Three Phase Coordinated Checkpointing Scheme for Mobile Distributed Systems

[1]Monika Nagpal, [2]Parveen Kumar & [3]Surender Jangra

[1]*Research Scholar, Dept. of CSE, Singhania University, Pacheri Bari, Jhunjhunu(Raj.)*
[2]*Professor, Bharat Institute of Engineering & Technology, Meerut (UP)*
[3]*Associate Professor, Deptt. of IT Engg., HCTM, Kaithal(Haryana)*

Abstract: *In this paper, a three phase minimum-process coordinated checkpointing algorithm for non-deterministic mobile distributed systems is proposed, where no useless checkpoints are taken. An effort has been made to minimize the blocking of processes and synchronization message overhead and to capture the partial transitive dependencies during the normal execution by piggybacking dependency vectors onto computation messages. Frequent aborts of checkpointing procedure may happen in mobile systems due to exhausted battery, non-voluntary disconnections of MHs, or poor wireless connectivity. Therefore, in the proposed scheme, all concerned MHs will take ad-hoc checkpoint only and these checkpoint 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 ad-hoc checkpoints only. In this way, we try to minimize the loss of checkpointing effort when any process fails to take its checkpoint in coordination with others.*

Keyword--Checkpointing, Ad-hoc checkpoint, Mobile Host, Mobile Support System

I. INTRODUCTION

Most of the existing coordinated checkpointing algorithms [1-3],[6],[9] rely on the two-phase protocol and save two kinds of checkpoints on the stable storage: tentative and permanent. In the first phase, the initiator process takes a tentative checkpoint and requests all or selective processes to take their tentative checkpoints. If all processes are asked to take their checkpoints, it is called all-process coordinated checkpointing [1],[5],[9]. Alternatively, if selective communicating processes are required to take checkpoints, it is called minimum-process checkpointing [6]. Each process informs the initiator whether it succeeded in taking a tentative checkpoint. After the initiator has received positive acknowledgments from all relevant processes, the algorithm enters the second phase. Alternatively, if a process fails to take its tentative checkpoint in the first phase, the initiator process requests all or concerned processes to abort their tentative checkpoint.

If the initiator learns that all concerned processes have successfully taken their tentative checkpoints, the algorithm enters in the second phase and the initiator asks the relevant processes to make their tentative checkpoints permanent. In order to record a consistent global checkpoint, when a process takes a checkpoint, it asks (by sending checkpoint requests to) all relevant processes to take checkpoints. Therefore, coordinated checkpointing suffers from high overhead associated with the checkpointing process [7], [11], [14-15]. Much of the previous work [4],[7-8],[11-12],[14-15] in coordinated checkpointing has focused on minimizing the number of synchronization messages and the number of checkpoints during the checkpointing process. However, some algorithms (called blocking algorithm) force all relevant processes in the system to block their computations during the checkpointing process [7-8], [11-15]. Checkpointing includes the time to trace the dependency tree and to save the states of processes on the stable storage, which may be long. Moreover, in mobile computing systems, due to the mobility of MHs, a message may be routed several times before reaching its destination. Therefore, blocking algorithms may dramatically reduce the performance of these systems [5]. Recently, non-blocking algorithms [5],[9] have received considerable attention. In these algorithms, processes need not block during the checkpointing by using a checkpointing sequence number to identify orphan messages. Moreover, these algorithms [5], [9] require all processes in the system to take checkpoints during checkpointing, even though many of them may not be necessary.

In this paper, we propose an efficient checkpointing algorithm for mobile computing systems that forces only a minimum number of processes to take checkpoints. An effort has been made to minimize the blocking of processes and synchronization message overhead. We capture the partial transitive dependencies during the normal execution by piggybacking dependency vectors onto computation messages. The Z-dependencies are well taken care of in this protocol. In order to reduce the message overhead, we also avoid collecting dependency vectors of all processes to find the minimum set as in [8], [10-11]. We also try to minimize the loss of checkpointing effort when any process fails to take its checkpoint.

II SYSTEM MODEL

A distributed system consists of a fixed number of processes, P1 , P2 , P3 , …. Pn , which communicate only through messages. Processes cooperate to execute a distributed application and interact with the outside world by receiving and sending input and output messages, respectively. Fig. 1 shows a system consisting of three processes and interactions with the outside world.

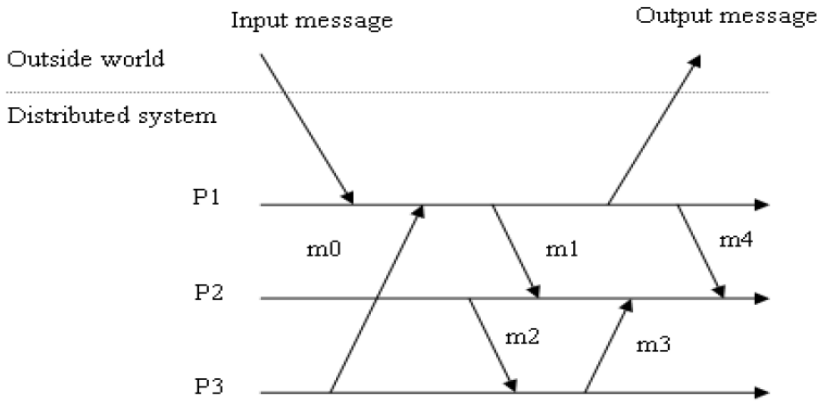

Fig. 1 Model of Distributed System

Rollback-recovery protocols generally make assumptions about the reliability of the inter-process communication. Some protocols assume that the communication subsystem delivers messages reliably, in first-in-first-out (FIFO) order, while other protocols assume that the communication subsystem can lose, duplicate, or reorder messages. The choice between these two assumptions usually affects the complexity of checkpointing and failure recovery. A generic correctness condition for rollback-recovery can be defined as follows : “a system recovers correctly if its internal state is consistent with the observable behavior of the system before the failure.” Rollback-recovery protocols therefore must maintain information about the internal interactions among processes and also the external interactions with the outside world.

In distributed systems, all processes save their local states at certain instants of time. This saved state is known as a local checkpoint. A local checkpoint is a snapshot of the state of the process at a given instance and the event of recording the state of a process is called local checkpointing. The contents of a checkpoint depend upon the application context and the checkpointing method being used. Depending upon the checkpointing method used, a process may keep several local checkpoints or just a single checkpoint at any time. By periodically invoking the checkpointing process, one can save the status of a program at regular intervals. If there is a failure one may restart computation from the last checkpoint thereby avoiding repeating computations from the beginning. The process of resuming computation by rolling back to a saved state is called rollback recovery.

Proposed system model consists of a number of MHs which communicate through mobility support stations (MSSs). Each MSS is a fixed network host which provides wireless communication support for a fixed geographical area, called a cell. MSSs are linked together over the wired data networks. The distributed system consisting of n processes, running on MHs or MSSs. The MHs can communicate with the MSS through wireless channels. We assume that wireless channels and logical channels are all FIFO order. If a MH moves to the cell of another base station, a wireless channel to the old MSS is disconnected and a wireless channel in the new MSS is allocated. However, its checkpoint related information is still with the old MSS. A MH may voluntarily disconnect from mobile computing networks. The MH does not send and receive any message when it is in a disconnected state. We also assume a closed system that consists of nodes, links, and disks. Input is stored on disk before operation begins. Output is stored on disk when the job ends.

There is no common clock, shared memory or central coordinator. Message passing is the only mode of communication between any pair of processes. The messages originated from a source Mh, are received by the local Mobile support stations and then forwarded to the destination MH. Any process can initiate checkpointing. It is assumed that processes may be failed during processing but there is no communication link failure. Messages are exchanged with finite but arbitrary delays. In our algorithm, we consider that the processes which are running in the distributed mobile systems are non-deterministic.

IV. BASIC IDEA

All Communications to and from MH pass through its local MSS. The MSS maintains the dependency information of the MHs which are in its cell. The dependency information is kept in Boolean vector R_i for process P_i. The vector has n bits for n processes. When $R_i[j]$ is set to 1, it represents P_i depends upon P_j. For every P_i, R_i is initialized to 0 except $R_i[i]$, which is initialized to l. When a process P_i running on an MH, say MH_p, receives a message from a process P_j, MH_p's local MSS should set $R_i[j]$ to 1.If P_j has taken its permanent checkpoint after sending m, $R_i[j]$ is not updated.

Suppose there are processes P_i and P_j running on MHs, MH_i and MH_j with dependency vectors R_i and R_j. The dependency vectors of MHs, MH_i and MH_j are maintained by their local MSSs, MSS_i and MSS_j. Process P_i running on MH_i sends message m to process P_j running on MH_j. The message is first sent to MSS_i (local MSS of MH_i). MSS_i maintains the dependency vector R_i of MH_i. MSS_i appends R_i with message m and sends it to MSS_j (local MSS of MH_j). MSS_j maintains the dependency vector R_j of MH_j. MSS_j replaces R_j with bitwise logical OR of dependency vectors R_i and R_j and sends m to P_j.

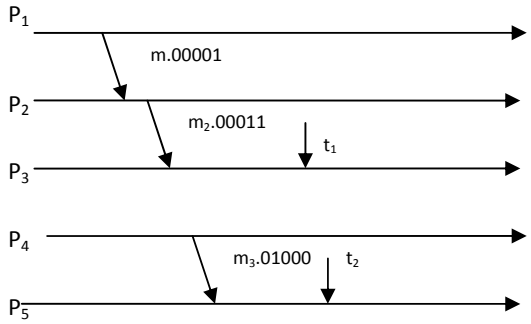

Fig. 2 Maintenance of Dependency Vectors

In Fig. 2, there are five processes P_1, P_2, P_3, P_4, P_5 with dependency vectors R_1, R_2, R_3, R_4, R_5 initialized to 00001, 00010, 00100, 01000, and 10000 respectively. Initially, every process depends upon itself. Now process P_1 sends m to P_2. P_1 appends R_1 with m. P_2 replaces R_2 with the bitwise logical OR of R_1(00001) and R_2(00010), which comes out to be (00011). Now P_2 sends m2 to P_3 and appends R_2 (00011) with m_2. Before receiving m_2, the value of R_3 at P_3 was 00100. After receiving m_2, P_3 replaces R_3 with the bitwise logical OR of R_2 (00011) and R_3 (00100) and R_3 becomes (00111). Now P_4 sends m_3 along with R_4 (01000) to P_5. After receiving m_3, R_5 becomes (11000).In this case, if P_3 starts checkpointing at t_1, it will compute the tentative minimum set equivalent to R_3(00111), which comes out to be $\{P_1, P_2, P_3\}$. In this way, partial transitive dependencies are captured during normal computations.

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 [4, 8, 10, 11, 13, 15]. Furthermore, 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 due to frequent aborts of checkpointing algorithms especially in mobile systems. In mobile distributed systems, there remain certain issues like: abrupt disconnection, exhausted battery power, or failure in wireless bandwidth. So there remains a good probability that some MH may fail to take its checkpoint in coordination with others. Therefore, we propose that, in the first phase, all processes in the minimum set, take ad hoc checkpoint only. Ad hoc checkpoint is stored on the memory of MH only. If some process fails to take its checkpoint in the first phase, then other MHs need to abort their ad hoc checkpoints only. The effort of taking an ad hoc checkpoint is negligible as compared to the tentative one. In other protocols [4],[8],[10], [11], [13],[15], all concerned processes need to abort their tentative checkpoints in this situation. Hence the loss of checkpointing effort in case of an abort of the checkpointing procedure is dramatically low in the proposed scheme as compared to other coordinated checkpointing schemes for mobile distributed systems.

In this second phase, a process converts its ad hoc checkpoint into tentative one. By using this scheme, we try to minimize the loss of checkpointing effort in case of abort of checkpointing algorithm in the first phase. A non-blocking checkpointing algorithm does not require any process to suspend its underlying computation. When processes do not suspend their computation, it is possible for a process to receive a computation message from another process, which is already running in a new checkpointing interval. If this situation is not properly dealt with, it may result in an inconsistency. During the checkpointing procedure, a process P_i may receive m from P_j

such that P_j has taken its checkpoint for the current initiation whereas P_i has not. Suppose, P_i processes m, and it receives checkpoint request later on, and then it takes its checkpoint. In that case, m will become orphan in the recorded global state. We propose that only those messages, which can become orphan, should be buffered at the sender's end. When a process takes its ad hoc checkpoint, it is not allowed to send any message till it receives the tentative checkpoint request. However, in this duration, the process is allowed to perform its normal computations and receive the messages. When a process receives the tentative checkpoint request, it is confirmed that every concerned process has taken its ad hoc checkpoint. Hence, a message generated for sending by a process after getting tentative checkpoint request cannot become orphan. Hence, a process can send the buffered messages after getting the tentative checkpoint request from the initiator.

V. PHASES OF PROPOSED ALGORITHM

A. First phase of the algorithm

When a process, say P_i, running on an MH, say MH_i, initiates a checkpointing, it sends a checkpoint initiation request to its local MSS, which will be the proxy MSS (if the initiator runs on an MSS, then the MSS is the proxy MSS). The proxy MSS maintains the dependency vector of P_i say R_i. On the basis of R_i, the set of dependent processes of P_i is formed, say S_{minset}. The proxy MSS broadcasts ckpt (S_{minset}) to all MSSs. When an MSS receive ckpt (S_{minset}) message, it checks, if any processes in S_{minset} are in its cell. If so, the MSS sends ad hoc checkpoint request message to them. Any process receiving a ad hoc checkpoint request takes a ad hoc checkpoint and sends a response to its local MSS. After an MSS received all response messages from the processes to which it sent ad hoc checkpoint request messages, it sends a response to the proxy MSS. It should be noted that in the first phase, all processes take the ad hoc checkpoints. For a process running on a static host, ad hoc checkpoint is equivalent to tentative checkpoint. But, for an MH, ad hoc checkpoint is different from tentative checkpoint. In order to take a tentative checkpoint, an MH has to record its local state and has to transfer it to its local MSS. But, the ad hoc checkpoint is stored on the local disk of the MH. It should be noted that the effort of taking a ad hoc checkpoint is very small as compared to the tentative one. For a disconnected MH that is a member of minimum set, the MSS that has its disconnected checkpoint, considers its disconnected checkpoint as the required come.

B. Second Phase of the Algorithm

After the proxy MSS has received the response from every MSS, the algorithm enters the second phase. If the proxy MSS learns that all relevant processes have taken their ad hoc checkpoints successfully, it asks them to convert their ad hoc checkpoints into tentative ones and also sends the exact minimum set along with this request. Alternatively, if initiator MSS comes to know that some process has failed to take its checkpoint in the first phase, it issues abort request to all MSS. In this way the MHs need to abort only the ad hoc checkpoints, and not the tentative ones. In this way we try to reduce the loss of checkpointing effort in case of abort of checkpointing algorithm in first phase.

When an MSS receives the tentative checkpoint request, it asks all the process in the minimum set, which are also running in itself, to convert their ad hoc checkpoints into tentative ones. When an MSS learns that all relevant process in its cell have taken their tentative checkpoints successfully, it sends response to proxy MSS. If any MH fails to transfer its checkpoint data to its local MSS, then the failure response is sent to the proxy MSS; which in turn, issues the abort message.

C. Third Phase of the Algorithm

Finally, when the proxy MSS learns that all processes in the minimum set have taken their tentative checkpoints successfully, it issues commit request to all MSSs. When a process in the minimum set gets the commit request, it converts its tentative checkpoint into permanent one and discards its earlier permanent checkpoint, if any.

VI. MASSAGE HANDLING DURING CHECKPOINTING

When a process takes its ad hoc checkpoint, it does not send any massage till it receives the tentative checkpoint request. This time duration of a process is called its uncertainty period. Suppose, Pi sends m to Pj after taking its ad hoc checkpoint and Pj has not taken its ad hoc checkpoint at the time of receiving m. In this case, if Pj takes its ad hoc checkpoint after processing m, then m will become orphan. Therefore, we do not allow Pi to send any massage unless and until every process in the minimum set have taken its ad hoc checkpoint in the first phase. Pi can send massages when it receives the tentative checkpoint request; because, at this moment every concerned process has taken its ad hoc checkpoint and m cannot become orphan. The massages to be sent are buffered at senders end. In this duration, a process is allowed to continue its normal computations and receive massages.

Suppose, Pj gets the ad hoc checkpoint request at MSSp. Now, we find any process Pk such that Pk does not belong to Sminset and Pk belongs to Rj[]. In this case, Pk is also included in the minimum set; and Pj sends ad hoc checkpoint request to Pk. It should be noted that the Sminset, computed on the basis of dependency vector of

initiator process is only a subset of the minimum set. Due to zigzag dependencies, initiator process may be transitively dependent upon some more processes which are not included in the Sminset computed initially.

The proposed Algorithm can be better understood by the example shown in Fig. 3. There are six processes (P0 to P5) denoted by straight lines. Each process is assumed to have initial permanent checkpoints with csn equal to "0". Cix denotes the xth checkpoints of Pi. Initial dependency vectors of P0, P1, P2, P3, P4, P5 are [000001], [000010] [000100], [001000], [010000], and [100000], respectively. P0 sends m2 to P1 along with its dependency vector [000001]. When P1 receives m2, it computes its dependency vector by taking bitwise logical OR of dependency vectors of P0 and P1, which comes out to be [000011]. Similarly, P2 updates its dependency vector on receiving m3 and it comes out to be [000111]. At time t1, P2 initiates checkpointing algorithm with its dependency vector is [000111]. At time t1, P2 finds that it is transitively dependent upon P0 and P1. Therefore, P2 computes the tentative minimum set [Sminset= {P0, P1, P2}]. P2 sends the ad hoc checkpoint request to P1 and P0 and takes its own ad hoc checkpoint C21. For an MH the ad hoc checkpoint is stored on the disk of MH. It should be noted that Sminset is only a subset of the minimum set. When P1 takes its ad hoc checkpoint C11, it finds that it is dependent upon P3 due to m4, but P3 is not a member of Sminset; therefore, P1 sends ad hoc checkpoint request to P3. Consequently, P3 takes its ad hoc checkpoint C31.

After taking its ad hoc checkpoint C21, P2 generates m8 for P3. As P2 has already taken its ad hoc checkpoint for the current initiation and it has not received the tentative checkpoint request from the initiator; therefore P2 buffers m8 on its local disk. We define this duration as the uncertainty period of a process during which a process is not allowed to send any massage. The massages generated for sending are buffered at the local disk of the sender's process. P2 can sends m8 only after getting tentative checkpoint request or abort massages from the initiator process. Similarly, after taking its ad hoc checkpoint P0 buffers m10 for its uncertainty period. It should be noted that P1 receives m10 only after taking its ad hoc checkpoint. Similarly, P3 receives m8 only after taking its ad hoc checkpoint C31.A process is allowed to receive all the massages during its uncertainty period; for example, P3 receives m11. A process is also allowed to perform its normal computations during its uncertainty period.

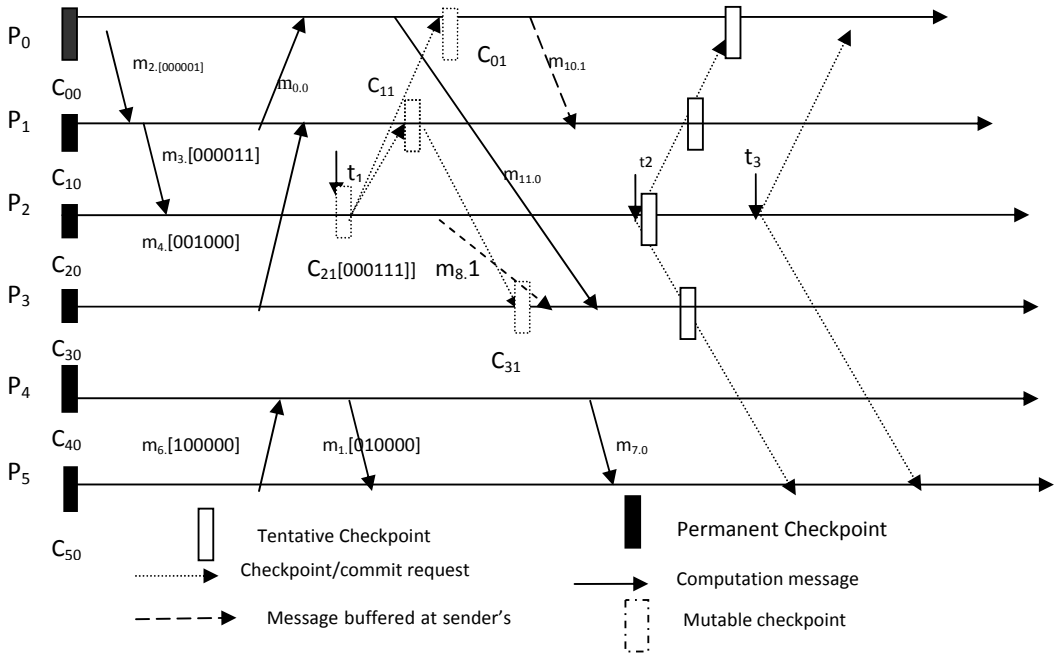

Fig. 3 Working Example of Proposed Scheme

At time t2, P2 receives responses to ad hoc checkpoints requests from all process in the minimum set (not shown in the Fig 3) and finds that they have taken their ad hoc checkpoints successfully, therefore, P2 issues tentative checkpoint request to all processes. On getting tentative checkpoint request, processess in the minimum set [P0, P1, P2, P3] convert their ad hoc checkpoints into tentative ones and send the response to initiator process P2; these process also send the massages, buffered at their local disks, to the destination processes For example, P0 sends m10 to P1 after getting tentative checkpoint request [not shown in the figure]. Similarly, P2 sends m8 to P3 after getting tentative checkpoint request. At time t3, P2 receives responses from the process in minimum set [not shown in the figure] and finds that they have taken their tentative checkpoints successfully, therefore, P2 issues

commit request to all process. A process in the minimum set converts its tentative checkpoint into permanent checkpoint and discards it old permanent checkpoint if any.

VII. CORRECTNESS PROOF BY CONTRADICTION

We can show that global state, collected by the proposed protocol in will be consistent. We can prove the result by contradiction. Suppose there is some orphan message in the recorded global state. We explore different possibilities with the help of Fig 3 Suppose, P_0 sends m_{10} after taking its ad hoc checkpoint and P_1 receives m_{10} before taking its ad hoc checkpoint. This situation is not possible, because, after taking its ad hoc checkpoint P_0 comes into its uncertainty period and it cannot send any message unless and until it receives the tentative checkpoint request. P_2 can issue the tentative checkpoint request only after getting confirmed that every concerned process (including P_1) has taken its ad hoc check point. Hence P_1 cannot receive m10 before taking its ad hoc checkpoint C_{11}. Suppose, P_5 sends m_{13} to P_3 after C_{50} and P_3 gets m_{13} before C_{31} (not show in the Fig. 2). In this case, when P_3 takes its ad hoc checkpoint C_{31}, it will find that P_5 does not belong to S_{minset} and P_3 is dependent upon P_5; therefore, P_3 will send ad hoc checkpoint request to P_5 and send (m_{13}) will also be included in the global state.

VIII. PERFORMANCE ANALYSIS

We use following notations to compare proposed algorithm with other algorithms:

N_{mss}: number of MSSs.
N_{mh}: number of MHs.
C_{pp}: cost of sending a message from one process to another
C_{st}: cost of sending a message between any two MSSs.
C_{wl}: cost of sending a message from an MH to its local MSS (or vice versa).
$C_{broadcast}$: cost of broadcasting a message over static network.
C_{search}: cost incurred to locate an MH and forward a message to its current local MSS, from a source MSS.
T_{st}: average message delay in static network.
T_{wl}: average message delay in the wireless network.
T_{ch}: average delay to save a checkpoint on the stable storage. It also includes the time to transfer the checkpoint from an MH to its local MSS.
N: total number of processes
N_{min}: number of minimum processes required to take checkpoints.
N_{mut}: number of useless mutable checkpoints [4].
T_{search}: average delay incurred to locate an MH and forward a message to its current local MSS.
N_{ucr}: average number of useless checkpoint requests in [4].
N_{dep}: average number of processes on which a process depends.
h_1 : height of the checkpointing tree in Koo-Toueg algorithm [6].
h_2 : height of the checkpointing tree in the proposed algorithm.

A. Message Overhead of the Proposed Algorithm

a) Message overhead in the first phase:

i) Initiator process sends ad hoc checkpoint request to the local MSS and (say MSS_{in}) and gets response from the MSS_{in}: 2 C_{wl}
ii) MSS in broadcasts ad hoc checkpoint request over the static network: $C_{broadcast}$
iii) We suppose that all the process are running on MHs.
iv) All the process in the minimum set get the ad hoc checkpoint request from the local MSS and sends response to the local MSS: $2*N_{min}*C_{wl}$
v) Every MSS sends response to MSS_{in}: $N_{mss}*C_{st}$

b) Message overhead in the second phase

i) MSS_{in} broadcasts tentative checkpoint request over static network:$C_{broadcast}$
ii) Every process in the minimum set receives tentative checkpoint request, and sends response to these requests to local MSS: $2*N_{min}*C_{wl}$
iii) Every MSS sends response to MSS_{in}: $N_{mss}*C_{st}$

c) *Message* overhead in the third phase

i) MSS_{in} broadcasts commit request over static network: $C_{broadcast}$
ii) Total Average message overhead: $2Cwl+3\ C_{broadcast} +4*N_{min}*C_{wl} + 2*N_{mss}*C_{st}$

Proposed algorithm is a three phase algorithm; therefore it suffers from extra message overhead of $C_{broadcast} +4*N_{min}*C_{wl}$. By doing so, we are able to reduce the loss of checkpointing effort in case of abort of the

checkpointing procedure in the first phase. In other algorithms [4, 6, 8], in case of abort in the first phase, all concerned processes are forced to abort their tentative checkpoint whereas in the proposed scheme, all relevant processes abort their ad hoc checkpoints only. The effort of taking an ad hoc checkpoint is negligible as compared to tentative one in the mobile distributed system [4]. Frequent abort of checkpointing algorithms, due to exhausted battery power, abrupt disconnections etc., may significantly increase the checkpointing overhead in two-phase algorithms [4],[8],[10], [11], [13],[15]. We try to minimize the same by designing the three phase algorithm.

In proposed algorithm, only minimum number of processes is required to take their checkpoints. The blocking time of the Koo-Toueg [6] protocol is highest, followed by Cao-Singhal [4] algorithm. We claim that the blocking time in the proposed scheme will be significantly smaller as compared to the KT Algorithm [6]. Because, in algorithm [6], transitive dependencies are collected by direct dependencies. The checkpoint initiator process, say P_{in}, sends the checkpoint request to any process P_i if P_{in} is causally dependent upon P_i. Similarly, P_i sends the checkpoint request to any process P_j if P_i is causally dependent upon P_j. In this way, a checkpointing tree is formed. In the proposed algorithm, transitive dependencies are captured during normal execution as described in Section 3.3. Some zigzag dependencies may not be captured in the proposed scheme during normal execution and they may form low order checkpointing tree in some typical situations. But, in general, the checkpointing tree formed in the proposed scheme will be negligibly small as compared to KT algorithm [6] and hence the blocking time of processes will be small in the proposed scheme as compared to KT algorithm [6]. Furthermore, in the proposed scheme, a process is blocked when it takes its ad hoc checkpoint and it waits for the other concerned process to take their ad hoc checkpoints to come out of blocking state. In KT algorithm [6], a process is blocked when it takes its tentative checkpoint and it waits for the other concerned process to take their tentative checkpoints to come out of blocking state. In mobile distributed systems, the time to take an ad-hoc checkpoint may be negligibly small as compared to tentative checkpoint.

Table 1 A Comparison of System Performance

	Cao-Singhal [8]	**Cao-Singhal [4]**	**Koo-Toeg Algorithm [6]**	**Elnozahy et al [5]**	**Proposed Algorithm**
Avg. bloc king Time	$2T_{st}$	0	h_1*T_{ch}	0	h_2*T_{ch}
Average No. of checkpoints	N_{min}	$N_{min}+N_{mut}$	N_{min}	N	N_{min}
Average Message Overhead	$3C_{broadcast}+2C_{wl}+2N_{mss}*C_{st}+3N_{mh}*C_{wl}$	$2*N_{min}*C_{pp}+C_{broadcast}+N_{ucr}*C_{pp}$	$3*N_{min}*C_{pp}*N_{dep}$	$2*C_{broadcast}+N*C_{pp}$	$2Cwl+3+C_{broadcast}+4*N_{min}*C_{wl}+2*N_{mss}*C_{st}$

Hence, in the proposed scheme, the blocking period of a process will be significantly small as compared to the KT algorithm [6]. Proposed blocking period is larger than CS algorithm [8], but it suffers from extra message overhead of collecting dependency vectors from all processes and moreover, it forces all the processes to block for a short duration. In proposed scheme, a process is blocked only if it is a member of the minimum set. Furthermore, a process is allowed to perform its normal computations and receive messages during its blocking period.

In the algorithms proposed in [4],[10],[13], no blocking of processes takes place, but some useless checkpoints are taken, which are discarded on commit. In Elnozahy et al [5] algorithm, all processes take checkpoints. In the protocols [6], [8], and in the proposed one, only minimum numbers of processes record their checkpoints. In algorithm [4], concurrent executions of the algorithm are allowed, but it may lead to inconsistencies in doing so [16]. We avoid the concurrent executions of the proposed algorithm..

IX. CONCLUSION

Proposed minimum-process synchronous checkpointing algorithm for mobile system try to minimize the blocking of processes during checkpointing. The blocking time of a process is bare minimum. During blocking period, processes can do their normal computations, send messages and can process selective messages. The number of processes that take checkpoints is minimized to avoid awakening of MHs in doze mode of operation and thrashing of MHs with checkpointing activity. It also saves limited battery life of MHs and low bandwidth of wireless channels which reduce the loss of checkpointing effort when any process fails to take its checkpoint in

coordination with others try to minimize the synchronization messages during checkpointing. In the proposed scheme, no synchronization messages are sent in order to enter the second or third phase of the algorithm.

REFERENCES

[1] Chandy, K. M., and L. Lamport," Distributed Snapshots Determining Global State of Distributed System,"ACM Transaction on Computer System.Vol. 3, no. 1, Feb. 1985, pp. 63-75.

[2] Lamport,L."Time, Clocks, and the Ordering of Events in Distributed System,"Communications of the ACM, vol. 21, no.7,July 1978, pp. 558-565. Proceedings of the 9th International Conference on Distributed Computing System, 1989.

[3] Le Lann, G., " Distributed System-Towards a Formal Approach," Information Processing Letter, North-Holland, Vol. 77, 1977, pp. 155-160.

[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] Elnozahy E.N., Johnson D.B. and Zwaenepoel W., "The Performance of Consistent Checkpointing," Proceedings of the 11th Symposium on ReliableDistributed Systems, pp. 39-47, October 1992.

[6] 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.

[7] Lalit Kumar Awasthi, Kumar p. 2007 A Synchoronous Checkpointing Protocol For Mobile Distributed Systems :Probabilistic Approach. Int J. Information and Computer Security, Vol.1, No.3 .pp 298-314

[8] G. Cao and M. Singhal. "On impossibility of Min-Process and Non-Blocking Checkpointing and An Efficient Checkpointing algorithm for mobile computing Systems". OSU Technical Report #OSU-CISRC-9/97-TR44, 1997.

[9] Silva L, Silva J 1992 Global checkpointing for distributed programs. Proc. IEEE 11th Symp. On Reliable Distributed Syst. pp 155-162.

[10] P. Kumar, L. Kumar and R.K. Chauhan, "A Non-Intrusive minimum process synchronous checkpointing protocol for mobile distributed systems", in proceeding of IEEE ICPWC-2005,2005

[11] Parveen Kumar, "A Low-Cost Hybrid Coordinated Checkpointing Protocol for mobile distributed systems", Mobile Information Systems. pp 13-32, Vol. 4, No. 1, 2007.

[12] 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.

[13] L. Kumar, M. Misra, R.C. Joshi, "Low overhead optimal checkpointing for mobile distributed systems" Proceedings. 19th IEEE International Conference on Data Engineering, pp 686 – 88, 2003.

[14] Parveen Kumar, Lalit Kumar, R K Chauhan, "A Non-intrusive Hybrid Synchronous Checkpointing Protocol for Mobile Systems", IETE Journal of Research, Vol. 52 No. 2&3, 2006.

[15] Sunil Kumar, R K Chauhan, Parveen Kumar, "A Minimum-process Coordinated Checkpointing Protocol for Mobile Computing Systems", *International Journal of Foundations of Computer science*,Vol 19, No. 4, pp 1015-1038 (2008).

[16] Ni, W., S. Vrbsky and S. Ray, "Pitfalls in Distributed Nonblocking Checkpointing", Journal of Interconnection Networks, Vol. 1 No. 5, pp. 47-78, March 2004.