REVIEW of SOME MINIMUM-PROCESS SYNCHRONOUS CHECKPOINTING SCHEMES for MOBILE DISTRIBUTED SYSTEMS

Preeti Gupta[#1], Parveen Kumar [#2], Anil Kumar Solanki[#3]

[1]*Singhania University,*
Pacheri Bari, (Rajasthan) India
[*2]*Meerut Institute of Engineering & Technology,*
Meerut (INDIA)-250005
Email:pk223475@yahoo.com
[*3]*Meerut Institute of Engineering & Technology,*
Meerut (INDIA)-250005

***Abstract*— The term Distributed Systems is used to describe a system with the following characteristics: i) it consists of several computers that do not share memory or a clock, ii) the computers communicate with each other by exchanging messages over a communication network, iii) each computer has its own memory and runs its own operating system. A mobile computing system is a distributed system where some of processes are running on mobile hosts (MHs), whose location in the network changes with time. The number of processes that take checkpoints is minimized to 1) avoid awakening of MHs in doze mode of operation, 2) minimize thrashing of MHs with checkpointing activity, 3) save limited battery life of MHs and low bandwidth of wireless channels. In minimum-process checkpointing protocols, some useless checkpoints are taken or blocking of processes takes place. To take a checkpoint, an MH has to transfer a large amount of checkpoint data to its local MSS over the wireless network. Since the wireless network has low bandwidth and MHs have low computation power, all-process checkpointing will waste the scarce resources of the mobile system on every checkpoint. Minimum-process coordinated checkpointing is a preferred approach for mobile distributed systems. In this paper, we discuss various existing minimum-process checkpointing protocols for mobile distributed systems.**

***Keywords*- Checkpointing algorithms; parallel & distributed computing; rollback recovery; fault-tolerant systems**

I. INTRODUCTION

Parallel computing with clusters of workstations is being used extensively as they are cost-effective and scalable, and are able to meet the demands of high performance computing. Increase in the number of components in such systems increases the failure probability. It is, thus, necessary to examine both hardware and software solutions to ensure fault tolerance of such parallel computers. To provide fault tolerance, it is essential to understand the nature of the faults that occur in these systems. There are mainly two kinds of faults: permanent and transient. Permanent faults are caused by permanent damage to one or more components and transient faults are caused by changes in environmental conditions. Permanent faults can be rectified by repair or replacement of components. Transient faults remain for a short duration of time and are difficult to detect and deal with. Hence it is necessary to provide fault tolerance particularly for transient failures in parallel computers. Fault-tolerant techniques enable a system to perform tasks in the presence of faults. It is easier and more cost effective to provide software fault tolerance solutions than hardware solutions to cope with transient failures [1, 2].

Local checkpoint is the saved state of a process at a processor at a given instance. Global checkpoint is a collection of local checkpoints, one from each process. 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. A transit message is a message whose send event has been recorded by the sending process but whose receive event has not been recorded by the receiving process [1, 7].

The problem of taking a checkpoint in a message passing distributed system is quite complex because any arbitrary set of checkpoints cannot be used for recovery [9]. This is due to the fact that the set of checkpoints used for recovery must form a consistent global state.

Checkpointing is classified into following categories:

- Asynchronous/Uncoordinated Checkpointing
- Synchronous/Coordinated Checkpointing
- Quasi-Synchronous or Communication-induced Checkpointing

- Message Logging based Checkpointing

The problem of taking a checkpoint in a message passing distributed system is quite complex because any arbitrary set of checkpoints cannot be used for recovery. This is due to the fact that the set of checkpoints used for recovery must form a consistent global state.

In coordinated or synchronous checkpointing, processes coordinate their local checkpointing actions such that the set of all recent checkpoints in the system is guaranteed to be consistent [add reference list……]. In case of a fault, every process restarts from its most recent permanent/committed checkpoint. Hence, this approach simplifies recovery and it does not suffer from domino-effect. Furthermore, coordinated checkpointing requires each process to maintain only one permanent checkpoint on stable storage, reducing storage overhead and eliminating the need for garbage collection. Its main disadvantage is the large latency involved in output commits [15].

The coordinated checkpointing protocols can be classified into two types: blocking and non-blocking. In blocking algorithms, as mentioned above, some blocking of processes takes place during checkpointing [4]. In non-blocking algorithms, no blocking of processes is required for checkpointing [5].

In a centralized algorithm like Chandy-lamport [7], there is one node which always initiates the checkpoints and coordinates the participating nodes. The disadvantage of a centralized algorithm is that all nodes have to initiate checkpoints whenever the centralized node decides to checkpoint. Nodes can be given autonomy in initiating checkpoints by allowing any node in the system to initiate checkpoints.

The existence of mobile nodes in a distributed system introduces new issues that need proper handling while designing a checkpointing algorithm for such systems. These issues are mobility, disconnections, finite power source, vulnerable to physical damage, lack of stable storage etc. [2]. The location of an MH within the network, as represented by its current local MSS, changes with time. Checkpointing schemes that send control messages to MHs, will need to first locate the MH within the network, and thereby incur a search overhead [2]. Due to vulnerability of mobile computers to catastrophic failures, disk storage of an MH is not acceptably stable for storing message logs or local checkpoints. Checkpointing schemes must therefore, rely on an alternative stable repository for an MH's local checkpoint [2]. Disconnections of one or more MHs should not prevent recording the global state of an application executing on MHs. It should be noted that disconnection of an MH is a voluntary operation, and frequent disconnections of MHs is an expected feature of the mobile computing environments [2]. The battery at the MH has limited life. To save energy, the MH can power down individual components during periods of low activity [2]. This strategy is referred to as the doze mode operation. An MH in doze mode is awakened on receiving a message. Therefore, energy conservation and low bandwidth constraints require the checkpointing algorithms to minimize the number of synchronization messages and the number of checkpoints. Prakash & Singhal [16] 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 processes takes place [4, 10] or some useless checkpoints are taken [[5, 11].

II. SOME MINIMUM-PROCESS COORDINATED CHECKPOINTING PROTOCOLS FOR MOBILE DISTRIBUTED SYSTEMS

A. Cao and Singhal non-intrusive Algorithm [5]

Cao and Singhal achieved non-intrusiveness in the minimum-process algorithm by introducing the concept of mutable checkpoints. In their algorithm, initiator, say Pin, sends the checkpoint request to any process, say Pj, only if Pin receives m from Pj in the current CI. Pj takes its tentative checkpoint if Pj has sent m to Pin in the current CI; otherwise, Pj concludes that the checkpoint request is a useless one. Similarly, when Pj takes its tentative checkpoint, it propagates the checkpoint request to other processes. This process is continued till the checkpoint request reaches all the processes on which the initiator transitively depends and a checkpointing tree is formed. During checkpointing, if Pi receives m from Pj such that Pj has taken some checkpoint in the current initiation before sending m, Pi may be forced to take a checkpoint, called mutable checkpoint. If Pi is not in the minimum set, its mutable checkpoint is useless and is discarded on commit. The huge data structure MR[] is also attached with the checkpoint requests to reduce the number of useless checkpoint requests. The response from each process is sent directly to initiator.

B. A Probabilistic Algorithm by Kumar & Kuma [14]

They have proposed a coordinated checkpointing protocol for mobile distributed systems, where only interacting processes are required to checkpoint. They are able to maintain exact dependencies among processes and make an approximate set of interacting processes at the beginning. In this way, the time to collect coordinated checkpoint is reduced. It also reduces number of useless checkpoints and blocking of processes. They have tried to minimize the blocking of processes by buffering some messages at the receiver end for a short duration. During blocking period, processes are allowed to do their normal computations and send messages. They 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.

The basic idea can be understood by the following description. Suppose, during the execution of the checkpointing algorithm, *Pi* takes its checkpoint and sends *m* to *Pj*. *Pj* 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. If *Pj* takes its checkpoint after processing *m*, *m* will become orphan. In order to avoid such orphan messages, they propose the following technique. If *Pj* has sent at least one message to a process, say *Pk* and *Pk* is in the tentative minimum set, there is a good probability that *Pj* will get the checkpoint request. Therefore, *Pj* takes its induced checkpoint before processing *m*. An induced checkpoint is similar to the mutable checkpoint. In this case, most probably, *Pj* will get the checkpoint request and its induced checkpoint will be converted into permanent one. There is a less probability that *Pj* will not get the checkpoint request and its induced checkpoint will be discarded. Alternatively, if there is not a good probability that *Pj* will get the checkpoint request, *Pj* buffers *m* till it takes its checkpoint or receives the commit message. They have tried to minimise the number of useless checkpoints and blocking of the process by using the probabilistic approach and buffering selective messages at the receiver end. Exact dependencies among processes are maintained. It abolishes the useless checkpoint requests and reduces the number of duplicate checkpoint requests.

C. HYbrid of Minimum Process & All Process checkpointing Scheme by Kumar [13]

In minimum-process checkpointing, some processes, having low communication activity, may not be included in the minimum set for several checkpoint initiations and thus may not advance their recovery line for a long time. In the case of a recovery after a fault, this may lead to their rollback to far earlier checkpointed state and the loss of computation at such processes may be exceedingly high. In all-process checkpointing, recovery line is advanced for each process after every global checkpoint but the checkpointing overhead may be exceedingly high, especially in mobile environments due to frequent checkpoints. MHs utilize the stable storage at the MSSs to store checkpoints of the MHs Thus, to balance the checkpointing overhead and the loss of computation on recovery, he designed a hybrid checkpointing algorithm for mobile distributed systems, where an all-process checkpoint is taken after certain number of minimum-process checkpoints. The number of times, the minimum-process checkpointing algorithm is executed, depends on the particular application and environment and can be fine-tuned.

In coordinated checkpointing, an ever-increasing integer csn is generally piggybacked onto normal messages [5, 15]. He proposed a strategy to optimize the size of the csn. In order to address different checkpointing intervals, he has replaced integer csn with k-bit CI. Integer csn is monotonically increasing, each time a process takes its checkpoint, it increments its csn by 1. k-bit CI is used to serve the purpose of integer csn. The value of k can be fine-tuned. In the present study, we assume that all-process coordinated checkpoint is taken after the execution of minimum-process algorithm for seven times which requires only three-bit CI. In this case, any delay of a message that extends to more than seven CIs may cause a false checkpoint, i.e., it may trigger a checkpoint even if an initiator does not trigger checkpointing activity. Thus, in this algorithm, such delay needs to be avoided. During the period, when a process sends its dependency set to the initiator and receives the minimum set, may receive some messages, which may alter its dependency set, and may add new members to the already computed minimum set. In order to keep the computed minimum set intact and to avoid useless checkpoints as in, he proposed to block the processes for this period. He has classified the messages, received during the blocking period, into two types: (i) messages that alter the dependency set of the receiver process (ii) messages that do not alter the dependency set of the receiver process. The former messages need to be delayed at the receiver side. The messages of the later type can be processed normally. All processes can perform their normal computations and send messages during their blocking period. When a process buffers a message of former type, it does not process any message till it receives the minimum set so as to keep the proper sequence of messages received. When a process gets the minimum set, it takes the checkpoint, if it is in the minimum set. After this, it receives the buffered messages, if any. By doing so, blocking of processes is reduced as compared to [4].

D. KUMAR & KHUNTETA ALGORITHM [17]

In this paper, authors design a minimum process algorithm for Mobile Distributed systems, where no useless checkpoints are taken and an effort has been made to optimize the blocking of processes. They propose to delay the processing of selective messages at the receiver end only during the checkpointing period. A Process is allowed to perform its normal computations and send messages during its blocking period. In this way, they try to keep blocking of processes to bare minimum. They captured the transitive dependencies during the normal execution by piggybacking dependency vectors onto computational messages. In this way, they try to reduce the Checkpointing time by avoiding formation of Checkpointing tree. The Z-dependencies are well taken care of. The proposed scheme forces zero useless checkpoints at the cost of very small blocking.

The basic idea of this scheme as described as follows. During the execution of checkpointing algorithm, a process P_i may receive m from P_j such that P_j has taken its tentative checkpoint for the current initiation whereas P_i has not taken. If P_i processes m and it receives checkpoint request later on and takes its checkpoint, then m will become orphan in the recorded global state. We propose that such messages should be buffered at the receiver end. In the present discussion, P_i processes m only after taking its tentative checkpoint if it is a member of the minimum set; otherwise, P_i processes m after

getting the exact minimum set and knowing that it is not a member of the minimum set.

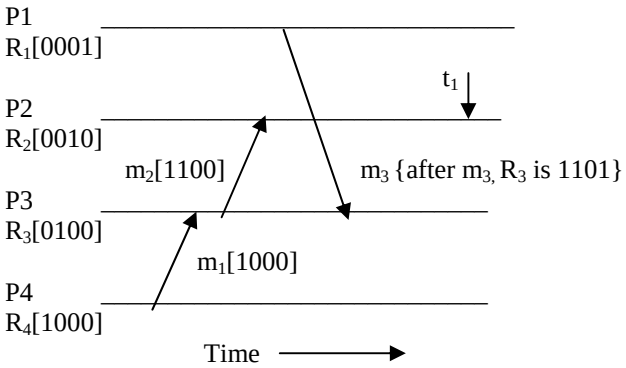

Figure. 1 Basic Idea

In the figure 1 P_4 sends m_1 to P_3 along with its own dependency vector $R_4[1000]$. When P_3 receives m_1 it updates its own dependency vector by taking logical OR of R_4 & $R_3[0100]$, which comes out to be 1100. When P_3 send m_2 to P_2, it appends $R_3[1100]$ along with m_2. When P_2 receive m_2, it updates its own dependency vector R_2 by taking logical OR of R_2 and R_3, which comes out to be [1110]. In this way, partial transitive dependencies are captured during normal computation. It should be noted that all the transitive dependencies are not captured during normal computation. At time t1, the dependency vector of P_2 shows that P_2 is not transitively dependent upon P_1, due to m_3 and m_2.

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 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.

E. Kumar & Garg Algorithm [18]

They 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, they try to reduce the number of useless checkpoints and blocking of processes. They 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. They try to reduce the loss of checkpointing effort when any process fails to take its checkpoint in coordination with others.

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, they propose that in the first phase, all concerned MHs will take soft checkpoint only. Soft checkpoint is similar to mutable checkpoint [5], 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.

F. Hybrid of Synchronous & Asynchronous Checkpointing in Mobile Systems [19]

It is difficult for multiple MHs to synchronously take checkpoints since the wireless channels are less reliable and may disconnect even during checkpointing. Mobile hosts are prone to frequent failures and it may lead to frequent rollbacks. Blocking of processes during checkpointing may degrade the system performance.

In this scheme, authors propose a hybrid checkpointing protocol that is non-blocking. MHs take checkpoints independently. All to and fro messages of an MH pass through its current local MSS. Therefore, an MSS logs the messages of the MHs in its cell. If an MH fails to take its checkpoint and transfer it to the current MSS, it can try later. MSSs take checkpoints synchronously. A process on an MH can recover independently. When a process on an MH crashes, a new process is created using checkpoint of the crashed MH, and then the logged messages are replayed in the order they were originally received. When a process on an MSS fails, all processes rollback to recent synchronous checkpoint. An MH uses its recent committed checkpoint and message logs to reach to a state consistent with the synchronous checkpoint. The algorithm does not awaken an MH in doze mode operation. An MH can remain disconnected for an arbitrary period of time without affecting checkpointing activity.

In order to realize non-blocking during coordinated checkpointing, an integer csn is generally piggybacked onto normal messages [5], [15]. They propose a strategy to optimize the size of the csn. In order to address different checkpointing

intervals (CIs), they have replaced integer csn used in [5], [15], with k-bits CI. The number of bits used in CI can be fine tuned in accordance with the system under consideration. If we use k-bits CI, we will be able to distinguish only 2^k different checkpointing intervals and it will be implicitly assumed that no message is delivered after 2^k-1 checkpointing intervals. . Higaki & Takizawa [24] originally proposed hybrid checkpointing protocols where MHs checkpoint independently and MSSs checkpoint synchronously.

G. Some More Checkpointing Schemes For Mobile Distributed Systems

Biswas & Neogy [20] proposed a checkpointing and failure recovery algorithm where mobile hosts save checkpoints based on mobility and movement patterns. Mobile hosts save checkpoints when number of hand-offs exceed a predefined handoff threshold value. Gao et al [21] developed an index-based algorithm which uses time-coordination for consistently checkpointing in mobile computing environments. In time-based checkpointing protocols, there is no need to send extra coordination messages. However, they have to deal with the synchronization of timers. This class of protocols suits to the applications where processes have high message sending rate. Rao and Naidu [22] proposed a new coordinated checkpointing protocol combined with selective sender-based message logging. The protocol is free from the problem of lost messages. The term 'selective' implies that messages are logged only within a specified interval known as active interval, thereby reducing message logging overhead. All processes take checkpoints at the end of their respective active intervals forming a consistent global checkpoint. Singh & Cabillic [23] proposed a minimum-process non-intrusive coordinated checkpointing protocol for deterministic mobile systems, where anti-messages of selective messages are logged during checkpointing.

III. CONCLUSIONS

Minimum-process coordinated checkpointing is a suitable approach to introduce fault tolerance in mobile distributed systems transparently. This approach is domino-free, requires at most two checkpoints of a process on stable storage, and forces only a minimum number of processes to checkpoint. It may require blocking of processes, extra synchronization messages, piggybacking of some information along with computation messages, or taking some useless checkpoints. In this paper we have given introductory concepts related to checkpointing in mobile distributed systems. We also gave a review of various minimum-process checkpointing schemes especially designed for mobile distributed systems.

REFERENCES

[1] Singhal, M, N G Shivratri, "Advanced concepts in Operating Systems, Tata Mc Graw Hill, 1994.

[2] Acharya A., "Structuring Distributed Algorithms and Services for networks with Mobile Hosts", Ph.D. Thesis, Rutgers University, 1995.

[3] 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.

[4] 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.

[5] 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.

[6] Cao G. and Singhal M., "Checkpointing with Mutable Checkpoints", *Theoretical Computer Science*, 290(2003), pp. 1127-1148.

[7] 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.

[8] 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.

[9] Kalaiselvi, S., Rajaraman, V., "A Survey of Checkpointing Algorithms for Parallel and Distributed Systems", *Sadhna*, Vol. 25, Part 5, October 2000, pp. 489-510.

[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, Lalit Kumar, R K Chauhan, V K Gupta "A Non-Intrusive Minimum Process Synchronous Checkpointing Protocol for Mobile Distributed Systems" *Proceedings of IEEE ICPWC-2005*, January 2005.

[12] Pushpendra Singh, Gilbert Cabillic, "A Checkpointing Algorithm for Mobile Computing Environment", *LNCS, No. 2775*, pp 65-74, 2003.

[13] Parveen Kumar, "A Low-Cost Hybrid Coordinated Checkpointing Protocol for Mobile Distributed Systems", *Mobile Information Systems* [An International Journal from IOS Press, Netherlands] pp 13-32, Vol. 4, No. 1, 2007.

[14]. Lalit Kumar, Parveen Kumar "A Synchronous Checkpointing Protocol for Mobile Distributed Systems: A Probabilistic Approach", *International Journal of Information and Computer Security* [An International Journal from Inderscience Publishers, USA], pp 298-314, Vol. 3 No. 1, 2007.

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

[16] R. Prakash and M. Singhal. "Low-Cost Checkpointing and Failure Recovery in Mobile Computing Systems". *IEEE Trans. on Parallel and Distributed System*, pages 1035-1048,Oct. 1996.

[17] Kumar, P., & Khunteta, A. ,"A Minimum-Process Coordinated Checkpointing Protocol For Mobile Distributed System", *International Journal of Computer Science issues, Vol. 7, Issue 3.,2010*

[18] Kumar, P., & Garg, R. ," Soft-Checkpointing Based Coordinated Checkpointing Protocol for Mobile Distributed Systems", International Journal of Computer Science issues, Vol. 7, Issue 3.,2010

[19] Lalit Kumar, Parveen Kumar, R K Chauhan, "Logging based Coordinated Checkpointing in Mobile Distributed Computing Systems", *IETE Journal of Research*, vol. 51, no. 6, pp. 485-490, 2005.

[20] Biswas, S., & Neogy, S.,"A Mobility-Based Checkpointing Protocol for Mobile Computing System", *International Journal of Computer Science & Information Technology, Vol.2, No.1, pp135-151, 2010*

[21] Gao, Y., Deng, C., & Che, Y. ," An Adaptive Index-Based Algorithm Using Time-Coordination in Mobile Computing", *International Symposiums on Information Processing, pp.578-585*.,2008.

[22] Rao, S., & Naidu, M.M.,"A New, Efficient Coordinated Checkpointing Protocol Combined with Selective Sender-Based Message Logging", *International Conference on Computer Systems and Applications.* IEEE/ACS, 2008.

[23] Singh, P., & Cabillic, G. (2003). A Checkpointing Algorithm for Mobile Computing Environment. *LNCS, No. 2775*, pp 65-74.

[24]. Higaki, H., & Takizawa, M. , Checkpoint-recovery Protocol for Reliable Mobile Systems. *Trans. of Information processing Japan, vol. 40, no.1, pp. 236-244*., 1999