IJCSI International Journal of Computer Science Issues, Vol. 7, Issue 3, No 5, May 2010
ISSN (Online): 1694-0784
ISSN (Print): 1694-0814

Soft-Checkpointing Based Coordinated Checkpointing Protocol for Mobile Distributed Systems

Parveen Kumar[1], Rachit Garg[2]

[1]Meerut Institute of Engineering & Technology, Department of Computer Science & Engineering, Meerut (INDIA)- 250005
[2]Singhania University, Department of Computer Science & Engineering, Pacheri Bari (Rajasthan), India

***Abstract*:** Minimum-process coordinated checkpointing is a suitable approach to introduce fault tolerance in mobile distributed systems transparently. It may require blocking of processes, extra synchronization messages or taking some useless checkpoints. All-process checkpointing may lead to exceedingly high checkpointing overhead. To optimize both matrices, the checkpointing overhead and the loss of computation on recovery, we propose a hybrid checkpointing algorithm, wherein an all-process coordinated checkpoint is taken after the execution of minimum-process coordinated checkpointing algorithm for a fixed number of times. In the minimum-process coordinated checkpointing algorithm; an effort has been made to optimize the number of useless checkpoints and blocking of processes using probabilistic approach and by computing an interacting set of processes at beginning. We try to reduce the loss of checkpointing effort when any process fails to take its checkpoint in coordination with others. We reduce the size of checkpoint sequence number piggybacked on each computation message.

1. Introduction

Mobile Hosts (MHs) are increasingly becoming common in distributed systems due to their availability, cost, and mobile connectivity. An MH is a computer that may retain its connectivity with the rest of the distributed system through a wireless network while on move. An MH communicates with the other nodes of the distributed system via a special node called mobile support station (MSS). A "cell" is a geographical area around an MSS in which it can support an MH. An MSS has both wired and wireless links and it acts as an interface between the static network and a part of the mobile network. Static nodes are connected by a high speed wired network [1].

A checkpoint is a local state of a process saved on the stable storage. In a distributed system, since the processes in the system do not share memory, a global state of the system is defined as a set of local states, one from each process. The state of channels corresponding to a global state is the set of messages sent but not yet received. A global state is said to be "consistent" if it contains no orphan message; i.e., a message whose receive event is recorded, but its send event is lost [5]. To recover from a failure, the system restarts its execution from the previous consistent global state saved on the stable storage during fault-free execution. This saves all the computation done up to the last checkpointed state and only the computation done thereafter needs to be redone.

In coordinated or synchronous checkpointing, processes take checkpoints in such a manner that the resulting global state is consistent. Mostly it follows the two-phase commit structure [2], [5], [6], [7], [10], [15]. In the first phase, processes take tentative checkpoints, and in the second phase, these are made permanent. The main advantage is that only one permanent checkpoint and at most one tentative checkpoint is required to be stored. In the case of a fault, processes rollback to the last checkpointed state [6]. The Chandy-Lamport [5] algorithm is the earliest non-blocking all-process coordinated checkpointing algorithm.

The existence of mobile nodes in a distributed system introduces new issues that need proper handling while designing a checkpointing algorithm for such systems [1], [4], [14], [16]. These issues are mobility, disconnections, finite power source, vulnerable to physical damage, lack of stable storage etc. Prakash and Singhal [14] proposed a nonblocking minimum-process coordinated checkpointing protocol for mobile distributed systems. They proposed that a good checkpointing protocol for mobile distributed systems should have low overheads on MHs and wireless channels; and it should avoid awakening of an MH in doze mode operation. The disconnection of an MH should not lead to infinite wait state. The algorithm should be non-intrusive and it should force minimum number of processes to take their local checkpoints. In minimum-process coordinated checkpointing algorithms, some blocking of the

ISSN (Online): 1694-0784
ISSN (Print): 1694-0814

processes takes place [3], [10], [11], or some useless checkpoints are taken [4], [15].

In minimum-process coordinated checkpointing algorithms, a process P_i takes its checkpoint only if it a member of the minimum set (a subset of interacting process). A process P_i is in the minimum set only if the checkpoint initiator process is transitively dependent upon it. P_j is directly dependent upon P_k only if there exists m such that P_j receives m from P_k in the current checkpointing interval [CI] and P_k has not taken its permanent checkpoint after sending m. The ith CI of a process denotes all the computation performed between its ith and $(i+1)^{th}$ checkpoint, including the ith checkpoint but not the $(i+1)^{th}$ checkpoint.

Cao and Singhal [4] achieved non-intrusiveness in the minimum-process algorithm by introducing the concept of mutable checkpoints. Kumar and Kumar [21] proposed a minimum-process coordinated checkpointing algorithm for mobile distributed systems, where the number of useless checkpoints and the blocking of processes are reduced using a probabilistic approach. Singh and Cabillic [20] proposed a minimum-process non-intrusive coordinated checkpointing protocol for deterministic mobile systems, where anti-messages of selective messages are logged during checkpointing. Higaki and Takizawa [8], and Kumar et al [17] proposed hybrid checkpointing protocols where MHs checkpoint independently and MSSs checkpoint synchronously. Neves et al. [13] gave a time based loosely synchronized coordinated checkpointing protocol that removes the overhead of synchronization and piggybacks integer csn (checkpoint sequence number). Pradhan et al [19] had shown that asynchronous checkpointing with message logging is quite effective for checkpointing mobile systems.

In the present study, we present a hybrid scheme, where an all process checkpoint is enforced after executing minimum-process algorithm for a fixed number of times as in [22]. In the first phase, the MHs in the minimum set are required to take soft checkpoint only. Soft Checkpoint is stored on the disk of the MH and is similar to mutable checkpoint [4]. In the minimum-process algorithm, a process takes its forced checkpoint only if it is having a good probability of getting the checkpoint request; otherwise, it buffers the received messages as in [21].

2. The Proposed Checkpointing Algorithm

2.1 System Model

The system model is similar to [3], [4]. A mobile computing system consists of a large number of MH's and relatively fewer MSS's. The distributed computation we consider consists of n spatially separated sequential processes denoted by P_0, P_1, ..., P_{n-1}, running on fail-stop MH's or on MSS's. Each MH or MSS has one process running on it. The processes do no share common memory or common clock. Message passing is the only way for processes to communicate with each other. Each process progresses at its own speed and messages are exchanged through reliable channels, whose transmission delays are finite but arbitrary. We assume the processes to be non-deterministic.

2.2 Basic Idea

Similar to [3], [21], [22], initiator process collects the dependency vectors of all processes and computes the tentative minimum set. Suppose, during the execution of the checkpointing algorithm, P_i takes its checkpoint and sends m to P_j. P_j receives m such that it has not taken its checkpoint for the current initiation and it does not know whether it will get the checkpoint request or not. If P_j takes its checkpoint after processing m, m will become orphan. In order to avoid such orphan messages, we use the following technique as mentioned in [21].

If P_j has sent at least one message to a process, say P_k, and P_k is in the tentative minimum set, there is a good probability that P_j will get the checkpoint request. Therefore, P_j takes its mutable checkpoint before processing m [4]. In this case, most probably, P_j will get the checkpoint request and its mutable checkpoint will be converted into permanent one. Alternatively, this message is buffered P_j. P_j will process m only after taking its tentative checkpoint or after getting commit as in [22].

In minimum-process checkpointing, some processes may not be included in the minimum set for several checkpoint initiations due to typical dependency pattern; and they may starve for checkpointing. In the case of a recovery after a fault, the loss of computation at such processes may be unreasonably high [22]. In Mobile Systems, the checkpointing overhead is quite high in all-process checkpointing [14]. Thus, to balance the checkpointing overhead and the loss of computation on recovery, we design a hybrid checkpointing algorithm for mobile distributed systems, where an all-process checkpoint is taken after certain number of minimum-process checkpoints. We enforce the all-process checkpointing protocol after executing the minimum-process algorithm for fifteen number of times.

In coordinated checkpointing, if a single process fails to take its checkpoint; all the checkpointing effort goes waste, because, each process has to abort its tentative checkpoint. In order to take the tentative checkpoint, an MH needs to transfer large checkpoint data to its local MSS over wireless channels. Hence, the loss of checkpointing effort may be exceedingly high. Therefore, we

ISSN (Online): 1694-0784
ISSN (Print): 1694-0814

propose that in the first phase, all concerned MHs will take soft checkpoint only. Soft checkpoint is similar to mutable checkpoint [4], which is stored on the memory of MH only. In this case, if some process fails to take checkpoint in the first phase, then MHs need to abort their soft checkpoints only. The effort of taking a soft checkpoint is negligible as compared to the tentative one. When the initiator comes to know that all relevant processes have taken their soft checkpoints, it asks all relevant processes to come into the second phase, in which, a process converts its soft checkpoint into tentative one. Finally, the initiator issues the commit request.

2.3 Data Structures

Here, we describe the data structures used in the proposed checkpointing protocol. A process that initiates checkpointing is called initiator process and its local MSS is called initiator MSS. Data structures are initialized on completion of a checkpointing process if not mentioned explicitly. A process is in the cell of an MSS if it is running on the MSS or on an MH supported by it. It also includes the processes running on MH's, which have been disconnected from the MSS but their checkpoint related information is still with this MSS.

i) Each process P_i maintains the following data structures, which are preferably stored on local MSS:

$d_vect_i[]$: a bit vector of size n; ; $d_vect_i[j]$ =1 implies P_i is directly dependent upon P_j for the current CI;

pr_block_i: a flag which indicates that P_i is in blocking state;

$pr_c_state_i$: a flag; set to '1' on soft or mutable checkpoint or on the receipt of a message of higher csn during checkpointing;

$mutable_i$: a flag; set to '1' on mutable checkpoint; reset on commit/abort or on tentative checkpoint;

$pr_sendv_i[]$: a bit vector of size n; $pr_sendv_i[j]$=1 implies P_i has sent at least one message to P_j in the current CI;

pr_send_i: a flag indicating that P_i has sent at least one message since last checkpoint;

pr_csn four bits checkpoint sequence no; initially, for a process pr_csn and pr_next_csn are [0000] and [0001] respectively; pr_csn is incremented as follows: pr_csn=pr_next_csn; pr_next_csn=modulo 16 (++pr_next_csn) ;

ii) Initiator MSS (any MSS can be initiator MSS) maintains the following Data structures:

mset[]: a bit vector of size n; mset[k]=1 implies P_k belongs to the minimum set; computation of minimum set on the bases of dependency vectors of all processes is given in [21].

R1[]: a bit vector of length n; R[i] =1 implies P_i has taken its soft checkpoint in the first phase;

R2[]: a bit vector of length n; R2[i] =1 implies P_i has taken its tentative checkpoint in the second phase;

Timer1: a flag; initialized to '0' when the timer is set; set to '1' when maximum allowable time for collecting coordinated checkpoint expires;

iii) MSS (say MSSp) maintains the following data structures:

$mss_local_p[]$: a bit vector of length n; mss_local_p [i]=1 implies P_i is running in the cell of MSS_p;

$mss_loc_tent_p[]$: a bit vector of length n; $mss_loc_tent_p[i]$=1 implies P_i has taken its tentative checkpoint at MSS_p;

$mss_loc_soft_p[]$: a bit vector of length n; $mss_loc_soft_p[i]$=1 implies P_i has taken its soft checkpoint in the first phase and P_i is local to MSS_p;

$mss_tent_req_p[]$: a bit vector of length n; $mss_tent_req_p[i]$=1 implies tentative checkpoint request has been sent to process P_i and P_i is local to MSS_p;

$mss_soft_req_p[]$: a bit vector of length n; $mss_soft_req_p[i]$=1 implies soft checkpoint request has been sent to process P_i in the first phase and P_i is local to MSS_p;

mss_fail_bit: a flag; set to '1' when some relevant process in its cell fails to take its checkpoint;

P_{in}: initiator process identification;

g_chkpt: a flag; set to '1' on the receipt of ddv[] request; It controls multiple checkpoint initiations;

rec_mset a flag; set to 1 on the receipt of mset[] from the initiator MSS; set to '0' on commit/abort;

new_set[] a bit vector of length n; it contains all new processes found for the minimum set at the MSS; on each checkpoint request: if ($tnew_set \neq \phi$) $new_set = new_set \cup tnew_set$;

Tnew_set[] a bit vector of length n; it contains the new

ISSN (Online): 1694-0784
ISSN (Print): 1694-0814

processes found for the minimum set while executing a particular checkpoint request. When a process, say P_i, takes its soft checkpoint, it may find some process P_j such that P_i is dependent upon P_j and P_j is not in the tentative minimum set known to the local MSS;

2.4 An Example

We explain the proposed minimum-process checkpointing algorithm with the help of an example. In Figure 1, at time t_1, P_1 initiates checkpointing process and sends request to all processes for their dependency vectors. At time t_2, P_1 receives the dependency vectors from all processes and computes the tentative minimum (mset[]) set as in [21], which in case of Figure 1 is $\{P_0, P_1, P_2\}$]. P_1 sends this tentative minimum set to all processes and takes its own soft checkpoint. A process takes its soft checkpoint if it is a member of the tentative minimum set. When P_0 and P_2 get the mset[], they find themselves in the mset[]; therefore, they take their soft checkpoints. When P_3, P_4 and P_5 get the mset[], they find that they do not belong to mse[], therefore, they do not take their soft checkpoints.

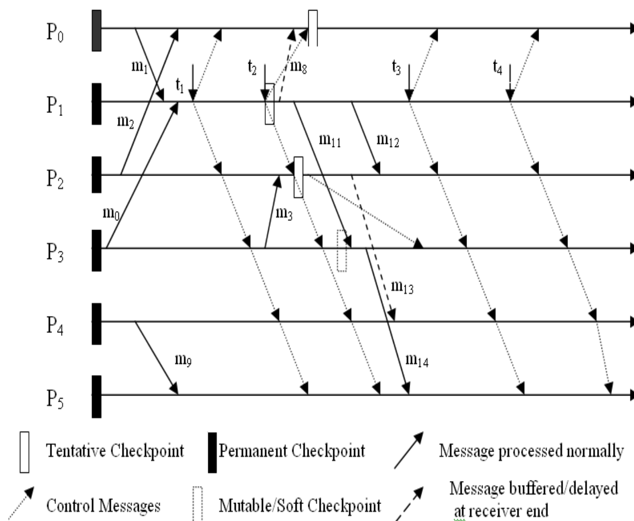

Figure 1 An Example of the proposed Protocol

P_1 sends m_8 after taking its soft checkpoint and P_0 receives m_8 before getting the mset[].When P_1 sends m_8 to P_0, P_1 also piggybacks pr_csn_i and $pr_c_state_i$ along with m. When P_0 receives m_8 it finds that $csn[i] < m.pr_csn_i$ and $m.pr_c_state_i=1$.P_0 concludes that P_1 has taken its checkpoint for some new initiation. P_0 also finds rec_mset = 0 implies P_0 has not received the mset[] for the new initiatition. In this case, P_0 buffers m_8 and processes it only after taking its soft checkpoint. After taking its soft checkpoint, P_1 sends m_{11} to P_3. At the time of receiving m_{11}, P_3 has received the mset[] and it P_3 is not the member of the tentative minimum set (mset[]). P_3 finds that it has sent m_3 to P_2 and P_2 is a member of tentative minimum set (mset[]). Therefore, P_3 concludes that most probably, it will get the checkpoint request in the current initiation; therefore, it takes its mutable checkpoint before processing m_{11}. When P_2 takes its soft checkpoint, it finds that it is dependent upon P_3, due to m_3, and P_3 is not in the tentative minimum set [mset[]]; therefore, P_2 sends checkpoint request to P_3. On receiving the checkpoint request, P_3 converts its mutable checkpoint into soft one. It should be noted that the soft checkpoint and mutable checkpoint are similar. Mutable checkpoint is a forced checkpoint and soft checkpoint is a regular checkpoint taken due to checkpoint request. In order to convert the mutable checkpoint into soft checkpoint, we only need to change the data structure (mss_local_soft[3]=1).

After taking its checkpoint, P_2 sends m_{13} to P_4. P_4 finds that it has not sent any message to a process of tentative minimum set. It takes the bitwise logical AND of $pr_sendv_4[]$ and mset[] and finds the resultant vector to be all zeroes ($pr_sendv_4[]$=[000001]; mset[]=[111000]). P_4 concludes that most probably, it will not get the checkpoint request in the current initiation; therefore, P_4 does not take mutable checkpoint but buffers m_{13}. P_4 processes m_{13} only after getting the tentative checkpoint request. P_5 processes m_{14}, because, it has not sent any message since last permanent checkpoint ($pr_send_5=0$) . After taking its checkpoint, P_1 sends m_{12} to P_2. P_2 processes m_{12}, because, it has already taken its checkpoint in the current initiation. At time t_3, P_1 receives positive responses to soft checkpoint requests from all relevant processes (not shown in the Figure 1) and issues tentative checkpoint request along with the exact minimum set [P_0, P_1, P_2, P_3] to all processes. On receiving tentative checkpoint request, all relevant processes convert their soft checkpoints into tentative ones and inform the initiator. A process, not in the minimum set, discards its mutable checkpoint, if any, or processes the buffered messages, if any. Finally, at time t_4, initiator P_2 issues commit. On receiving commit following actions are taken. A process, in the minimum set, converts its tentative checkpoint into permanent one and discards its earlier permanent checkpoint, if any.

2.5 The Minimum Process Checkpointing Algorithm

2.5.1 Checkpoint Initiation

Each process P_i can initiate the checkpointing procedure. If MH_i wants to initiate checkpointing, it sends the request to its local MSS, called initiator MSS, that initiates and coordinates checkpointing procedure on behalf of MH_i.

ISSN (Online): 1694-0784
ISSN (Print): 1694-0814

The initiator MSS sends a request to all MSSs (MSSs of the mobile system under consideration) to send the d_vect[] vectors of the processes in their cells. All d_vect[] vectors are at MSSs and thus no initial checkpointing messages or responses travel wireless channels. On receiving the d_vect[] request, an MSS records the identity of the initiator MSS (say mss_id= mss_id_{in}) and sends back the d_vect[] of the processes in its cell, and sets timer1. If the initiator MSS receives a request for d_vect[] from some other MSS (say mss_id= mss_id_{in2}) and mss_id_{in} is lower than mss_id_{in2},the, current initiation (having mss_id= mss_id_{in}) is discarded and the new one (having mss_id= mss_id_{in2}) is continued. Similarly, if an MSS receives d_vect[] requests from two MSSs, then it discards the request of the initiator MSS with lower mss_id. Otherwise, on receiving d_vect[] vectors of all processes, the initiator MSS (MSS_{in}) computes mset[], sends soft checkpoint request to all MSSs. On receiving positive responses from all relevant processes, MSS_{in} issues tentative checkpoint request to all processes in the minimum set. If some process, fails to take soft checkpoint in the first phase, MSS_{in} issues abort() request to all MSSs. When, MSS_{in} comes to know that all concerned processes have taken their tentative checkpoint, it issues commit() request to all MSSs.

Concurrent executions of the minimum-process checkpointing algorithms may exhaust the limited battery life and congest the wireless channels. Therefore, the concurrent executions of the proposed protocol are not allowed. The proposed protocol is distributed in nature; because, any process can initiate checkpointing. In case of concurrent initiations, only one is allowed to proceed. Hence, concurrent initiations of the proposed protocol do not cause its concurrent executions.

2.5.2 Reception of a Checkpoint Request

When an MSS (say MSS_p) receives the soft checkpoint request along with the mset[] from MSS_{in} in the first phase, it asks the relevant processes in its cell to take the soft checkpoint and stores them in $mss_soft_req_p$[]. The soft checkpoint of an MH is stored on the disk of the MH. When a process (say P_i) takes its soft checkpoint at MSS_p; it is stored in $mss_loc_soft_p$[]. When it comes to know that all the relevant processes in its cell have taken their soft checkpoints($mss_soft_req_p$[]= $mss_loc_soft_p$[]) or some process failed to take its soft checkpoint, MSS_p sends the response to MSS_{in}. On receiving mset[] or tnew_set[] along with the checkpoint request, an MSS, say MSS_j, updates tminset[] on the basis of mset[] or req.tnew_set[]. It sends the soft checkpoint request to any process P_i if P_i belongs to the mset[] or req.tnew_set[], P_i is running in its cell (mss_local_j[i]=1) and P_i has not been issued soft checkpoint request ($mss_loc_soft_j$[i]=0). If P_i has already taken its mutable checkpoint ($mutable_i$=1), it simply converts its mutable checkpoint into soft checkpoint ($mss_loc_soft_j$[i]=1). On getting the soft checkpoint request, P_i takes its soft checkpoint. P_i processes the buffered messages, if any. For a disconnected MH, that is a member of the minimum set, the MSS that has its disconnected checkpoint, considers its disconnected checkpoint as the required checkpoint.

On receiving mset[], if MSS_j finds a process P_k such that P_k is in blocking state and bitwise logical AND of mset[] and pr_sendv_k[] is not all zeroes, P_k takes the mutable checkpoint, processes the buffered messages, if any.

On issuing checkpoint request to a process, says P_i, MSS_j computes tnew_set[]. It contains the new processes for the minimum set. When a process P_i takes its soft checkpoint, it checks its dependency vector and the tentative minimum set computed so far known at the local MSS. If there is any process P_j such that P_i is dependent upon P_j and P_j is not in the tentative minimum set, then P_j is the new process found for the minimum set. A process P_j is in tnew_set[] only if P_j does not belong to the tminset[], and P_i is directly dependent upon P_j. If tnew_set[] is not empty, MSS_j sends the checkpoint request to processes in tnew_set[]. MSS_j also updates new_set[] and tminset[] on the basis of tnew_set[].

2.5.3Computation Message Received During Checkpointing

Suppose, P_i receives m from P_j, different cases are described as follows:

Case1. P_j has taken some permanent checkpoint after sending m.

If (m.pr_csn_j < csn[j]) then P_i processes m.

Case2. If (P_i is in its blocking state (pr_block_i=1)) then (m is buffered for the blocking period of P_i)

Case3. P_i has not entered the checkpoint state (pr_cstate_i=0).Following sub-cases are possible:

(a) P_j has not taken any checkpoint before seding m.

If (m.$pr_own_csn_j$ = csn[j]) then P_i processes m.

(b) P_j is in checkpointing state at the time of sending m but P_i has not sent

any message since last committed checkpoint.

If ((m.pr_c_state) $\wedge$ (pr_send_i==0)) then (P_i sets $pr_c_state_i$, updates pr_csn_i and processes m).

(c) If ((m.pr_c_state) $\wedge$ (pr_send_i==1) $\wedge$ (!rec_mset==1)) then (P_i sets pr_block_i and buffers m).

ISSN (Online): 1694-0784
ISSN (Print): 1694-0814

If((m.pr_c_state) $\wedge$ (pr_send_i==1) $\wedge$ (rec_mset==1) $\wedge$(Bitwise logical AND of mset[] and $sendv_i$[] is not all zeroes)) then (P_i takes mutable checkpoint before processing m, sets pr_c_state=1, and pr_own_csn=1).

If ((m.pr_c_state) $\wedge$ (pr_send_i==1) $\wedge$ (rec_mset==1) $\wedge$ (Bitwise logical AND of mset[] and pr_sendv_i[] is all zeroes)) then (P_i sets pr_block_i flag and buffers m).

Case4. P_i has entered the checkpoint state ($pr_c_state_i$=1)

P_i processes m.

2.5.4 Termination

When an MSS learns that all of its relevant processes have taken their soft/tentative checkpoints successfully or at least one of its relevant process has failed to take its soft/tentative checkpoint, it sends the response message along with new_set to the initiator MSS. If, after sending the response message, an MSS receives the checkpoint request along with tnew_set, and learns that there is at least one process in the tnew_set running in its cell and it has not taken the soft checkpoint, the MSS requests such process to take checkpoint. It again sends the response message to the initiator MSS. Initiator MSS commits only if every relevant process takes its tentative checkpoint.

When the initiator MSS receives a response from some MSS, it updates its mset[] on the basis of new_set received with the response. Finally, initiator MSS sends commit/abort to all processes. On receiving commit: if a process, say P_i, belongs to the minimum set, it converts its tentative checkpoint into permanent one and discards its earlier permanent checkpoint, if any; otherwise, it processes the buffered messages, if any or discards its mutable checkpoint, if any.

2.6 All Process Checkpointing Algorithm

Our all process checkpointing algorithm is an updating of Elnozahy et al [7]. Initiator MSS sends soft checkpoint request to all MSSs. On receiving the soft checkpoint request, an MSS sends the request to all processes in its cell. A process takes its soft checkpoint if it has not taken the same during the current initiation. A process, after taking its tentative checkpoint or knowing its inability to take the checkpoint, informs its local MSS. When an MSS learns that all of its processes have taken their soft checkpoints, it informs the initiator MSS. When the initiator MSS receives positive response from all MSSs, it issues tentative checkpoint request to all MSSs. If any process fails to take soft checkpoint, initiator MSS issues abort request. Finally, initiator MSS issues commit request.

When a process sends a computation message, it appends its pr_csn with the message. When a process, say P_i, receives a computation message m from some other process, say P_j, P_i takes the soft checkpoint before processing the message if m.pr_csn > csn[j] ; otherwise, it simply processes the message.

3. Conclusions

We propose a hybrid checkpointing algorithm, wherein, an all-process coordinated checkpoint is taken after the execution of minimum-process coordinated checkpointing algorithm for a fixed number of times. In minimum-process checkpointing, we try to reduce the number of useless checkpoints and blocking of processes. We have proposed a probabilistic approach to reduce the number of useless checkpoints. Thus, the proposed protocol is simultaneously able to reduce the useless checkpoints and blocking of processes at very less cost of maintaining and collecting dependencies and piggybacking checkpoint sequence numbers onto normal messages. Concurrent initiations of the proposed protocol do not cause its concurrent executions. We try to reduce the loss of checkpointing effort when any process fails to take its checkpoint in coordination with others.

References

1) Acharya A. and Badrinath B. R., "Checkpointing Distributed Applications on Mobile Computers," Proceedings of the 3rd International Conference on Parallel and Distributed Information Systems, pp. 73-80, September 1994.
2) Cao G. and Singhal M., "On coordinated checkpointing in Distributed Systems", IEEE Transactions on Parallel and Distributed Systems, vol. 9, no.12, pp. 1213-1225, Dec 1998.
3) Cao G. and Singhal M., "On the Impossibility of Min-process Non-blocking Checkpointing and an Efficient Checkpointing Algorithm for Mobile Computing Systems," Proceedings of International Conference on Parallel Processing, pp. 37-44, August 1998.
4) Cao G. and Singhal M., "Mutable Checkpoints: A New Checkpointing Approach for Mobile Computing systems," IEEE Transaction On Parallel and Distributed Systems, vol. 12, no. 2, pp. 157-172, February 2001.
5) Chandy K. M. and Lamport L., "Distributed Snapshots: Determining Global State of Distributed Systems," ACM Transaction on Computing Systems, vol. 3, No. 1, pp. 63-75, February 1985.
6) Elnozahy E.N., Alvisi L., Wang Y.M. and Johnson D.B., "A Survey of Rollback-Recovery Protocols in Message-Passing Systems," ACM Computing Surveys, vol. 34, no. 3, pp. 375-408, 2002.
7) Elnozahy E.N., Johnson D.B. and Zwaenepoel W., "The Performance of Consistent Checkpointing," Proceedings of the 11th Symposium on Reliable Distributed Systems, pp. 39-47, October 1992.
8) Higaki H. and Takizawa M., "Checkpoint-recovery Protocol for Reliable Mobile Systems," Trans. of Information processing Japan, vol. 40, no.1, pp. 236-244, Jan. 1999.

9) J.L. Kim, T. Park, "An efficient Protocol for checkpointing Recovery in Distributed Systems," IEEE Trans. Parallel and Distributed Systems, pp. 955-960, Aug. 1993.
10) Koo R. and Toueg S., "Checkpointing and Roll-Back Recovery for Distributed Systems," IEEE Trans. on Software Engineering, vol. 13, no. 1, pp. 23-31, January 1987.
11) Parveen Kumar, R K Chauhan, "A Coordinated Checkpointing Protocol for Mobile Computing Systems", International Journal of Information and Computing Science, Vol. 9, No. 1, pp. 18-27, 2006.
12) Lalit Kumar, M. Misra, R.C. Joshi, "Low overhead optimal checkpointing for mobile distributed systems" Proceedings. 19th International Conference on IEEE Data Engineering, pp 686 – 88, 2003.
13) Neves N. and Fuchs W. K., "Adaptive Recovery for Mobile Environments," Communications of the ACM, vol. 40, no. 1, pp. 68-74, January 1997.
14) Prakash R. and Singhal M., "Low-Cost Checkpointing and Failure Recovery in Mobile Computing Systems," IEEE Transaction On Parallel and Distributed Systems, vol. 7, no. 10, pp. 1035-1048, October1996.
15) Weigang Ni, Susan V. Vrbsky and Sibabrata Ray, " Pitfalls in nonblocking checkpointing" World Science's journal of Interconnected Networks. Vol. 1 No. 5, pp. 47-78, March 2004.
16) Parveen Kumar, Lalit Kumar, R K Chauhan, "A low overhead Non-intrusive Hybrid Synchronous checkpointing protocol for mobile systems", Journal of Multidisciplinary Engineering Technologies, Vol.1, No. 1, pp 40-50, 2005.
17) Lalit Kumar, Parveen Kumar, R K chauhan "Logging based Coordinated Checkpointing in Mobile Distributed Computing Systems", IETE journal of research, vol. 51, no. 6, 2005.
18) Lamports L., "Time, clocks and ordering of events in distributed systems" Comm. ACM, 21(7), 1978, pp 558-565.
19) Pradhan D.K., Krishana P.P. and Vaidya N.H., "Recovery in Mobile Wireless Environment: Design and Trade-off Analysis," Proceedings 26th International Symposium on Fault-Tolerant Computing, pp. 16-25, 1996.
20) Pushpendra Singh, Gilbert Cabillic, "A Checkpointing Algorithm for Mobile Computing Environment", LNCS, No. 2775, pp 65-74, 2003.
21) Lalit Kumar Awasthi, P.Kumar, "A Synchronous Checkpointing Protocol for Mobile Distributed Systems: Probabilistic Approach" International Journal of Information and Computer Security, Vol.1, No.3 pp 298-314, 2007.
22) Parveen Kumar, "A Low-Cost Hybrid Coordinated Checkpointing Protocol for Mobile Distributed Systems", Mobile Information Systems pp 13-32, Vol. 4, No. 1, 2007.