A Comparative Study on Snapshot Protocols for Mobile Distributed Systems

Vijaya Kapoor
Shri Venkateshwara University
Gajraula, UP (INDIA)

Parveen Kumar
Amity University
Gurgaon, Haryana (INDIA)

ABSTRACT
In MDS (Mobile Distributed Systems) , we come across some issues like: low bandwidth of wireless channels , mobility, and lack of stable storage on mobile nodes, limited battery power ,disconnections and high failure rate of mobile nodes. Fault Tolerance Techniques enable systems to perform tasks in the presence of faults. In case of a fault in DS, snapshot enables the execution of a program to be resumed from a previous consistent Global State rather than resuming the execution from the beginning. Thus, a lot of useful processing amount is lost because of the fault is significantly condensed. Coordinated global Snapshot is an effective FTT (Fault Tolerant Technique) in DS(Distributed Systems), as it avoids the domino effect and require minimum storage requirement. In this paper, we will study the accessible snapshot compilation schemes for DS & MDS. Then, a comparative analysis of the different schemes will be performed.

Keywords
Fault tolerance, Coordinated snapshot, Message logging and Mobile Distributed Systems

1. INTRODUCTION
1.1 Definitions
Snapshot: It is a designated place in a program at which normal process is interrupted specifically to preserve the status information necessary to allow recommencement of processing at a later time. A snapshot is a local state of a process saved on stable storage. By periodically invoking the snapshot process, one can save the status of a program at expected intervals.

Rollback Recovery: In case of failure one may restart computation from the last snapshot thereby avoiding repeating computation from the starting. The process of recommencing computation by rolling back to a saved state is called rollback recovery. Besides its use to recover from failures, snapshot is also used in debugging distributed programs and migrating processes in multiprocessor system. In debugging distributed programs, state changes of a process during execution are monitored at various time instances. Snapshots assist in such monitoring. In Distributed System the load of the processors is balanced by moving processes from heavily loaded processors to lightly loaded ones. Snapshot a process periodically provides the information necessary to move it from one processor to another [16]. With snapshot, an arbitrary temporal section of a program's runtime can be extracted for exhaustive analysis without the need to restart the program from beginning.

Global State: In MDS, since the processes do not have shared memory, a Global State of the system is defined as a set of local states, one from each process. The state of channels resulting to a global state is the set of messages sent but not yet received.

Orphan Message: A message whose send event is lost but received event is recorded.

Consistent Global State: A global state is said to be "Consistent" if it contains no orphan message. To recover from a failure, the system restarts its implementation from a previous consistent global state saved on the stable storage during fault-free execution.

In DS, snapshot can be Independent, Coordinated [3], [8], [11] or Quasi-Synchronous [2], [9]. Message Logging is also used for fault tolerance in DS [14].

Under the Asynchronous approach, snapshots at each process are taken autonomously without any synchronization among the processes. Because of absence of synchronization, there is no guarantee that a set of local snapshots taken will be a consistent set of snapshots. It may require cascaded rollbacks that may lead to the initial state due to domino-effect [5].

Domino-effect, may lead to loss of a large amount of valuable work. A process may record unproductive snapshots that will never be part of a consistent GS. Ineffective snapshots are unwanted. They consume the limited resources of the MDS and do not progress the recovery line.

Independent snapshot compilation forces each process to store many snapshots. It requires periodically garbage collection process to destroy the snapshots that are no longer needed. In order to determine a consistent global snapshot during recovery, the processes store the dependencies among their snapshots during regular operations. Hence, independent snapshot compilation protocols are not appropriate for MDS.

Communication-induced snapshot compilation does not lead to the domino effect. It requires processes to record some of their snapshots independently. However, process independence is forced to guarantee the eventual progress of the recovery line and therefore processes may be enforced to record additional snapshots. The snapshots that a process records independently are called local snapshots, while those that a process is forced to record are called forced snapshots.

Communication-induced snapshot compilation piggybacks protocol related information on each application message [6].

In message-logging based snapshot compilation schemes, when a process fails, a new process is generated in its place. The newly created process is given the suitable recorded LS (local state) and then the logged communications are replayed in the correct order. These schemes require that once a stopped process recovers, its state is required to be consistent with the states of the other processes [6].

In coordinated or synchronous snapshot compilation, processes record their snapshots in coordination with each other. In such schemes, recorded GS is forever consistent. Generally it is based on two-stage commit arrangement. In the first stage, processes record tentative snapshots and in the second stage, these are made permanent. The major benefit is

that only one permanent snapshot and at most one tentative snapshot is necessary to be stored. In case of error, the system rollbacks to last steady snapshot state. A permanent snapshot cannot be undone. It makes sure that the working required to achieve the snapshot state will not be repeated. A tentative snapshot can be destroyed or committed. The coordinated snapshot compilation procedures can be divided into two categories: intrusive and non-intrusive. In intrusive procedures, some intrusion of processes takes place during snapshot compilation. In non-intrusive algorithms, no intrusion of processes is compulsory during snapshot compilation [2, 3, 4]. The coordinated snapshot compilation algorithms can also be classified into following two categories: selective-process and all process protocols. In all-process coordinated snapshot compilation algorithms, it is mandatory for every process to record its snapshot in a snapshot commencement. In selective-process algorithms, only least interacting processes are forced to record their snapshots in a commencement.

The occurrence of mobile nodes in a DCS (Distributed Computing System) invites new challenges that need appropriate treatment while designing a snapshot compilation algorithm for such systems. These challenges are mobility, disconnections, deficient in stable storage, inadequate power source, susceptible to physical spoil[1].

Selective-process coordinated snapshot compilation is considered salient approach to introduce fault tolerance in MDS transparently. This scheme is domino-free. It maintains at most two snapshots of a process on stable storage. It requires least number of processes to participate in snapshot compilation. But, it leads to extra synchronization communications, intrusion of the underlying working or taking some ineffective snapshots**.** A good snapshot compilation procedure for MDS should have small memory expenses on MHs (Mobile Hosts), low outlay on wireless channels and should keep away from awakening of an MH in doze mode operation. The disconnection of an MH should not lead to never-ending wait state. The procedure should force least number of processes to record their local snapshots [20].

In selective-process synchronous snapshot compilation, the originator process asks all communicating processes to record their tentative snapshots. In this scheme, if a solitary process fails to record its snapshot; all the snapshot compilation effort goes waste, because, each process has to terminate its tentative snapshot. In order to record the tentative snapshot, an MH is required to transmit big snapshot data to its local MSS over wireless channels. Due to repeated aborts, total snapshot compilation effort may be exceedingly high, which may be disagreeable in MDS due to inadequate possession of resources. Repeated aborts may happen in MDS due to tired battery, sudden disconnection or bad wireless connectivity [6].

2. Summary of Related Works

Cao and Singhal [3] designed selective-process intrusion-based procedure for snapshot compilation in MDS. In this procedure, intrusion time is considerably abridged with respect to [9]. Direct dependencies of every process is retained in a bit array of length n for n processes. Originator process catches the direct dependency vectors of all processes and finds out minimum set. Then, the snapshot appeal is sent along with the minimum set to all processes. During the period, a process sends its dependency vector to the originator process and captures the minimum set, it continues to be in the intrusion period. A process records its snapshot if it is in the minimum set.

Neves et al. [10] designed a loosely synchronized snapshot compilation procedure that eliminates the operating cost of synchronization. In this approach it is assumed that the clocks at the processes are loosely synchronized. Loosely synchronized clocks can activate the local snapshots at all the processes approximately at the same time without a controller. After recording a snapshot, a process waits for a period, which is total of greatest time to perceive a failure of other process in the system and the highest divergence between clocks. It is understood that all snapshots concerning to a specific coordination session have been considered without the need of sharing any communication. If a failure encountered, it is detected within the particular time and the procedure is abandoned.

The Chandy-Lamport [5] procedure is one of the earliest non-intrusive all-process coordinated snapshot compilation procedure for static nodes. In this protocol, *markers* are transferred along all paths in the network which reaches to a message complexity of $O(N^2)$ and requires paths to be FIFO. To relax the FIFO hypothesis, Lai and Yang proposed an algorithm [6]. In this algorithm, when a process records a snapshot, it piggybacks a flag to the communication it sends out from each channel. The recipient considers the piggybacked flag to see if there is a requirement to record a snapshot before processing the communication. If so, it records a snapshot before processing the communication to avoid an inconsistency. Each process maintains the entire communication history on each channel as part of the local snapshot to record the channel information. It requires all processes to record snapshots. Elnozahy [7] et al. proposed an all-process non-intrusion synchronous snapshot compilation procedure with a message complexity of O(N). They use snapshot sequence numbers to identify orphan messages, thus avoiding the need for processes to be blocked during snapshot compilation. However, this approach requires the initiator to communicate with all processes in the computation.

In the procedure proposed by Silva and Silva [18], the processes which did not communicate with others during the previous snapshot compilation interval do not need to record new snapshots. The above discussed protocols aim to decrease the operating cost connected with synchronized snapshot compilation. Studies are performed in order to decrease the synchronization communications, reduce the number of processes to snapshot and to make the protocols non-intrusive. The above mentioned algorithms are either selective-process or non-intrusive. Prakash and Singhal [13] were first to give selective-process non-intrusive coordinated snapshot compilation protocol for MDS. But their procedure may lead to inconsistencies [3]. In [3], it was proved that there does not exist a selective-process non-intrusive coordinated snapshot compilation algorithm. Hence in selective-process coordinated snapshot compilation algorithms, some intrusion of the processes records place [3], or some useless snapshots are taken [4].

Koo-Toueg [9] designed a selective-process synchronous snapshot compilation protocol which relaxes the idea that all communications are atomic. It reduces the number of synchronization messages and number of snapshots. The algorithm consists of two phases. During the first phase, the snapshot maker identity all processes with which it has communicated since the last snapshot and sends them a request. Upon receiving the request, each process in turn identifies all process it has communicated with since the last

snapshot and sends them a request, and so on, until no more processes can be identified. During the second phase, all processes identified in the first phase take a snapshot. The result is a consistent snapshot that involves only the participate processes In this protocol, when second phase terminates successfully then only a process can send a message, although, receiving messages after the snapshot is allowable.

Garg and Kumar[23] proposed a non-blocking coordinated snapshot algorithm for mobile computing systems, which requires only a minimum number of processes to take permanent snapshots. They reduce the message complexity as compared to the Cao-Singhal algorithm[4], while keeping the number of useless snapshots unchanged. They also address the related issues like: failures during snapshot process, disconnections, concurrent initiations of the algorithm and maintaining exact dependencies among processes. Finally, they presented an optimization technique, which significantly reduces the number of useless snapshots at the cost of minor increase in the message complexity. In coordinated snapshot, if a single process fails to take its tentative snapshot; all the snapshot efforts were aborted. They try to reduce this effort by taking soft snapshots in the first phase at Mobile Hosts. They have proposed a non blocking coordinated snapshot protocol for MDS, where only minimum number of processes takes permanent snapshots. They have reduced the message complexity as compared to Cao & Singhal algorithm, while keeping the number of useless snapshots unchanged. The proposed algorithm was designed to impose low memory and computation overheads on MHs and low communication overheads on wireless channels. An MH can remain disconnected for an arbitrary period of time without affecting snapshot activity. They also try to minimize the loss of snapshot effort if some process fails to take its snapshot in the first phase but it will increase the synchronization overhead.

Kumar and Kumar[24] proposed a minimum set coordinated snapshot algorithm for mobile Distributed systems, which keeps track of direct dependencies of processes. Initiator MSS collects the direct dependency vectors of all processes, computes the tentative minimum set (minimum set or its subset), and sends the snapshot request along with the tentative minimum set to all MSSs. This step is taken to reduce the time to collect the coordinated snapshot. It will also reduce the number of useless snapshots and the blocking of the processes. Suppose, during the execution of the checkpointing algorithm, *Pi* takes its snapshot and sends *m* to *Pj*. *Pj* receives *m* such that it has not taken its snapshot for the current initiation and it does not know whether it will get the snapshot request. If *Pj* takes its snapshot 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 snapshot request. Therefore, *Pj* takes its induced snapshot before processing *m*. An induced snapshot is similar to the mutable snapshot [4]. In this case, most probably, *Pj* will get the snapshot request and its induced snapshot will be converted into permanent one. There is a less probability that *Pj* will not get the snapshot request and its induced snapshot will be discarded. Alternatively, if there is not a good probability that *Pj* will get the snapshot request, *Pj* buffers *m* till it takes its snapshot or receives the commit message. They have tried to minimize the number of useless snapshots 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 snapshot requests and reduces the number of duplicate snapshot requests as compared to [4].

P. Kumar [25] proposed a hybrid snapshot algorithm for mobile distributed systems. In minimum-process snapshot, some processes, having low communication activity, may not be included in the minimum set for several snapshot 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 snapshot state and the loss of computation at such processes may be exceedingly high. In all-process snapshot, recovery line is advanced for each process after every global snapshot but the snapshot overhead may be exceedingly high, especially in mobile environments due to frequent snapshots. MHs utilize the stable storage at the MSSs to store snapshots of the MHs . Thus, to balance the snapshot overhead and the loss of computation on recovery, a hybrid snapshot algorithm for mobile distributed systems is proposed, where an all-process snapshot is taken after certain number of minimum-process snapshots.

A strategy is proposed to optimize the size of the csn. In order to address different checkpointing intervals, the integer csn is replaced with k-bit CI(checkpointing interval). 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.

The minimum-process snapshot algorithm is based on keeping track of direct dependencies of processes. Initiator process collects the direct dependency vectors of all processes, computes minimum set, and sends the checkpoint request along with the minimum set to all processes. In this way, blocking time has been significantly reduced as compared to [9].

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, processes are blocked during this period.

3. CONCLUSION

MDS (Mobile Distributed Systems) pose new challenging problems in designing fault tolerant systems because of the dynamics of limited bandwidth and mobility available on wireless links. Traditional fault tolerance techniques cannot be applied to these systems. As the snapshot scheme saves the status of system at some transitional points (Snapshots) and a rollback to the latest saved state is done at the occurrence of a failure. Therefore, it reduces the rollback portion through at the cost of additional overheads for snapshots. In this paper we have provided the fundamental concepts of snapshot algorithms. Most of the Coordinated snapshot protocols able to reduce the useless snapshots and blocking of processes at very low cost for maintaining and collecting direct dependencies and piggybacking checkpoint sequence numbers onto normal messages .Coordinated snapshots generally simplifies garbage collection and recovery, provide good performance in practice. Therefore we have compared and reviewed Selective Intrusive , Selective Non Intrusive, All process Intrusive, All process Non Intrusive & Hybrid approaches of Coordinated snapshot protocols to rollback recovery for MDS.

More general approaches showing the effect of the snapshot protocols on MDS may also be considered for further study.

4. REFERENCES

[1] Acharya A. and Badrinath B. R., "Checkpointing Distributed Applications on Mobile Computers," Proceedings of the 3rd International Conference on Parallel and Distributed Information Systems, pp. 73-80, September 1994.

[2] Cao G. and Singhal M., "On coordinated checkpointing in Distributed Systems", IEEE Transactions on Parallel and Distributed Systems, vol. 9, no.12, pp. 1213-1225, Dec 1998.

[3] Cao G. and Singhal M., "On the Impossibility of Min-process Non-blocking Checkpointing and an Efficient Checkpointing Algorithm for Mobile Computing Systems," Proceedings of International Conference on Parallel Processing, pp. 37-44, August 1998.

[4] Cao G. and Singhal M., "Mutable Checkpoints: A New Checkpointing Approach for Mobile Computing systems," IEEE Transaction On Parallel and Distributed Systems, vol. 12, no. 2, pp. 157-172, February 2001.

[5] Chandy K. M. and Lamport L., "Distributed Snapshots: Determining Global State of Distributed Systems," ACM Transaction on Computing Systems, vol. 3, no. 1, pp. 63-75, February 1985.

[6] Elnozahy E.N., Alvisi L., Wang Y.M. and Johnson D.B., "A Survey of Rollback-Recovery Protocols in Message-Passing Systems," ACM Computing Surveys, vol. 34, no. 3, pp. 375-408, 2002.

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

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

[9] 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.

[10] Neves N. and Fuchs W. K., "Adaptive Recovery for Mobile Environments, " Communications of the ACM, vol. 40, no. 1, pp. 68-74, January 1997.

[11] Parveen Kumar and Poonam Gahlan, "A Low-overhead Minimum Process Coordinated Checkpointing Algorithm for Mobile Distributed Systems", International Journal of Computer Applications, vol. 10, no. 6, pp 30-36 June 2010.

[12] Parveen Kumar and Ruchi Tuli, "Analysis of Recent checkpointing Techniques for Mobile Computing Systems", International Journal of Computer Science & Engineering, vol. 2, no. 3, August 2011.

[13] Prakash R. and Singhal M., "Low-Cost Checkpointing and Failure Recovery in Mobile Computing Systems," IEEE Transaction On Parallel and Distributed Systems, vol. 7, no. 10, pp. 1035-1048, October1996.

[14] Ssu K.F., Yao B., Fuchs W.K. and Neves N. F., "Adaptive Checkpointing with Storage Management for Mobile Environments," IEEE Transactions on Reliability, vol. 48, no. 4, pp. 315-324, December 1999.

[15] T. Park and J.L. Kim, "An efficient Protocol for checkpointing Recovery in Distributed Systems," IEEE Trans. Parallel and Distributed Systems, pp. 955-960, Aug. 1993.

[16] 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.

[17] L. Lamport, "Time, clocks and ordering of events in a distributed system" Comm. ACM, vol.21, no.7, pp. 558-565, July 1978.

[18] Silva, L.M. and J.G. Silva, "Global checkpointing for distributed programs", Proc. 11th symp. Reliable Distributed Systems, pp. 155-62, Oct. 1992.

[19] Mukesh Singhal, Niranjan G Shivaratri, Advanced Concepts in Operating Systems, vol 18, pp. 71, 2008.

[20] Kumar Parveen, Gupta Sunil Kumar, Chauhan R.K., "Backward Error Recovery Protocols in Distributed Mobile Systems: A Survey", Journal of Theoretical and Applied Information Technology, pp. 337-347, 2008.

[21] Murthy and Manoj, "Ad hoc Wireless Networks Architectures and Protocols", Pearson Education, 2004.

[22] Ruchi Tuli and Parveen Kumar, "Minimum Process Coordinated Checkpointing Scheme for Ad Hoc Networks", International Journal on AdHoc Networking Systems, vol. 1, no. 2, October 2011.

[23] Garg., R and Kumar, P., "A Nonblocking Coordinated Checkpointing Algorithm for Mobile Computing Systems, International Journal of Computer Science Issues, Vol. 7, Issue 3, No 3, May 2010

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

[25] **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.