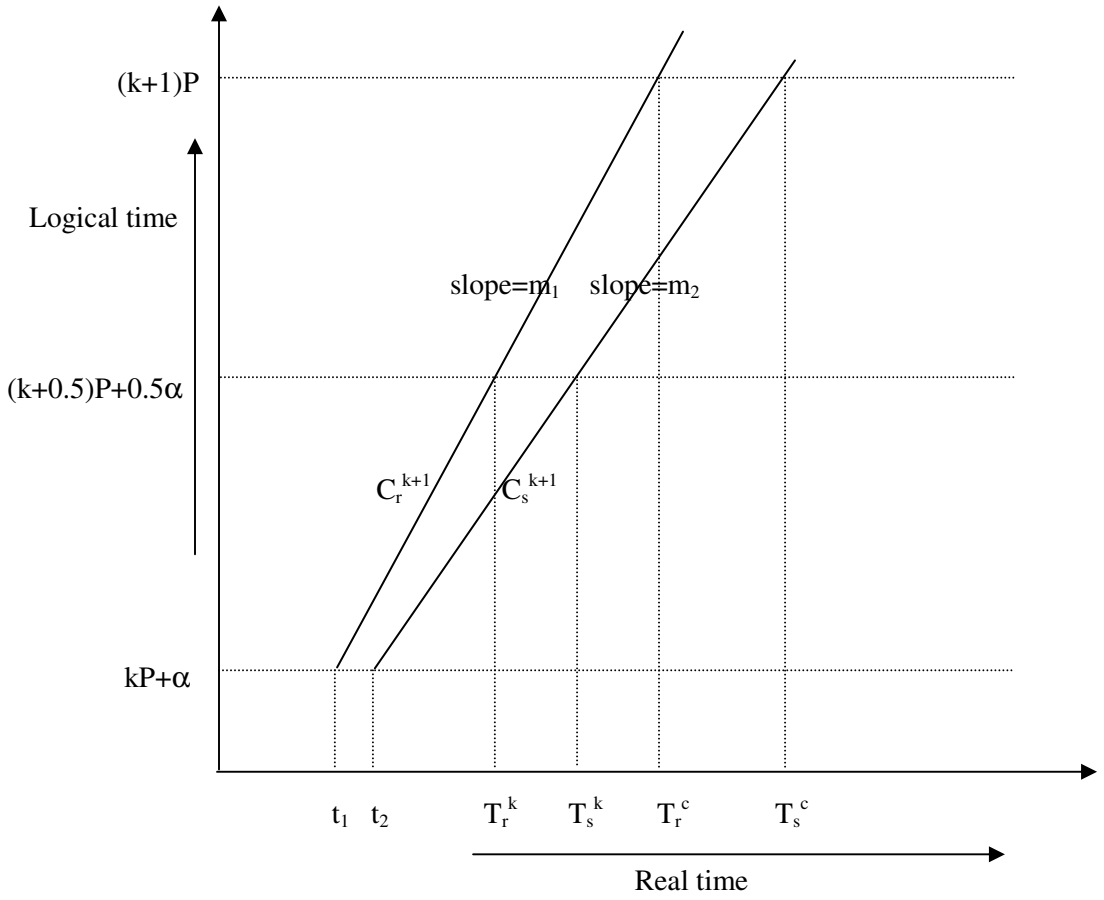

Figure 8. Checkpointing and resynchronization instants of P_r and P_s during the (k+1)th resynchronization interval

Similarly, since $(T_s^k, C_s^{k+1}(T_s^k))$ and $(t_2, C_s^{k+1}(t_2))$ are two points on straight line C_s^{k+1} having slope m_2,

$$[C_s^{k+1}(T_s^k) - C_s^{k+1}(t_2)] / (T_s^k - t_2) = m_2 \quad (11.6)$$

Substituting (11.1) and (11.4b) in (11.6)

$$T_s^k - t_2 = 0.5(P-\alpha)/m_2 \quad (11.7)$$

From (11.5) and (11.7), we get

$$(T_r^c - t_1) - (T_s^k - t_2) = (P-\alpha)(1/m_1 - 0.5/m_2) \quad (11.8)$$

$$\Rightarrow (T_r^c - T_s^k)+(t_2 - t_1) = (P-\alpha)(1/m_1 - 0.5/m_2)$$

$$\Rightarrow (T_r^c - T_s^k) = (P-\alpha)(1/m_1 - 0.5/m_2) - (t_2 - t_1) \quad (11.9)$$

Lemma 12: If (i) $(1/m_1 - 0.5/m_2 > (t_2 - t_1)/(P - \alpha))$ and

(ii) $t_1 < t_2$

then $1/m_1 - 0.5/m_2 > 0$

Proof:

1. $d_{min} > 0$	[by Axiom 7]
2. $\rho > 0$	[by Axiom 1]
3. $1+\rho > 0$	[by 2]
4. $d_{min}(1+\rho) > 0$	[by (1), (3) and Lemma 2]
5. $(P - \alpha) > d_{min}(1+\rho)$	[by Axiom 3]
6. $(P - \alpha) > 0$	[by (4) and (5)]
7. $t_2 - t_1 > 0$	[by Hypothesis (ii)]
8. $(t_2 - t_1)/(P - \alpha) > 0$	[by (6), (7) and Lemma 2]
9. $1/m_1 - 0.5/m_2 > (t_2 - t_1)/(P - \alpha)$	[by Hypothesis (i)]

10. $1/m_1 - 0.5/m_2 > 0$ [by (8) and (9)]
End Proof
Lemma 13: If $\rho < (\sqrt{2} - 1)$ then $(1+\rho)^{-1} - 0.5(1+\rho) > 0$
Proof:
1. $\rho < (\sqrt{2} - 1)$ [by Hypothesis]
2. $\rho > 0$ [by Axiom 1]
3. $\rho > -(\sqrt{2} - 1)$ [by (2)]
4. $\rho + (\sqrt{2} - 1) > 0$ [by (3)]
5. $(\rho - \sqrt{2} + 1) < 0$ [by (1)]
6. $(\rho + \sqrt{2} + 1)(\rho - \sqrt{2} + 1) < 0$ [by (4), (5) and Lemma 2]
7. $(\rho + 1)^2 - (\sqrt{2})^2 < 0$ [by (6)]
8. $2 - (\rho + 1)^2 > 0$ by (7)]
9. $2(1 + \rho)^{-1} - (1 + \rho) > 0$ [by (8), Lemma 2 and Axiom 1]
10. $(1 + \rho)^{-1} - 0.5(1 + \rho) > 0$ [by (9) and Lemma 2 since $0.5 > 0$]
End Proof
Lemma 14: If $\rho < (\sqrt{2} - 1)$ then $(1/m_1 - 0.5/m_2) > 0$
Proof:
1. $\rho < (\sqrt{2} - 1)$ [by Hypothesis]
2. $(1 + \rho)^{-1} - 0.5(1 + \rho) > 0$ [by (1) and Lemma 13]
3. $1/m_1 \geq (1+\rho)^{-1}$ [by Axiom 6]
4. $1/m_2 \leq (1+\rho)$ [by Axiom 6]
5. $0.5/m_2 \leq 0.5(1+\rho)$ [by (4) and Lemma 2 since $0.5 > 0$]
6. $(1 + \rho)^{-1} - 0.5(1 + \rho) > 0 \Rightarrow (1/m_1 - 0.5/m_2 > 0)$ [by (3), (5) and Lemma 9]
7. $(1/m_1 - 0.5/m_2 > 0)$ [by (2) and (6)]
End Proof
Lemma 15: If (i) $\rho < (\sqrt{2} - 1)$ and
(ii) $P - \alpha > (t_2 - t_1) / (1/m_1 - 0.5/m_2)$
then $T_r^c - T_s^k > 0$
Proof:
1. $1/m_1 - 0.5/m_2 > 0$ [by Hypothesis (i) and Lemma 14]
2. $T_r^c - T_s^k = (P - \alpha)(1/m_1 - 0.5/m_2) - (t_2 - t_1)$ [by Lemma 11]
3. $P - \alpha > (t_2 - t_1) / (1/m_1 - 0.5/m_2)$ [by Hypothesis (ii)]
4. $(P - \alpha)(1/m_1 - 0.5/m_2) > (t_2 - t_1)$ [by (3), (1) and Lemma 2]
5. $T_r^c - T_s^k > 0$ [by (4) and (2)]
End Proof
Lemma 16: If (i) $\rho < (\sqrt{2} - 1)$ and (ii) $t_2 > t_1$
then $(t_2 - t_1) / (1/m_1 - 0.5/m_2) \leq d_{min} / (1 + \rho)^{-1} - 0.5(1 + \rho)$
Proof:
1. $t_2 - t_1 > 0$ [by Hypothesis (ii)]
2. $0 < t_2 - t_1 \leq d_{min}$ [by (1) and Axiom 7]
3. $(1 + \rho)^{-1} - 0.5(1 + \rho) > 0$ [by Hypothesis (i) and Lemma 13]
4. $1/m_1 \geq (1+\rho)^{-1}$ [by Axiom 6]
5. $1/m_2 \leq (1+\rho)$ [by Axiom 6]
6. $0.5/m_2 \leq 0.5(1+\rho)$ [by (5) and Lemma 2 since $0.5 > 0$]
7. $(1 + \rho)^{-1} - 0.5(1 + \rho) \leq (1/m_1 - 0.5/m_2)$ [by (4), (6) and Lemma 9(b)]
8. $(1/m_1 - 0.5/m_2) \geq (1 + \rho)^{-1} - 0.5(1 + \rho) > 0$ [by (7) and (3)]
9. $(t_2 - t_1) / (1/m_1 - 0.5/m_2) \leq d_{min} / (1 + \rho)^{-1} - 0.5(1 + \rho)$ [by (2), (8) and Lemma 10]
End Proof
Lemma 17: $d_{min} / (1 + \rho)^{-1} - 0.5(1 + \rho) = 2d_{min} / (1 + \rho)^{-1} - dr$

Proof: $d_{min} / (1 + \rho)^{-1} – 0.5(1 + \rho)$
$= 2d_{min} / 2(1 + \rho)^{-1} – (1 + \rho)$
$= 2d_{min} / ((1 + \rho)^{-1} + (1 + \rho)^{-1} – (1 + \rho))$
$= 2d_{min} / ((1 + \rho)^{-1} – ((1 + \rho) - (1 + \rho)^{-1}))$
$= 2d_{min} / ((1 + \rho)^{-1} – dr)$ [by Axiom 5]
End Proof

Lemma 18: If (i) $(P-\alpha) > 2d_{min} / ((1 + \rho)^{-1} – dr)$
(ii) $\rho < (\sqrt{2} - 1)$ and
$t_2 > t_1$
then $T_r^c > T_s^k$
Proof:
1. $(P-\alpha) > 2d_{min} / ((1 + \rho)^{-1} – dr)$ [by Hypothesis]
2. $2d_{min} / ((1 + \rho)^{-1} – dr) = d_{min} / ((1 + \rho)^{-1} – 0.5 (1 + \rho))$ [by Lemma 17]
3. $(P-\alpha) > d_{min} / ((1 + \rho)^{-1} – 0.5 (1 + \rho))$ [by (1) and (2)]
4. $(t_2 – t_1) / (1/m_1 – 0.5/m_2) \leq d_{min} / (1 + \rho)^{-1} – 0.5(1 + \rho)$ [by Hypothesis (ii), (iii) and Lemma 16]
5. $(P-\alpha) > (t_2 – t_1) / (1/m_1 – 0.5/m_2)$ [by (3) and (4)]
6. $T_r^c - T_s^k > 0$ [by (5), Hypothesis (ii) and Lemma 15]
End Proof
Lemma 19: If $\rho < (\sqrt{2} - 1)$ then $2d_{min} / ((1 + \rho)^{-1} – dr) > d_{min}(1 + \rho)$
Proof:
$\rho > 0$ [by Axiom 1]
$1+\rho > 1$ [by (1)]
$[(1+\rho) -1] > 0$ [by (2)]
$[(1+\rho) +1] > 1 + 1$ [by (2)]
$[(1+\rho) +1] > 0$ [by (4) and since $2 > 0$]
6. $[(1+\rho)-1][(1+\rho)+1] > 0$ [by (3),s (5) and Lemma 2]
7. $(1+\rho)^2 -1 > 0$ [by (6)]
8. $(1+\rho)^2 - 2 + 1> 0$ [by (7)]
9. $2 - (1+\rho)^2 < 1$ [by (8)]
10. $\rho < (\sqrt{2} - 1)$ [by Hypothesis]
11. $1 + \rho < \sqrt{2}$ [by (10)]
12. $1 + \rho - \sqrt{2} < 0$ [by (11)]
13. $1 + \rho + \sqrt{2} > 0$ [by Axiom 1]
14. $[(1+\rho) - \sqrt{2}][(1+\rho) + \sqrt{2}]< 0$ [by (12), (13) and Lemma 2]
15. $(1+\rho)^2 - 2 < 0$ [by (14)]
16. $2 - (1+\rho)^2 > 0$ [by (15)]
17. $0 < 2 - (1+\rho)^2 < 1$ [by (9) and (16)]
18. $1 < 1 / [2 - (1+\rho)^2]$ [by (17) and Lemma 3]
19. $2 < 2 / [2 - (1+\rho)^2]$ [by (18) and Lemma 2 since $2 > 0$]
20. $2 - 1 < 2 / [2 - (1+\rho)^2] - 1$ [by (19)]
21. $2 / [2 - (1+\rho)^2] – 1 > 0$ [by (20) since $1 > 0$]
22. $2d_{min} / ((1 + \rho)^{-1} – dr) = 2d_{min}(1 + \rho) / (1 – [(1 + \rho)^2 – 1])$ [by Axiom 5]
23. $2d_{min} / ((1 + \rho)^{-1} – dr) - d_{min}(1 + \rho) = d_{min}(1 + \rho)[2 / (1 – [(1 + \rho)^2 – 1]) – 1]$
$= d_{min}(1 + \rho)[2 / [(2 – (1 + \rho)^2)] – 1]$ [by (22)]
24. $d_{min}(1 + \rho) > 0$ [since $d_{min} > 0$ by Axiom7 and $\rho > 0$ by Axiom 1]
25. $d_{min}(1 + \rho)[2 / [(2 – (1 + \rho)^2)] – 1] > 0$ [by (24), (21) and Lemma 2]
26. $2d_{min} / ((1 + \rho)^{-1} – dr) - d_{min}(1 + \rho) > 0$ [by (23) and (25)]

27. $2d_{min} / ((1 + \rho)^{-1} - dr) > d_{min}(1 + \rho)$ [by (26)]

End Proof

Definition 1. $k_3 \equiv d_{min}(1+\rho)$

Definition 2. $k_{37} \equiv 2d_{min} / [(1+\rho)^{-1} - dr]$

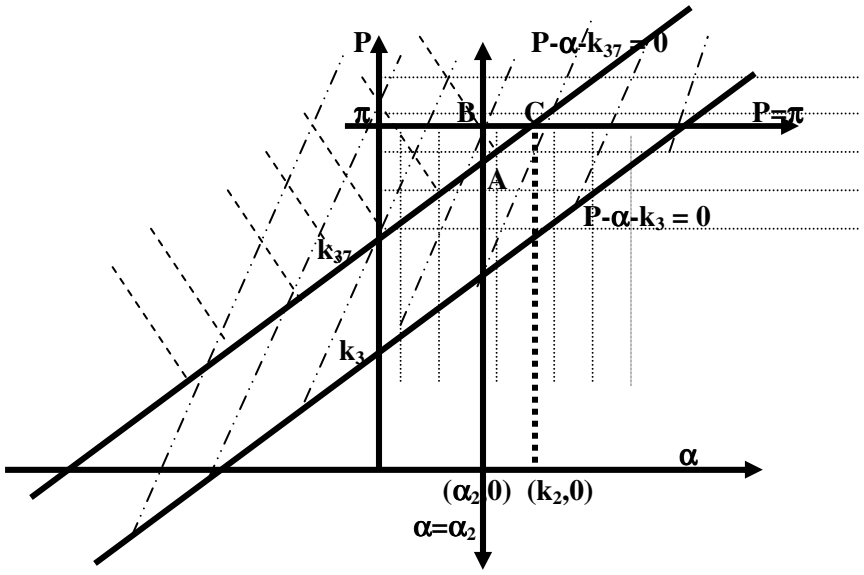

Figure 9: Finding the feasible values of P and α

In figure 9, we plot the design parameters P and α. For convenience, we define,

Definition 3. $\pi \equiv [[[D_{max}-d_{min}(1+\rho)]/dr] - t_{del}] / (1+\rho)$

Definition 4. $\alpha_2 \equiv [(1+\rho)D_{max}+t_{del}] (1+\rho)$

Definition 5. $k_2 \equiv \pi - k_{37}$

Definition 6. $Z \equiv d_{min}/dr + t_{del}/(1+\rho) + t_{del}(1+\rho)$

Definition 7. $W \equiv 2d_{min} / [dr(1+\rho)] + d_{min}/dr + t_{del}/(1+\rho) + 2\, t_{del}(1+\rho)$

Lemma 20: If $D_{max}[1 / dr(1+\rho) - (1+\rho)^2] > 2d_{min} / ((1 + \rho)^{-1} - dr) + Z$
Then $k_2 > \alpha_2$ and conversely.

Proof:

Let $k_2 > \alpha_2$

$\Leftrightarrow \pi - k_{37} > \alpha_2$

$\Leftrightarrow \pi - k_{37} > [(1+\rho)D_{max}+t_{del}] (1+\rho)$ [by Definition 4]

$\Leftrightarrow \pi > 2d_{min} / [(1+\rho)^{-1} - dr] + [(1+\rho)D_{max}+t_{del}] (1+\rho)$ [by Definition 2]

$\Leftrightarrow [[[D_{max}-d_{min}(1+\rho)]/dr] - t_{del}] / (1+\rho) > 2d_{min} / [(1+\rho)^{-1} - dr] + [(1+\rho)D_{max}+t_{del}] (1+\rho)$ [by Definition 2]

$\Leftrightarrow D_{max}[[1/(1+\rho)dr] - (1+\rho)^2] > d_{min}/dr + t_{del}/(1+\rho)+2d_{min}/[(1+\rho)^{-1} - dr] + t_{del}(1+\rho)$

$\Leftrightarrow D_{max}[[1/(1+\rho)dr] - (1+\rho)^2] > 2d_{min}/[(1+\rho)^{-1} - dr] + Z$ [by Definition 6]

End Proof

Lemma 21: If $\rho < 0.2054687$
then $dr(2 + \rho) - (1+\rho)^{-1} < 0$

Proof: Let $dr(2 + \rho) - (1+\rho)^{-1} < 0$ (L21.1)

$\Leftrightarrow dr(2 + \rho)(1+\rho) - 1 < 0$ [by Lemma 2(i) since $1+\rho>0$ by Axiom 1]

$\Leftrightarrow (2 + \rho)[(1+\rho)^2 - 1] - 1 < 0$ [by Axiom 5]

$\Leftrightarrow (2 + \rho)[(1+\rho)+1][(1+\rho)-1] - 1 < 0$ [by Axiom 5]

$\Leftrightarrow (2 + \rho)(2+\rho)\rho - 1 < 0$
$\Leftrightarrow \rho^3 + 4\rho^2 + 4\rho < 1$ (L21.2)
Let $f(\rho) = \rho^3 + 4\rho^2 + 4\rho$ …… (L21.3)
By iterative techniques, we find that
$f(0.205664) = 1.000546 > 1$ (L21.4)
and $f(0.2054687) = 0.9994189 < 1$
By Lemma 8 each of the terms on the right hand side of [L21.3] is monotone increasing. It follows, therefore, from [L21.4] that
$\rho < 0.2054687 \Rightarrow f(\rho) < 1 \Rightarrow \rho^3 + 4\rho^2 + 4\rho < 1$ [by L21.3]
$\Rightarrow dr(2 + \rho) - (1+\rho)^{-1} < 0$ [by L21.1 and L21.2]
End Proof
Lemma 22: If $\rho < 0.2054687$
then $dr(1 + \rho) < (1+\rho)^{-1} - dr$
Proof: $\rho < 0.2054687$
$\Rightarrow dr(2 + \rho) - (1+\rho)^{-1} < 0$ [by Lemma 21]
$\Rightarrow dr(1 + \rho) + dr - (1+\rho)^{-1} < 0$
$\Rightarrow dr(1 + \rho) < (1+\rho)^{-1} - dr$
End Proof
Lemma 23: If $\rho < 0.2054687$
then $2d_{min} / ((1 + \rho)^{-1} - dr) < 2d_{min} / dr(1 + \rho)$
Proof:
$dr > 0$ [by Lemma 4] (L23.1)
$\rho > 0$ [by Axiom 1] (L23.2)
So, $1 + \rho > 0$ (L23.3)
By (L23.1) and (L23.3),
$dr(1 + \rho) > 0$ [Product of positives is positive]
$\rho < 0.2054687$ (L23.4)
$\Rightarrow dr(1+\rho) < (1+\rho)^{-1} - dr$ [by Lemma 22]
$\Rightarrow 0 < dr(1+\rho) < (1+\rho)^{-1} - dr$ [by L23.4]
$\Rightarrow 1 / [(1+\rho)^{-1} - dr] < 1 / dr(1+\rho)$ [by Lemma 3]
$\Rightarrow 2d_{min} / [(1+\rho)^{-1} - dr] < 2d_{min} / dr(1+\rho)$ [by Lemma 2(i) since $d_{min}>0$ by Axiom 7]
End Proof
Lemma 24: If (i) $\rho < 0.2054687$ and
(ii) $D_{max}[[1/dr(1+\rho)] - (1+\rho)^2] > 2d_{min} / dr(1+\rho) + Z$
then $D_{max}[[1/dr(1+\rho)] - (1+\rho)^2] > 2d_{min} / [(1+\rho)^{-1} - dr] + Z$
Proof:
(1) $2d_{min} / [(1+\rho)^{-1} - dr] < 2d_{min} / dr(1+\rho)$ [by Hypothesis (i) and Lemma 23]
(2) $2d_{min} / [(1+\rho)^{-1} - dr] + Z < 2d_{min} / dr(1+\rho) + Z$ [by (1)]
(3) $2d_{min} / dr(1+\rho) + Z < D_{max}[1/dr(1+\rho) - (1+\rho)^2]$ [by Hypothesis (ii)]
(4) $D_{max}[1/dr(1+\rho) - (1+\rho)^2] > 2d_{min} / [(1+\rho)^{-1} - dr] + Z$ [by (2) and (3)]
End Proof
Lemma 25: If (i) $\rho < 0.2054687$ and
(ii) $D_{max}[[1/dr(1+\rho)] - (1+\rho)^2] > t_{del}/dr(1+\rho)[2\rho^3+6\rho^2+6\rho+4]$
then $D_{max}[[1/(1+\rho)dr] - (1+\rho)^2] > 2d_{min}/[(1+\rho)^{-1} - dr] + Z$
Proof: We will try to apply Lemma 24 to prove this proposition.
We focus our attention to the expression $2d_{min} / dr(1+\rho) + Z$ which forms the right hand side of the antecedent of Lemma 24.
$2d_{min} / dr(1+\rho) + Z$ (L25.1)
$= 2d_{min} / dr(1+\rho) + d_{min}/dr + t_{del}/(1+\rho) + t_{del}(1+\rho)$ [by Definition 6]

$= [2d_{min} + d_{min}(1+\rho) + t_{del}dr + t_{del}(1+\rho)^2dr] / dr(1+\rho)$

$< [2d_{min} + d_{min}(1+\rho) + t_{del}(1+\rho) + t_{del}(1+\rho)^2dr] / dr(1+\rho)$ [since $dr=(1+\rho)-(1+\rho)^{-1}$ and $(1+\rho)^{-1} > 0$ and $(1+\rho) > dr$]

$< [2d_{min} + d_{min}(1+\rho) + t_{del}(1+\rho) + 2t_{del}(1+\rho)^2dr] / dr(1+\rho)$ (L25.2) [since $t_{del}(1+\rho)^2dr > 0$]

Now, $2d_{min} + d_{min}(1+\rho) + t_{del}(1+\rho) + 2t_{del}(1+\rho)^2dr$

$= t_{del}[2+(1+\rho)+(1+\rho)+2(1+\rho)^2dr]$[by Axiom 7]

$= t_{del}[2\rho^3+6\rho^2+6\rho+4]$ (L25.3) [by Axiom 5]

From (L25.1), (L25.2) and (L25.3),

$2d_{min} / dr(1+\rho) + Z < t_{del}[2\rho^3+6\rho^2+6\rho+4] / dr(1+\rho)$ (L25.4)

We have $D_{max}[[1/dr(1+\rho)] – (1+\rho)^2] > t_{del}/dr(1+\rho)[2\rho^3+6\rho^2+6\rho+4]$ [by Hypothesis (ii)]

$\Rightarrow D_{max}[[1/dr(1+\rho)] – (1+\rho)^2] > 2d_{min} / dr(1+\rho) + Z$ (L25.5) [by L25.4]

$\Rightarrow D_{max}[[1/dr(1+\rho)] – (1+\rho)^2] > 2d_{min} / [(1+\rho)^{-1} – dr] + Z$ [by Hypothesis (i), (L25.5) and Lemma 24]

End Proof

Lemma 26: If (i) $\rho < 0.2054687$

then $1/dr(1+\rho) – (1+\rho)^2 > 0$

Proof: Let $1/dr(1+\rho) – (1+\rho)^2 > 0$

$\Leftrightarrow [1-dr(1+\rho)^3] / dr(1+\rho) > 0$

$\Leftrightarrow [1-dr(1+\rho)^3] > 0$ [since $dr>0$ by Lemma 4 and $\rho>0$ by Axiom 1 so that $(1+\rho)dr > 0$]

$\Leftrightarrow \rho^4 + 4\rho^3 + 5\rho^2 + 2\rho < 1$ (L26.1) [by Axiom 5]

Let $g(\rho) = \rho^4+4\rho^3+5\rho^2+2\rho$ (L26.2)

By iterative techniques, we find that

$g(0.272) = 0.999882 < 1$ and

$g(0.275) = 1.0170316 > 1$ (L26.3)

By Lemma 7 and Lemma 8, $g(\rho)$ is monotone increasing. Hence, from (L26.3) we can infer that

$(\rho < 0.72) \Rightarrow g(\rho) < 1$ (L26.4)

From (L26.4) and (L26.1) we get

$(\rho < 0.72) \Rightarrow [1/[dr(1+\rho)] – (1+\rho)^2 > 0]$

End Proof

Lemma 27: $1 - 2\rho - 5\rho^2 - 4\rho^3 – \rho^4 = 1 – dr(1+\rho)(1+\rho)^2$

Proof: $1 – dr(1+\rho)(1+\rho)^2$

$= 1 – [(1+\rho) – (1+\rho)^{-1}] (1+\rho)(1+\rho)^2$ [by Axiom 5]

$= 1 – [(1+\rho)^2 – 1] (1+\rho)^2$

$= 1 – [2\rho + \rho^2][1 + \rho]^2$

$= 1 - 2\rho - 5\rho^2 - 4\rho^3 – \rho^4$

End Proof

Lemma 28: If (i) $\rho < 0.2054687$

and (ii) $D_{max} > t_{del}[2\rho^3+6\rho^2+6\rho+4] / [1-2\rho-5\rho^2-4\rho^3-\rho^4]$

then $D_{max}[[1/dr(1+\rho)] – (1+\rho)^2] > t_{del}/[dr(1+\rho)][2\rho^3+6\rho^2+6\rho+4]$

Proof:

(1) $D_{max} > t_{del}[2\rho^3+6\rho^2+6\rho+4] / [1-2\rho-5\rho^2-4\rho^3-\rho^4]$ [by Hypothesis (ii)]

(2) $D_{max} > t_{del}[2\rho^3+6\rho^2+6\rho+4] / [1- dr(1 + \rho)(1 + \rho)^2]$ [by (1) and Lemma 27]

(3) $D_{max} > t_{del}/[dr(1 + \rho)] [2\rho^3+6\rho^2+6\rho+4] / [1/[dr(1 + \rho)]- (1 + \rho)^2]$

(4) $\rho < 0.272$ [by Hypothesis (i)]

(5) $1/[dr(1+\rho)] – (1+\rho)^2 > 0$ [by (4) and Lemma 26]

(6) $D_{max}[[1/[dr(1+\rho)]] – (1+\rho)^2] > t_{del}/[dr(1+\rho)][2\rho^3+6\rho^2+6\rho+4]$ [by (3), (5) and Lemma 2(ii)]

End Proof

Lemma 29: If (i) $\rho < 0.2054687$

and (ii) $D_{max} > t_{del}[2\rho^3+6\rho^2+6\rho+4] / [1-2\rho-5\rho^2-4\rho^3-\rho^4]$

then $D_{max}[[1/[(1+\rho)dr]] - (1+\rho)^2] > 2d_{min}/[(1+\rho)^{-1} - dr] + Z$

Proof:

(1) $D_{max}[[1/[(1+\rho)dr]] - (1+\rho)^2] > t_{del}/[dr(1+\rho)][2\rho^3+6\rho^2+6\rho+4]$ [by Hypothesis and Lemma 28]

(2) $D_{max}[[1/[(1+\rho)dr]] - (1+\rho)^2] > 2d_{min}/[(1+\rho)^{-1} - dr] + Z$ [by Hypothesis (i), (1) and Lemma 25]

End Proof

Lemma 30: If (i) $\rho < 0.2054687$

and (ii) $D_{max} > t_{del}[2\rho^3+6\rho^2+6\rho+4] / [1-2\rho-5\rho^2-4\rho^3-\rho^4]$

then $k_2 > \alpha_2$

Proof:

(1) $D_{max}[[1/[(1+\rho)dr]] - (1+\rho)^2] > 2d_{min}/[(1+\rho)^{-1} - dr] + Z$ [by Hypothesis and Lemma 29]

(2) $k_2 > \alpha_2$ [by (1) and Lemma 20]

End Proof

Lemma 31: If $\rho < (\sqrt{2} - 1)$ then the following set of inequalities :

$P \leq \pi$

$\alpha \geq \alpha_2$

$P - \alpha - k_3 > 0$ and

$P - \alpha - k_{37} > 0$

has a solution if and only if $k_2 > \alpha_2$. [This lays down the criterion for existence of a solution]

Proof: With reference to Figure 9 we have

(1) $k_{37} > k_3$ [from Lemma 19 since $\rho < (\sqrt{2} - 1)$ by hypothesis]

(2) $P - \alpha > k_{37} \Rightarrow P - \alpha > k_3$ [by (1)]

(3) The set of inequalities to be satisfied is

$P \leq \pi$

$\alpha \geq \alpha_2$

$P - \alpha - k_{37} > 0$ [by (2) and Hypothesis]

(4) The feasible region in the $P - \alpha$ plane is the triangle ABC. [by (3) and Figure 9]

(5) The triangle ABC exists only if BC lies above A, that is, if π, the y-intercept of BC, is greater than Ay, the y-coordinate of A. [follows from Figure 9]

(6) A is the intersection of $\alpha \geq \alpha_2$ and $P - \alpha - k_{37} = 0$. So, Ay is given by $Ay - \alpha_2 - k_{37} = 0$ that is, $Ay = \alpha_2 + k_{37}$

(7) The required condition is

$\pi > Ay$

i.e, $\pi > \alpha_2 + k_{37}$

i.e., $\pi - k_{37} > \alpha_2$

or, $k_2 > \alpha_2$ [by Definition 5]

End Proof

We are now approaching the culmination of the Proof. While Lemma 18 lays down a sufficient condition for $T_r^c > T_s^k$, it does not specify whether this condition is attainable. To design the system, we must set the clock synchronization parameters P, α and D_{max}. We will now specify, in Theorem 1, how this can be done.

Definition 8: A process P_r starting C_r^{k+1} at t_1 is said to be faster than a process P_s starting C_s^{k+1} at t_2 during the k-th resynchronization interval if $t_1 < t_2$.

Theorem 1: For any two processes P_s and P_r whose k-th checkpointing instants T_s^k and T_r^k during the k-th resynchronization interval are selected by RULE-1, the instant T^c of next occurrence of event1 of the process with the faster clock is greater than the checkpointing instant of the process with slower clock, i.e.,

$\min(T_s^c, T_r^c) > \max(T_s^k, T_r^k)$

if the following conditions are satisfied

$\rho < 0.2054687$
$P - \alpha > 2d_{min} / [(1+\rho)^{-1} - dr]$ and
$D_{max} > t_{del}[2\rho^3+6\rho^2+6\rho+4] / [1-2\rho-5\rho^2-4\rho^3-\rho^4]$

Proof: With reference to Figure 8 and without loss of generality, we assume that P_r is faster than P_s (Definition 8), i.e., $t_1 < t_2$ (T1.1.)
Then we have to prove $T_r^c > T_s^k$
The proof is developed as a sequence of the following assertions:
(1) $k_2 > \alpha_2$ [by Hypothesis (i), Hypothesis(iii) and Lemma 30]
(2) $(\sqrt{2} - 1) > 0.2054687$ [Number Thoery]
(3) $\rho < (\sqrt{2} - 1)$ [by Hypothesis (i) and (2)]
(4) The set of inequalities
$P \leq \pi$
$\alpha \geq \alpha_2$
$P - \alpha - k_3 > 0$ and
$P - \alpha - k_{37} > 0$
has a solution [by (3), (1) and Lemma 31]
(5) There exists P and α: $(P - \alpha - k_{37} > 0)$ [by 4(iv)]
(6) There exists P and α: $(P - \alpha > 2d_{min} / [(1+\rho)^{-1} - dr])$ [by (5) and Definition 2]
(7) There exists P and α: $T_r^c > T_s^k$ [by (6), (3), (T1.1) and Lemma 18]
End Proof

We now proceed to prove that the checkpoints taken by the algorithm are consistent. We like to prove this by contradiction, that is, the checkpointing algorithm records checkpoints that are not consistent with each other. This means that either:
Case1: there is a missing message m that is, there is a message m that is recorded "sent" in checkpoint CP_s^k in sender P_s but not recorded "received" in the corresponding checkpoint CP_r^k of the receiver P_r or
Case2: there is an orphan message m that is, there is a message m that is recorded "received" in checkpoint CP_r^k in receiver P_r but not recorded "sent" in the corresponding checkpoint CP_s^k of the sender P_s.
We consider the following cases for each of the above Cases1 and 2:
Case 1: (sent∧¬received)
Case 2: (received∧¬sent)
Case 1.1: P_r slower, Case 1.2: P_r faster
Case 2.1: P_r slower, Case 2.2: P_r faster
Case 1.2.1: T_r^k occurs in midst of communication
Case 1.2.2: T_r^k occurs before communication
Case 2.1.1: T_s^k occurs in midst of communication
Case 2.1.2: T_s^k occurs before communication

Now let us analyze each of the above cases in details – Theorem 2 deals with Case1 and Theorem 3 deals with Case2.
The events of taking kth checkpoints by P_s at T_s^k and by P_r at T_r^k are denoted by EC_s^k and EC_r^k respectively. Similarly, the event of clock synchronization in any process P_i is denoted by ECl_i^c. It must be mentioned here that, neither CP_s^k nor CP_r^k happen before each other under the scenario described in Case1. Hence it is not mandatory that messages recorded "sent" in CP_s^k should also have to be recorded "received" in CP_r^k, but may have to be dealt with in special ways. However, in the present work it has been proved that the algorithm would not give rise to either any missing message or any orphan message.

Theorem 2: There is no message m that is recorded "sent" in checkpoint CP_s^k in sender P_s but not recorded "received" in the corresponding checkpoint CP_r^k of the receiver P_r

Proof:

Case1.1: P_r, the receiver is slower than the sender P_s. Figure 10 shows a situation where message m would be recorded 'sent' but not recorded 'received' though it does not reflect the assumption that P_r is slower than P_s.

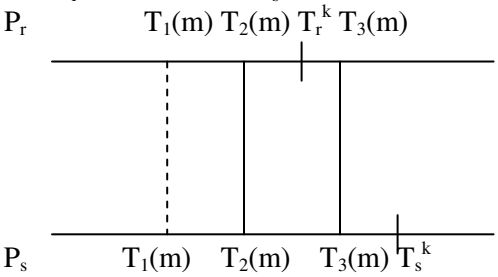

Figure 10

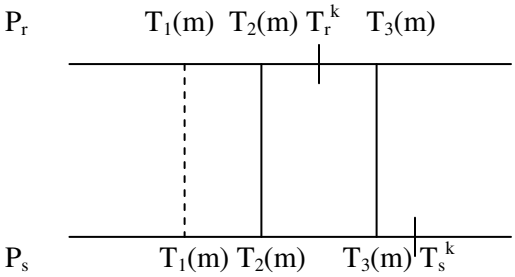

Figure 11

Assumptions:

Message m recorded sent and message m not recorded received

P_r is slower than P_s, so $EC_s^k \rightarrow EC_r^k$

The following scenario is observed:

Checkpoint CP_s^k at P_s is at T_s^k $(E_3(m) \rightarrow EC_s^k$ by (i))

Checkpoint CP_r^k at P_r is at T_r^k $(EC_r^k \rightarrow E_3(m)$ by (i))

Therefore, $EC_r^k \rightarrow EC_s^k$ (from (1) and (2))

From the above we have (3) contradicts (ii) meaning (3) $\equiv \neg$ (ii)

So, there can not be a message m that is recorded sent in checkpoint CP_s^k in sender P_s but not recorded received in the corresponding checkpoint CP_r^k of the receiver P_r

Case1.2: P_r, the receiver is faster than the sender P_s. Figure 11 shows a situation where message m would be recorded sent but not recorded received though it does not reflect the assumption that P_r is faster than P_s. Here, either of the following cases may happen: 1.2.1. Receiver P_r is blocked in communication at the checkpointing instant T_r^k or, 1.2.2. Receiver P_r has taken checkpoint earlier than sender and is not blocked at checkpointing instant T_r^k.

Subcase 1.2.1: Assumptions:

Message m recorded sent and message m not recorded received

P_r is faster than P_s, so $EC_r^k \rightarrow EC_s^k$

P_r is blocked at T_r^k

The following scenario is observed:

Checkpoint CP_s^k at P_s is at T_s^k $(E_3(m) \rightarrow EC_s^k$ by i)

Checkpoint at CP_r^k P_r is at T_r^k $(EC_r^k \rightarrow E_3(m)$ by i)

3. Assuming T_r^k occurs in midst of communication, that is, P_r is blocked at T_r^k (by (iii)):

P_r reaches step 3 of algorithm take_ckpt() following steps 1,2.

P_r reaches step 3.1 of algorithm, takes checkpoint, could not record the "receive" of m since it is not yet complete and reaches step 3.2.

P_r eventually reaches step 3.5.2 of algorithm from steps 3.3, 3.4 and 3.5.

By Theorem 1, $T_r^c > T_s^k$ and by (1) $T_s^k > T_3(m)$. Thus $T_r^c > T_3(m)$

P_r reaches step 3.6, that is, it had already executed step 3.2 which terminated at time T_r^c. So, step 3.6 is executed at a time $T_{ex} > T_r^c > T_3(m)$.

In step 3.6.1, P_r again saves state and that saving instant $> T_3(m)$ (by (e)). So, P_r records the receipt of message m. This contradicts hypothesis (i), so, this scenario can not occur.

Subcase 1.2.2: With assumptions (i) and (ii) remaining same and the scenario observed from (1) to (2) remaining same we have:

Assuming T_r^k occurs before P_r has executed its synchronous "receive" statement, $T_r^k < T_2(m)$, that is, P_r is not blocked at T_r^k

P_r reaches step 2.1 and hence 2.2 and thereby freezes after taking CP_r^k at T_r^k till clock synchronization T_r^c occurs. In other words, P_r does not participate in any event in the interval (T_r^k, T_r^c)

b) $T_2(m)$ denotes the time of an event in which P_r participates (by step 2 and definition of $T_2(m)$)

c) $T_2(m) \notin [T_r^k, T_r^c]$ (by 3a and 3b)

d) (3c) implies that $((T_2(m) < T_r^k)$ OR $(T_2(m) > T_r^c))$

e) (3d) gives (FALSE OR $(T_2(m) > T_r^c)$) (since by 3, $T_2(m) > T_r^k$)

f) So, $T_2(m) > T_r^c$ (by 3e)

g) $T_r^c > T_s^k$ (by Theorem 1)

h) $T_s^k > T_2(m)$ (since $T_s^k > T_3(m)$ (by Assumption (i)) and $T_3(m) > T_2(m)$ (by definition))

i) $T_r^c > T_2(m)$ (by 3g and 3h)

j) Therefore, 3i contradicts 3f, that is, 3i == ¬ 3f .

So, there can not be any message m that is recorded sent but not recorded received in the same checkpointing interval under the above assumptions.

Theorem 3: There is no message m that is recorded "received" in checkpoint CP_r^k in receiver P_r but not recorded "sent" in the corresponding checkpoint CP_s^k of the sender P_s.

Proof:

Case2.1: Figure 12 and Figure 13 show situations where message m would be recorded received but not recorded sent. First we consider the case where P_r, the receiver is slower than the sender P_s. Here, either of the following cases may happen: 2.1.1. Sender P_s is blocked in communication during the checkpointing instant T_s^k or, 2.1.2. Sender P_s has taken checkpoint earlier than receiver and is not blocked during checkpointing instant T_s^k.

For 2.1.1 let us consider figure 12 where the dashed line denotes a timing instant T where the following situation is observed.

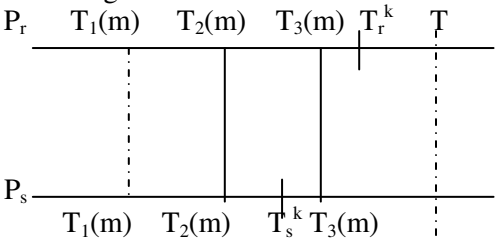

Figure 12

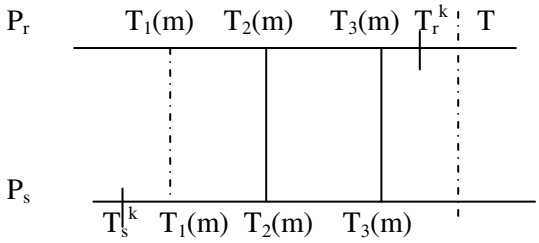

Figure 13

Subcase 2.1.1. Assumptions:

Message m recorded received and message m not recorded sent

P_s is faster than P_r, so $EC_s^k \rightarrow EC_r^k$

P_s is blocked at T_s^k

The following scenario is observed:

Checkpoint at P_s is at T_s^k $(EC_s^k \rightarrow E_3(m)$ by (i))

Checkpoint at P_r is at T_r^k $(E_3(m) \rightarrow EC_r^k$ by (i))

Assuming P_s is blocked at T_s^k (by (iii))

P_s reaches steps 3.1 and 3.2 of algorithm

In step 3.4 of algorithm P_s calls wait(event2) after T_s^k.

Let T_r^c = Instant of first occurrence of event1 in P_r after T_r^k and

T_s^c = Instant of first occurrence of event1 in P_s after T_s^k

By (ii) $T_s^c < T_r^c$

By Theorem 1 $T_s^c > T_r^k$
By (3d) and (3e) $T_r^k < T_r^c$
P_s can not terminate wait(event2) before it receives sp_mess from P_r (by definition of event2)
P_r can send sp_mess to P_s only after T_r^c (by Srikanth[15])
4. a) P_r is not blocked at T_r^k (by assumption (i))
b) P_r takes checkpoint in step 2.1 of the algorithm at an instant $\geq T_r^k$ (by 4(a) and algorithm)
c) P_r calls wait(event1) in step 2.2. after T_r^k (by 4(b) and sequential execution semantics)
d) P_r terminates wait(event1) after T_r^c (by 3(f))
e) P_r executes step 2.3.1 (by 4(d)) to construct sp_mess after T_r^c
f) $T_r^c > T_3(m)$ (by (2) and 3(f))
g) P_r records the receipt of message m in sp_mess (by 4(e) and 4(f))
5. P_s detects inconsistency in step 3.5.1 of algorithm
6. P_s modifies checkpoint CP_s^k in step 3.7 of the algorithm.
7. By (6) we have P_s recording the send of message m thereby contradicting assumption (i).
Hence the scenario 2.1.1 can not occur.
Subcase 2.1.2. Assumptions:
Message m not recorded sent by P_s but recorded received by P_r that is, $T_s^k < T_3(m)$ and $T_r^k > T_3(m)$
$T_s^k < T_r^k$ (since P_s is faster)
P_s is not blocked at T_s^k that is, $T_s^k < T_2(m)$
The following scenario is observed:
P_s begins executing algorithm take_ckpt() at T_s^k (by definition of T_s^k)
2. P_s reaches step 2.2. of algorithm and commences wait(event1) at time $T_x > T_s^k$ (by (iii) P_s is not blocked)
P_s can not participate in any event in the interval $[T_x, T_s^c]$where T_s^c is the time of occurrence of event1 in P_s (by definition of wait(event1))
$T_2(m)$ denotes the time of an event in which P_s participates (by definition of $T_2(m)$ and since P_s sends message m)
$T_2(m) \notin [T_x, T_s^c]$ (by (3) and (4))
$(T_2(m) < T_x)$ OR $(T_2(m) > T_s^c)$ (by (5))
P_s is executing algorithm in the interval $[T_s^k, T_x]$ and hence can not participate in message transfer in this interval (by (1) and (2))
$T_2(m) \notin [T_s^k, T_x]$ (by (4) and (7))
$T_2(m) > T_s^k$ (by (8) and (iii))
From (6) using (9): (FALSE) OR $(T_2(m) > T_s^c)$
So, $T_2(m) > T_s^c$ (by (10)
Therefore, $ECl k_s^c \rightarrow E_2(m)$ (by (11))
From Theorem 1 we have $T_s^c > T_r^k$, that is, $EC_r^k \rightarrow ECl k_s^c$
Therefore, $EC_r^k \rightarrow E_2(m)$ (by (12) and (13))
Since $E_2(m) \rightarrow E_3(m)$ (section 2) and from (14) we have $EC_r^k \rightarrow E_3(m)$
(15) therefore contradicts the scenario observed in (i) that is, $E_3(m) \rightarrow EC_r^k$ or, $T_r^k > T_3(m)$
Thus it is proved that there can not be any message m that is recorded received but not recorded sent under scenario 2.1.2.
Case2.2: Figure 14 shows a situation where message m would be recorded 'received' but not recorded 'sent'. P_r, the receiver is faster than the sender P_s. Let us consider figure14 (though it does not reflect the fact that P_r is faster than P_s) where the dashed line denotes a timing instant T where the following situation is observed.

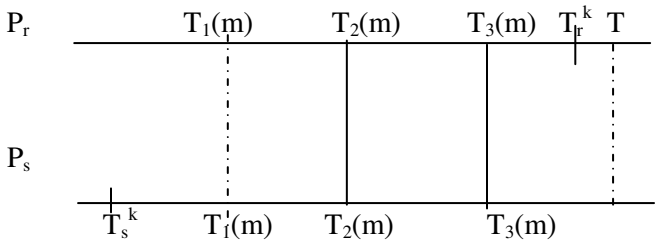

Figure 14

Assumptions:
Message m recorded received and message m not recorded sent
P_r is faster than P_s, so $EC_r^k \rightarrow EC_s^k$
The following scenario is observed:
Checkpoint at P_s is at T_s^k ($EC_s^k \rightarrow E_3(m)$ by i)
Checkpoint at P_r is at T_r^k ($E_3(m) \rightarrow EC_r^k$ by i)
Therefore, $EC_s^k \rightarrow EC_r^k$ (by 1 and 2)
4. (3) contradicts (ii)
So, there can not be any message m that is recorded received but not recorded sent under scenario 2.2.
Theorem 4: The checkpoints taken by the algorithm are consistent.
Proof: We would prove the theorem by contradiction. Let the checkpoints taken by the algorithm be inconsistent. This means that either:
Case1: there is a missing message m, or
Case2: there is an orphan message m
Case1: It has been proved in Theorem 2 above that there can not be any missing message m
Case2: It has been proved in Theorem 3 above that there can not be any orphan message m
From Cases (1) and (2), which include all possible cases of inconsistencies, we see from above that inconsistency can not occur in any of the cases.
This ends the proof of Theorem 4.
5.4. Approximate time of freezing after taking checkpoint
In a multiprogrammed environment CPU time-slice generally belongs to the range of 10 msec to 50 msec. Let us take the degree of multiprogramming to be 16 and average CPU time-slice to be 25 msec. So, average waiting time per process W_m approximately becomes : 375 msec (= (16 – 1) * 25) msec). Typical round trip time (RTT) for a cross-country connection in USA is 100 msec [11]. So, we choose t_{del} (Section 5.3) to be approximately equal to 50 msec in the following calculations.
It is already stated in section 5.3 (in Proof of Theorem 1) that the checkpointing instant is chosen to be: $kP + \alpha + 0.5P$ (2)
and thus the immediate next clock synchronization instant would be
$(k+1)P + \alpha$ (3)
Hence maximum "freezing" time of a process is calculated to be 0.5P (by subtracting (2) from (3)). From Srikanth [16] we have

$$P > t_{del}(1+\rho) + \alpha \quad (4)$$

We also have from Srikanth [16] the following relationship:

$$\alpha \geq [(1+\rho)D_{max} + t_{del}](1+\rho) \quad (5)$$

For small ρ: $\alpha \geq D_{max} + t_{del}$ (6)
Let $\rho = 10^{-5}$ and $t_{del} = 50$ msec
We choose $D_{max} = 1.25 * t_{del}$ (since $D_{max} \geq t_{del}$ from (5a) in section 5.3)) = 62.5 msec
So, from (5) we have $\alpha \geq 112.5$ (= 50 + 62.5)

We choose α = 150 msec and from (4) we have
$P > 50*(1+10^{-5}) + 150$ msec
Or $P > 200$ msec (7)
From (5) in section 5.3 we have $P \leq (D_{max} - t_{del}) / 2*\rho$
Or, $P \leq 6.25*10^5$ msec (8)
However, choice of P (the clock resynchronization interval) depends upon the drift rates of the hardware clocks of the various processors of the system. Freezing time lies in the range (100 msec, 312.5 sec) and is obviously dependent upon the choice of P. So, if P is chosen to be not too large compared to W_m, the blocking time is tolerable. But, it may be such that the drift rates are within bounds that is, the clocks are well synchronized and hence P is closer to the upper limit of 625 sec (as shown in (8)). So, freezing time becomes an overhead and that is not desirable. In such situation the scheme presented in the paper can be modified as follows: after each checkpoint a process sends a special message containing status information of itself to all other processes for checkpoint synchronization purposes. Hence explicit checkpoint synchronization messages are exchanged and no time is wasted in blocking. On the other hand, clock synchronization messages are utilized for piggybacking the checkpoint synchronization information, as discussed in the paper, for systems where the clock resynchronization period is reasonably small.

6. SIMULATION

6.1. SIMULATION PRINCIPLES

In the simulation that we have carried out to evaluate the performance of our algorithm, we have measured the following variables:
the performance metric R = (number of forced checkpoints)/(number of messages passed) with instantaneous state-saves, the execution time of the same computation without checkpointing and the overhead of freezing (suspension time) in our algorithm.
The independent variable is:
BCF (Basic Checkpoint Frequency) = (Checkpoint period)/(Total execution time)
The parameters that are varied in the expression are:
NODES, the number of nodes in the network; and
MEAN-GRAIN, the average time for which a process uses the CPU before requesting a message transfer.
Experiment-1: MEAN-GRAIN=0.16 sec and NODES is varied from 6 to 27. We observe that for NODES in the range 15 to 21, there are several forced checkpoints.
Experiment-2: NODES = 20 while MEAN-GRAIN is varied from 0.085 sec to 0.13 sec. We observe that the number of forced checkpoints is higher for MEAN-GRAIN values around 0.115 sec.

6.2. SIMULATION RESULTS

I. Application Program parameters: MEAN-GRAIN=0.16 sec, NODES = 27

Checkpoint parameters: TSAVESTATE=0.000000 Measuring overhead for freezing time only
Clock Synchronization parameters: P=8.000000, RHO=0.000010,TDEL=0.065000,ALPHA=0.300000

Performance Summary: (All time values are in seconds)

I	BCF	R	cut_off_time (with ckpt)	exec_time (NO ckpt)	Chkpt _ count	forced ckpts	msgs_ passed	Overhead (%)
0	0.001	0.00002180	8000.00	4031.12	1030	2	91763	98.46
1	0.002	0.00000000	8000.00	6014.73	515	0	136955	33.01
2	0.003	0.00000000	8000.00	6674,14	344	0	152405	19.87
3	0.004	0.00000000	8000.00	7005.94	258	0	160018	14.19
4	0.005	0.00000000	8000.00	7206.13	206	0	164339	11.02
5	0.006	0.00000000	8000.00	7337.13	172	0	167816	9.03
6	0.010	0.00000000	8000.00	7603.2	103	0	172768	5.22
7	0.022	0.00000000	8000.00	7818.70	47	0	179261	2.32
8	0.046	0.00000000	8000.00	7911.30	23	0	180243	1.12

II. Application Program parameters: MEAN-GRAIN=0.100000 sec, NODES = 20
Checkpoint parameters: TSAVESTATE=0.000000 Measuring overhead for freezing time only
Clock Synchronization parameters: P=8.000000, RHO=0.000010,TDEL=0.065000,ALPHA=0.300000
Our general observation from the two different parameters is that the number of forced checkpoints is extremely small, which automatically implies that our algorithm is extremely efficient.
The overhead of freezing time (i.e., the time for which a process remains suspended after taking a checkpoint) is higher at low values of BCF, i.e, at high checkpointing frequencies. For a BCF of 0.010, the overhead can be as low as 5.23%. Thus, if the execution time with instantaneous checkpoints is 8000 sec during which 103 checkpoints are taken, the execution time with no checkpointing would be 7602.74 sec. With NODES = 6 and MEAN-GRAIN = 0.16 sec, there would be 55552 messages exchanged among the nodes.
The highest value of R = 0.00006022 that we have obtained is orders of magnitude lower than R = 0.35 of [22].

Performance Summary: (All time values are in seconds)

I	BCF	R	cut_off_time (with ckpt)	exec_time (NO ckpt)	chkpt_ count	forced ckpts	msgs_ passed	Overhead (%)
0	0.001	0.00000000	8000.00	4027.24	1030	0	115250	98.65
1	0.002	0.00000000	8000.00	6013.36	515	0	172335	33.04
2	0.003	0.00000000	8000.00	6673.08	344	0	190203	19.88
3	0.004	0.00000000	8000.00	7004.65	258	0	200944	14.21
4	0.005	0.00000000	8000.00	7205.52	206	0	205632	11.03
5	0.006	0.00000000	8000.00	7336.92	172	0	210144	9.04
6	0.010	0.00000000	8000.00	7602.50	103	0	218196	5.23
7	0.022	0.00000892	8000.00	7818.73	47	2	224092	2.32
8	0.046	0.00000000	8000.00	7911.02	23	0	226798	1.12

7. CONCLUSION

We have presented here a synchronized-clock-based checkpointing method. This method does not have a central checkpoint coordinator that poses a single point failure threat to the system. The overhead of coordinating messages is also absent since clock synchronization messages have been utilized for checkpoint-synchronizing purposes. Though frequency of checkpointing is basically driven by the requirement of the application program the time for which processes have to wait to synchronize their checkpointing action is not an overhead since the clock synchronization frequency can be adjusted to meet the requirement. At the same time it must be ensured that clock synchronization itself does not become an overhead. So, coordinating messages for synchronizing the checkpointing action may be used in cases where the clocks are well synchronized. Since any global checkpoint is consistent, only one checkpoint needs to be stored in the stable storage [7]. The old checkpoint in a process is deleted once checkpointing synchronization is over and the new checkpoint is written. So, the system does not have to roll back more than once to restart from a previous consistent state in case recovery is required.

REFERENCES

1. K.M.Chandy, & L.Lamport, (1985) Distributed Snapshots: Determining Global States of Distributed Systems, *ACM Trans. On Computer Systems*, Vol. 3, No.1, pp. 63-75.
2. S.Kalaiselvi & V.Rajaraman, (1997) Checkpointing Algorithm for Parallel Computers based on Bounded Clock Drifts; *Computer Science & Informatics*, Vol. 27, No. 3, pp. 7-11
3. R.Koo & S.Toueg, (1987) Checkpointing and Rollback Recovery for Distributed Systems, *IEEE Trans. on Software Engineering*, Vol. SE-13, No.1, pp. 28-31.
4. L. Lamport, (1978) Time, Clocks and the Ordering of Events in a Distributed System, *Communications of the ACM*, Vol. 21, No. 7, pp. 558-565.

5. D.Manivannan, R.H.B.Netzer & M.Singhal, (1997) Finding Consistent Global Checkpoints in a Distributed Computation, *IEEE Trans. On Parallel & Distributed Systems*, Vol.8, No.6, pp. 623-627.

6. D.Manivannan, (1999) Quasi-Synchronous Checkpointing:Models, Characterization, and Classification; *IEEE Trans. on Parallel and Distributed Systems*, Vol.10, No.7, pp703-713.

7. S.Neogy, A.Sinha & P.K.Das, (2001) Checkpoint processing in Distributed Systems Software Using Synchronized Clocks; IEEE Proceedings of the International Conference on Information Technology: Coding and Computing: ITCC 2001, pp. 555-559.

8. S.Neogy, A.Sinha & P.K.Das, (2002) Distributed Checkpointing Using Synchronized Clocks; IEEE Proceedings of the 26th International Computer Software COMPSAC 2002, pp. 199 - 206.

9. S. Neogy & P. K. Das, (2002) A Reliable Time-out-free Fault-Tolerant Architecture without Dynamic Reconfiguration, Proceedings of the 28th Annual Convention and Exhibition of IEEE India Council (IEEE ACE 2002), pp. 200 – 203.

10. N.Neves, K.W.Fuchs, Using Time to Improve the Performance of Coordinated Checkpointing, *http://composer.ecn.purdue.edu/~fuchs/fuchs/ipdsNN96.ps*

11. N.Neves, K.W.Fuchs, Coordinated Checkpointing without Direct Coordination, http://composer.ecn.purdue.edu/~fuchs/fuchs

12. Larry L. Peterson, Bruce S. Davie, (2000) Computer Networks – A Systems Approach, *Harcourt Asia PTE Ltd*.

13. P. Ramanathan, K.G.Shin, (1993) Use of Common Time Base for Checkpointing and Rollback Recovery in a Distributed System; *IEEE Trans. On Sofware Engg*., Vol.19, No.6, pp. 571-583.

14. B.Randell, (1975) System Structure for Software Fault Tolerance; *IEEE Trans. On Software Engg.*, Vol. SE-1, No.2, pp. 220-232.

15. R.H.B. Netzer & J. Xu, (1993) Adaptive Independent Checkpointing for Reducing Rollback Propagation, *Technical Report*, CS-93-25.

16. A.Sinha, P.K.Das, P.K. D.Basu, (1998) Implementation and Timing Analysis of Clock Synchronization on a Transputer based replicated system, *Information & Software Technology*, 40(1998), pp. 291-309.

17. T.K.Srikanth, S.Toueg, (1987) Optimal Clock Synchronization; *JACM*, Vol. 34, No.3, pp. 626-645.

18. Z.Tong, Y.K.Richard & W.T.Tsai, (1992) Rollback Recovery in Distributed Systems Using Loosely Synchronized Clocks; *IEEE Trans. On Parallel & Distributed Systems*, Vol. 3, No.2, pp. 246-251.

19. J.Tsai & S.Kuo, (1998) Theoretical Analysis for Communication-Induced Checkpointing Protocols with Rollback-Dependency Trackability; *IEEE Trans. On Parallel & Distributed Systems*, Vol.9, No.10, pp. 963-971.

20. R. H. B. Netzer & J. Xu, (1993) Adaptive Independent Checkpointing for Reducing Rollback Propagation, *CS-93-25*.

21. D. Manivannan, R. H. B. Netzer & M. Singhal, (1997) Finding Consistent Global Checkpoints in a Distributed Computation, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 8, No. 6, pp. 623-627.

22. R. Baldoni, J-M. Helary, A. Mostefaoui & M. Raynal, (1997) A Communication induced Checkpointing Protocol that ensures Rollback Dependancy Trackability, *Proceedings of the 27th International Symposium on Fault Tolerant Computing,* pp. 68-77.

23. R. Baldoni, J. M. Helary & M. Raynal, (2001) Rollback-Dependency Trackability: A Minimal Characterization and its Protocol, *Information and Computation*, Vo. 165, No. 2, pp. 144-173.

Authors:
Sarmistha Neogy received her Ph.D. degree from Jadavpur University and is Reader in the Dept. of Computer Science & Engineering, Jadavpur University, Kolkata, India
Anupam Sinha received his Ph.D. degree from Jadavpur University and is Professor in the Dept. of Computer Science & Engineering, Jadavpur University, Kolkata, India
P. K. Das is currently working as Dean in the Faculty of Engg. And Technology, Mody Institute of Technology & Science, Rajasthan, India. Previously he was Professor in Dept. of Computer Science & Engineering, Jadavpur University, Kolkata, India