International Journal of Computer Sciences and Engineering Systems
Vol. 12 No.2 (December, 2018)

An Efficient Snapshot Collection Protocol for Deterministic Mobile Distributed Systems

S. K Gupta[1], R. K. Chauhan[2] & Parveen Kumar[3]

[1]BCET Gurdaspur (Punjab), Sunil Kumar Gupta, BCET, Gurdaspur-143521 Punjab, INDIA
E-mail: skgbcet1965@rediffmail.com
[2]Kurukshetra University Kurukshetra (Haryana)
[3]APIIT, Panipat (Haryana)

Manuscript received October 25, 2017, Manuscript revised December 15, 2017

Abstract: *In this paper, we propose a minimum-process coordinated checkpointing algorithm for deterministic mobile distributed systems, where no useless checkpoints are taken, no blocking of processes takes place, and anti-messages of very few messages are logged during checkpointing. We also address the related issues like: failures during checkpointing, disconnections, concurrent initiations of the algorithm and maintaining exact dependencies among processes.*

Keywords: Distributed Systems, Fault Tolerance, Anti-message, Coordinated Checkpointing, Mobile Systems

1. INTRODUCTION

Checkpoint is defined as a designated place in a program at which normal process is interrupted specifically to preserve the status information necessary to allow resumption of processing at a later time. A checkpoint is a local state of a process saved on stable storage. 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 checkpoints thereby avoiding repeating computation from the beginning. The process of resuming computation by rolling back to a saved state is called rollback recovery. 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. To recover from a failure, the system restarts its execution from a previous consistent global state saved on the stable storage during fault-free execution. In distributed systems, checkpointing can be independent, coordinated[3,8,11] or quasi-synchronous [2,9]. Message Logging is also used for fault tolerance in distributed systems [14].

Under the asynchronous approach, checkpoints at each process are taken independently without any synchronization among the processes. Because of absence of synchronization, there is no guarantee that a set of local checkpoints taken will be a consistent set of checkpoints. It may require cascaded rollbacks that may lead to the initial state due to domino-effect [7]. In coordinated or synchronous checkpointing, processes take checkpoints in such a manner that the resulting global state is consistent. Mostly it follows two-phase commit structure [3, 8, 11, 22]. 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. Communication-induced checkpointing avoids the domino-effect without requiring all checkpoints to be coordinated [2, 7, 9].

In the mobile distributed system, some of the processes are running on mobile hosts (MHs). An MH communicates with other nodes of the system via a special node called mobile support station (MSS) [1]. A cell is a geographical area around an MSS in which it can support an MH. An MH can change its geographical position freely from one cell to another or even to an area covered by no cell. An MSS can have both wired and wireless links and acts as an interface between the static network and a part of the mobile network. Static network connects all MSSs. A static node that has no support to MH can be considered as an MSS with no MH.

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, disconnection, finite power source, vulnerable to physical damage, lack of stable storage etc. These issues make traditional checkpointing techniques unsuitable to checkpoint mobile distributed systems [1, 5, 15]. A good checkpointing protocol for mobile distributed systems should have low overheads on MHs and wireless

channels and should avoid awakening of MHs in doze mode operation. The disconnection of MHs should not lead to infinite wait state. The algorithm should be non-intrusive and should force minimum number of processes to take their local checkpoints [15]. In minimum-process coordinated checkpointing algorithms, some blocking of the processes takes place [4, 11], or some useless checkpoints are taken [5, 13, 19].

Cao and Singhal[5] achieved non-intrusiveness in the minimum-process algorithm by introducing the concept of mutable checkpoints. The number of useless checkpoints in [5] may be exceedingly high in some situations [19]. Kumar *et al.* [19] and Kumar *et al.* [13] reduced the height of the checkpointing tree and the number of useless checkpoints by keeping non-intrusiveness intact, at the extra cost of maintaining and collecting dependency vectors, computing the minimum set and broadcasting the same on the static network along with the checkpoint request.

Koo and Toeg [11], and Cao and Singhal [4] proposed minimum-process blocking coordinated checkpointing algorithms. Neves *et al.* [12] gave a loosely synchronized coordinated protocol that removes the overhead of synchronization. Higaki and Takizawa [10] proposed a hybrid checkpointing protocol where the mobile stations take checkpoints asynchronously and fixed ones synchronously. Kumar and Kumar [29] proposed a minimum-process coordinated checkpointing algorithm where the number of useless checkpoints and blocking are reduced by using a probabilistic approach. A process takes its mutable checkpoint only if the probability that it will get the checkpoint request in the current initiation is high. To balance the checkpointing overhead and the loss of computation on recovery, P. Kumar [27] and Kumar *et al.* [26], proposed a hybrid-coordinated checkpointing protocol for mobile distributed systems, where an all-process checkpoint is taken after executing minimum-process checkpointing algorithm for a certain number of times.

In deterministic systems, if two processes start in the same state, and both receive the identical sequence of inputs, they will produce the identical sequence outputs and will finish in the same state. The state of a process is thus completely determined by its starting state and by sequence of messages it has received [23, 24, 25].

David R. Jefferson [23] introduced the concept of anti-message. Anti-message is exactly like an original message in format and content except in one field, its sign. Two messages that are identical except for opposite signs are called anti-messages of one another. All messages sent explicitly by user programs have a positive (+) sign; and their anti-messages had a negative sign (-). Whenever a message and its anti-message occur in the same queue, they immediately annihilate one another. Thus the result of enqueueing a message may be to shorten the queue by one message rather than lengthen it by one. We depict the anti-message of m by m-1.

Johnson and Zwaenepoel [24] proposed sender based message logging for deterministic systems, where each message is logged in volatile memory on the machine from which the message is sent. The massage log is then asynchronously written to stable storage, without delaying the computation, as part of the sender's periodic checkpoint. Johnson and Zwaenepoel [25] used optimistic message logging and checkpointing to determine the maximum recoverable state, where every received message is logged.

In the present study, we propose a minimum-process coordinated checkpointing algorithm for checkpointing deterministic distributed applications on mobile systems. We eliminate useless checkpoints as well as blocking of processes during checkpoints at the cost of logging anti-messages of very few messages during checkpointing.

From this section, input the body of your manuscript according to the constitution that you had. There is a limit of 12 single spaced, double-column pages for each article in the Journal. The deadline for receiving your paper in PDF is two weeks after the paper's acceptation for publication.

2. THE PROPOSED CHECKPOINTING ALGORITHM

2.1 System Model

There are n spatially separated sequential processes denoted by P_0, P_1,.., P_{n-1}, running on MHs or MSSs, constituting a mobile distributed computing system. Each MH/MSS has one process running on it. The processes do not share memory or 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 also assume that the processes are deterministic as in [24, 25, 28].

2.2 Basic Idea

The proposed algorithm is based on keeping track of direct dependencies among processes. Similar to [5,11,28], initiator process captures the transitive dependencies by direct dependencies. The initiator process (say P_{in}) sends the checkpoint request to P_i only if P_{in} is directly dependent upon P_i. Similarly, P_i sends the checkpoint request to any process P_j only if P_i is directly dependent upon P_j.

During the checkpointing procedure, 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. In order to avoid such orphan messages, Cao and Singhal[5] proposed that P_i should take a forced checkpoint before processing m. If P_i receives a checkpoint request after processing m, then the forced checkpoint already taken is converted into tentative one. By doing so, m will not become orphan. We propose that the anti-messages of only those messages, which can become

orphan, should be recorded at the receiver end. In deterministic systems, orphan messages are received as duplicate messages on recovery. A duplicate message is annihilated by its anti-message at the receiver end before processing. Hence, in deterministic distributed systems, an orphan message in global checkpoint does not create any inconsistency during recovery if its anti-message is logged at the receiver end. By doing so, we avoid the blocking of processes as well as the useless checkpoints in minimum-process checkpointing. It should be noted that in minimum-process coordinated checkpointing, some useless checkpoints are taken or blocking of processes takes place [3,4]. The overheads of logging a few anti-messages may be negligible as compared to taking some useless checkpoints or blocking the processes during checkpointing.

The initiator MSS computes minset [subset of the minimum set] on the basis of dependencies maintained locally; and sends the checkpoint request along with the minset [ ] to the relevant MSSs. On receiving checkpoint request, an MSS asks concerned processes to checkpoint and computes new processes for the minimum set. By using this technique, we have tried to optimize the number of messages between MSSs.

2.3 An Example

We explain our checkpointing algorithm with the help of an example. In Fig. 1, at time t_1, P_2 initiates checkpointing process. $ddv_2[1] = 1$ due to m_1; and $ddv_1[4] = 1$ due to m_2. On the receipt of m_0, P_2 does not set $ddv_2[3] = 1$, because, P_3 has taken its permanent checkpoint after sending m_0. We assume that P_1 and P_2 are in the cell of the same MSS, say MSS_{in}. MSS_{in} computes minset (subset of minimum set) on the basis of ddv vectors maintained at MSS_{in}, which in case of Figure 1 is $\{P_1, P_2, P_4\}$. Therefore, P_2 sends checkpoint request to P_1 and P_4. After taking its tentative checkpoint, P_1 sends m_4 to P_3. P_3 logs $m4^{-1}$. After taking its checkpoint, P_4 also finds that it was dependent upon P_5 before taking its checkpoint due to m_6 and P_5 is not in the minimum set computed so far. Therefore, P_4 sends checkpoint request to P_5. On receiving the checkpoint request, P_5 takes its tentative checkpoint. At time t_2, P_2 receives responses from all relevant processes and sends the commit request along with the minimum set $[\{P_1, P_2, P_4, P_5\}]$ to all processes. When a process, in the minimum set, receives the commit message, converts its tentative checkpoint into permanent one. In this example, $\{C_{11}, C\text{-}_{21}, C_{30}, C_{41}, C_{51}, m_4^{-1}\}$ constitute a recovery line. It should be noted that, in the recorded global state, m_4 is an orphan message and its anti-message is also recorded at the receiver end.

2.4 The Checkpointing Algorithm

When an MH sends an application message, it needs to first send to its local MSS over the wireless cell. The MSS can piggyback appropriate information onto the application message, and then route it to the appropriate destination.

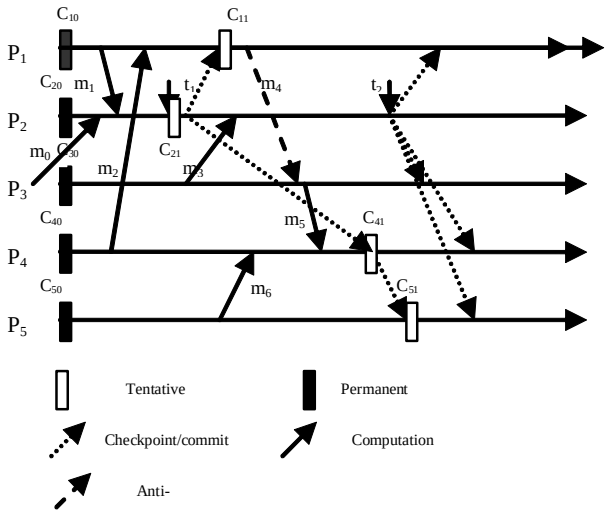

Figure 1: An Example

Conversely, when the MSS receives an application message to be forwarded to a local MH, it first updates the relevant vectors that it maintains for the MH, strips all piggybacked information from the message, and then forwards it to the MH. Thus, an MH sends and receives application messages that do not contain any additional information; it is only responsible for checkpointing its local state appropriately and transferring it to the MSS.

Each process P_i can initiate the checkpointing process. Initiator MSS initiates and coordinates checkpointing process on behalf of MH_i. It computes minset (subset of the minimum set on the basis of direct dependencies maintained locally); and sends c_req along with minset to an MSS if the later supports at least one process in the minset. It also updates its tminset on the basis of minset. We assume that concurrent invocations of the algorithm do not occur.

On receiving the c-req, along with the minset from the initiator MSS, an MSS, say MSS_i, takes the following actions. It updates its tminset on the basis of minset. It sends the c_req to P_i if the following conditions are met: (i) P_i is running in its cell (ii) P_i is a member of the minset and (iii) c_req has not been sent to P_i. If no such process is found, MSS_i ignores the c_req. Otherwise, on the basis of tminset, ddv vectors of processes in its cell, initial ddv vectors of other processes, it computes tnp_minset . If tnp_minset is not empty, MSS_i sends c_req along with tminset, tnp_minset to an MSS, if the later supports at least one process in the tnp_minset. MSS_i updates np_minset, tminset on the basis of tnp_minset and initializes tnp_minset.

On receiving c_req along with tminset, tnp_minset from some MSS, an MSS, say MSS_j, takes the following actions. It updates its own tminset on the basis of received tminset, tnp_minset and finds any process P_k such that P_k is running in its cell, P_k has not been sent c_req and P_k is in tnp_minset. If no such process exists, it simply ignores this request. Otherwise, it sends the checkpoint request to P_k. On the basis of tminset, ddv[] of its processes and initial ddv[] of other

processes, it computes tnp_minset. If tnp_minset is not empty, MSS_j sends the checkpoint request along with tminset, tnp_minset to an MSS, which supports at least one process in the tnp_minset. MSS_j updates np_minset, tminset on the basis of tnp_minset. It also initializes tnp_minset.

For a disconnected MH, that is a member of minimum set, the MSS that has its disconnected checkpoint, converts its disconnected checkpoint into tentative one.

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

When the initiator MSS receives a response from some MSS, it updates its minset on the basis of np_minset, received along with the response. Finally, initiator MSS sends commit/ abort to all the processes. When a process in the minimum set receives the commit request, it converts its tentative checkpoint into permanent one and discards its earlier permanent checkpoint, if any.

3. A PERFORMANCE EVALUATION

3.1 General Comparison with the Cao-Singhal Algorithm [5]:

In[5], some useless checkpoint requests are sent in the algorithm [5]; whereas, in the proposed protocol, no such useless checkpoint requests are sent. In algorithm [5], when P_i sends checkpoint request to P_j, it also piggybacks $csn_i[j]$ and a data structure MR. MR is an array of n pairs and each pair contains two fields: csn and r, where csn contains the csn number and r is a bit vector of length n. MR provides information to the request receivers on checkpoint request propagation decision-making. $csn_i[j]$ enables P_j to decide whether P_j inherits the request. These data structures are piggybacked onto checkpoint requests to handle useless checkpoint requests. In the proposed protocol, no useless checkpoint request is sent; therefore, there is no need to piggyback these data structures onto checkpoint requests. The $csn_i[j]$ is integer; its size is 4 bytes. In worst case the size of MR[] is (4n +n/8) bytes (n is the number of processes in the distributed system). In the proposed protocol, tminset and tnp_minset are piggybacked onto checkpoint requests. Size of each data structure is: n/8 bytes. The extra bytes piggybacked onto each checkpoint request in the algorithm [5] as compared to the proposed one are: (29n+32)/8. The number of useless checkpoint requests in [5] depends upon the number of processes, message sending rate, dependency pattern of processes etc. In some cases, the number of useless checkpoint requests in [5] may be exceedingly high. The useless checkpoint requests further increase the message complexity of the algorithm [5]. In the proposed protocol, the exact minimum set is broadcasted on the static network along with commit request, whereas in the Cao-Singhal [5] algorithm, only commit request is broadcasted. The size of the minimum set is n/8 bytes.

Concurrent executions of the algorithm are allowed in [5]. The algorithm [5] may lead to inconsistencies during its concurrent executions [20]. The proposed algorithm can be modified to allow concurrent executions on the basis of the methodology proposed in [20]. We do not compare our algorithm with the Singh-Cabillic algorithm [28], as it may lead to inconsistencies [Refer Section 2.1].

3.2 Comparison with other Algorithms

We use following notations to compare our 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_{bst}: 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 [5].
- N_{ind}: number of useless mutable checkpoints in the proposed protocol.
- 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 [5].
- N_{dep}: average number of processes on which a process depends.

Performance of our Algorithm: *The Synchronization message overhead*: In the first phase, a process taking a tentative checkpoint needs two system messages: request and reply. A process may receive more than one request for the same checkpoint initiation from different processes. However, we have used some techniques to reduce the duplicate checkpoint requests. Thus the system overhead is approximately $2*N_{min}*C_{pp}$ in the first phase. In the second

phase, the commit requested is broadcasted on the static network; and the system overhead is C_{bst}.

Number of processes taking checkpoints: It requires only minimum number of processes to take their checkpoints.

A Comparative Study: The blocking time of the Koo-Toueg [11] protocol is highest, followed by Cao-Singhal [4] algorithm. The other schemes are non-blocking [5], [8], like the proposed one. In Elnozahy *et al.* [8] algorithm, all processes are required to take their checkpoints in an initiation. In the protocols [4], [11], and the proposed one, only minimum numbers of processes record their checkpoints. The message overhead in the proposed protocol is greater than [8], but less than [5]. The algorithms proposed in [4], [5], [8] and [11], assume that the processes are non-deterministic, whereas, we assume in the proposed algorithm that the processes are deterministic in nature.

Table 1
A Comparison of System Performance

	Cao-Singhal [4]	*Cao-Singhal [5]*	*Koo-Toeg Algorithm [11]*	*Elnozahy et al. [8]*	*Proposed Algorithm*
Avg. blocking Time	$2T_{st}$	0	$N_{min}*T_{ch}$	0	0
Average No. of checkpoints	N_{min}	$N_{min}+N_{mut}$	N_{min}	N	N_{min}
Average Message Overhead	$3C_{bst}+2C_{wireless}+2 N_{mss}*C_{st}+3N_{mh}* C_{wl}$	$2*N_{min}*C_{pp}+C_{bst}+ N_{ucr}*C_{pp}$	$3*N_{min}*C_{pp}* N_{dep}$	$2*C_{bst}+ N *C_{pp}$	$2*N_{min}* C_{pp}+C_{bst}$

4. CONCLUSIONS

In this paper, we have proposed a minimum-process non-intrusive checkpointing protocol for deterministic mobile distributed systems, where no useless checkpoints are taken. 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; we eliminate both by logging anti-messages of selective messages at the receiver end only during the checkpointing period. The overheads of logging a few anti-messages may be negligible as compared to taking some useless checkpoints or blocking the processes during checkpointing.

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, 73-80, 1994.

[2] Baldoni R., Hélary J. M., Mostefaoui A. and Raynal M., "A Communication-Induced Checkpointing Protocol that Ensures Rollback-Dependency Trackability," Proceedings of the International Symposium on Fault-Tolerant-Computing Systems, 68-77, 1997.

[3] Cao G. and Singhal M., "On Coordinated Checkpointing in Distributed Systems", *IEEE Transactions on Parallel and Distributed Systems*, **9**(12), 1213-1225, 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*, 37-44, 1998.

[5] Cao G. and Singhal M., "Mutable Checkpoints: A New Checkpointing Approach for Mobile Computing systems," *IEEE Transaction on Parallel and Distributed Systems*, **12**(2), 157-172, 2001.

[6] Chandy K. M. and Lamport L., "Distributed Snapshots: Determining Global State of Distributed Systems," *ACM Transaction on Computing Systems*, **3**(1), 63-75, 1985.

[7] 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*, **34**(3), 375-408, 2002.

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

[9] Hélary J. M., Mostefaoui A. and Raynal M., "Communication-Induced Determination of Consistent Snapshots," Proceedings of the 28th International Symposium on Fault-Tolerant Computing, 208-217, June 1998.

[10] Higaki H. and Takizawa M., "Checkpoint-recovery Protocol for Reliable Mobile Systems," Trans. of Information processing Japan, **40**(1), 236-244, 1999.

[11] Koo R. and Toueg S., "Checkpointing and Roll-Back Recovery for Distributed Systems," *IEEE Trans. on Software Engineering*, **13**(1), 23-31, 1987.

[12] Neves N. and Fuchs W. K., "Adaptive Recovery for Mobile Environments," Communications of the ACM, **40**(1), 68-74, 1997.

[13] 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, 491-95, 2005.

[14] 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, 16-25, 1996.

[15] Prakash R. and Singhal M., "Low-Cost Checkpointing and Failure Recovery in Mobile Computing Systems," *IEEE Transaction on Parallel and Distributed Systems*, **7**(10), 1035-1048, 1996.

[16] Ssu K. F., Yao B., Fuchs W. K. and Neves N. F., "Adaptive Checkpointing with Storage Management for Mobile Environments," *IEEE Transactions on Reliability*, **48**(4), 315-324, 1999.

[17] J. L. Kim, T. Park, "An Efficient Protocol for Checkpointing Recovery in Distributed Systems," *IEEE Trans. Parallel and Distributed Systems*, 955-960, 1993.

[18] L. Kumar, M. Misra, R. C. Joshi, "Checkpointing in Distributed Computing Systems" Book Chapter "Concurrency in Dependable Computing", 273-92, 2002.

[19] L. Kumar, M. Misra, R. C. Joshi, "Low Overhead Optimal Checkpointing for Mobile Distributed Systems" Proceedings. 19th IEEE International Conference on Data Engineering, 686–88, 2003.

[20] Ni, W., S. Vrbsky and S. Ray, "Pitfalls in Distributed Nonblocking Checkpointing", *Journal of Interconnection Networks*, **1**(5), 47-78, 2004.

[21] L. Lamport, "Time, Clocks and Ordering of Events in a Distributed System" Comm. ACM, **21**(7), 558-565, 1978.

[22] Silva, L. M. and J. G. Silva, "Global Checkpointing for Distributed Programs", Proc. 11th Symp. Reliable Distributed Systems, 155-62, 1992.

[23] David R. Jefferson, "Virtual Time", *ACM Transactions on Programming Languages and Systems*, **7**(3), 404-425, 1985.

[24] Johnson, D. B., Zwaenepoel, W., "Sender-based Message Logging", In Proceedingss of 17th International Symposium on Fault-Tolerant Computing, 14-19, 1987.

[25] Johnson, D. B., Zwaenepoel, W., "Recovery in Distributed Systems using Optimistic Message Logging and Checkpointing. 171-181, 1988.

[26] Parveen Kumar, Lalit Kumar, R. K. Chauhan, "A Non-Intrusive Hybrid Synchronous Checkpointing Protocol for Mobile Systems", *IETE Journal of Research*, **52**(2&3), 2006.

[27] Parveen Kumar, "A Low-Cost Hybrid Coordinated Checkpointing Protocol for Mobile Distributed Systems", To appear in Mobile Information Systems.

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

[29] Lalit Kumar Awasthi, P. Kumar, "A Synchronous Checkpointing Protocol for Mobile Distributed Systems: Probabilistic Approach" *International Journal of Information and Computer Security*, **1**(3), 298-314.