Message Logging & Checkpointing *in* Mobile Computing

Lalit Kumar*
Parveen Kumar*
R. K. Chauhan**

ABSTRACT

Mobile computing raises many new issues, such as lack of stable storage; low bandwidth of wireless channels, high mobility and limited battery life. It becomes difficult for multiple mobile hosts to synchronously take checkpoints since the communication channels with the mobile hosts are less reliable and may disconnect even during checkpointing. In the synchronous checkpointing, every process has to abort the checkpoints, if some process fails to take the checkpoint; and a failure at a single node forces all or selective nodes to rollback to the last consistent checkpoint. MHs are prone to frequent failures and if a process on MH fails it becomes costly to rollback all processes. In this paper, we propose a non-intrusive checkpointing protocol, where mobile hosts take checkpoints independently and fixed hosts checkpoint synchronously. We also optimize the information piggybacked onto each message. In case of a failure at a process on an MH, the faulty process on the MH recovers independently of other processes in the distributed system.

Key words : Fault tolerance, checkpointing, message logging, independent checkpointing, consistent global state, distributed systems, coordinated checkpointing and mobile systems.

1. Introduction

A mobile distributed computing system (MDCS) is a distributed system where some of the processes are running on mobile hosts (MHs). An MH can change its geographical position freely from one cell to another or even to an area covered by no cell. A mobile host communicates with other nodes of the distributed 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. MSS has 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 [1]. A static node that has no support to MH can be considered as MSS with no MH.

Checkpointing is a technique that can be used for fault tolerance provisioning in distributed systems. A checkpoint is the state of a process on stable storage. In distributed system, a system state is said to be consistent if it contains no orphan message; i.e., a message whose receive event is recorded in the state, but its send event is not recorded in that state. To recover from a failure, the system restarts its execution from a previous consistent global state saved on the stable storage. This saves all the computation done up to the last checkpointed state and only the computation done after that needs to be redone [8]. In distributed systems, checkpointing can be independent (asynchronous), coordinated (synchronous) or quasi-synchronous [2], [10]. Message logging is also used for fault tolerance provisioning in distributed systems [16].

In the asynchronous protocols, processes take the local checkpoints independent of the other. In coordinated checkpointing, processes take checkpoints in such a manner that the resulting global state is consistent. Mostly the coordinated checkpointing protocol follows the two-phase commit structure. 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 and the recovery is very simple [9], [12].

In message logging protocols, each process periodically records its local state and logs the messages that it receives after having recorded that state. When a process crashes, a new process is created in its place. The new process is given the

appropriate recorded local state, and then the logged messages are replayed in the order they were originally received at the original process. All message-logging protocols require that once a crashed process recovers, its state is consistent with the states of the other processes [8], [16], [15], [3].

The MSSs can easily take the consistent checkpoints using synchronous protocols [9], [12], [13] since they can communicate with each other by using the high-speed network and they have enough stable storage to store their state information. There are synchronous checkpointing protocols for mobile systems [5], [6], [14], [20]. Here, all processes need to abort the checkpoints in case of a fault at any node. It becomes difficult for multiple mobile hosts to synchronously take checkpoints since the communication channels with the mobile hosts are less reliable and may disconnect even during taking checkpoints. Higaki and Takizawa [11] have shown that there is a good probability that at least one MH fails to take the checkpoint synchronously with other hosts and it is difficult to take a synchronous checkpoint.

Therefore, some asynchronous checkpointing protocols have been proposed for mobile systems [1], [11], [15]. Acharya and Badrinath [1] introduced a two-phase method for taking global consistent checkpoints. They proposed that checkpoints be stored on the stable storage of mobile support stations instead of on mobile hosts. In their protocol, processes alternate between two states, SEND and RECV. If a process is in the send mode and receives a message, it is forced to take a checkpoint. During recovery, the global state is reconstructed from a set of checkpoints for each process.

Pradhan et al analytically evaluated the performance of different state saving protocols and hand off strategies [15]. They also proposed storing checkpoints and message logs at MSSs. Their result indicates that message logging is suitable for mobile environments except in cases where mobile host has high mobility, failure rate is high, and wireless bandwidth is low. Bin Yao et al. [3] describes a receiver based message logging protocol for mobile hosts, mobile support stations and home agents in a Mobile IP environment, which guarantees independent recovery. Checkpointing is utilized to limit log size and recovery latency. Higaki and Takizawa [11] proposed a checkpointing protocol where mobile hosts checkpoint independently and fixed ones synchronously and all processes are blocked during taking synchronous checkpoints.

Neves N. et al. [17] proposed a time based coordinated checkpointed algorithm for mobile environments, which incurs zero message overhead and only piggybacks csn.

In this paper, we propose a hybrid checkpointing protocol having following characteristics. It is non-blocking. MHs take checkpoints independently. If an MH fails to take its checkpoint and transfer it to the current MSS, it can try later. MSSs take checkpoints synchronously. In order to realize non-blocking during checkpointing, we use three bits CI to be piggybacked onto normal messages instead of integer csn used in literature [6], [9], [14], [20]. If a process on an MH, that his higher probability of failure than MSS, fails, it can recover independently. When a process on an MSS fails, all processes rollback to most recent synchronous checkpoint. An MH uses its recent committed checkpoint and message logs to reach to a state consistent with synchronous checkpoint of MSSs. 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.

The rest of the paper is organized as follows. Section 2 presents system model. In Section 3, we describe the hybrid checkpointing algorithm. In section 4, we compare the proposed algorithm with existing ones. Section 5 presents conclusions.

2. System Model

Our system model is similar [5] and [11]. A mobile computing system consists of a large number of MHs and relatively fewer MSSs. An MSS has 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 cell is a logical or geographical area covered by an MSS. An MH can directly communicate with an MSS by a reliable FIFO wireless channel only if it is present in the cell supported by the MSS. There are n spatially separated sequential processes denoted by P_1, P_2, ..., P_n, running on mobile hosts (MHs) or on static hosts (MSSs), forming a mobile distributed computing system. Each MH or MSS has one process running on it. The processes do no share common memory or common clock. Message passing is the only way for processes to communicate with each other. Each process

progresses at its own speed and messages are exchanged through reliable channels, whose transmission delays are finite but arbitrary. The messages generated by the underlying computation are referred to as computation messages or simply messages, and are denoted by m_i. A process is in the cell of MSS means the process is either running on the MSS or on an MH supported by it. It also includes the processes of MHs, which have been disconnected from the MSS but their checkpoint related information is still with this MSS. The ith Checkpointing interval of a process denotes all the computation performed between its ith and $(i+1)^{th}$ checkpoint, including the ith checkpoint but not the $(i+1)^{th}$ checkpoint. We assume a three-bit sequence number for identifying the checkpointing intervals.

3. The Hybrid Checkpointing Algorithm

3.1 Basic Idea

It is difficult for multiple mobile hosts to synchronously take checkpoints since the channels with the mobile hosts are less reliable and may disconnect even during taking checkpoints [11]. Mobile hosts are prone to frequent failures and it is not advisable to rollback all processes in the event of an MH failure. Therefore, we have proposed independent checkpointing for MHs. All the to and fro messages of an MH pass through its current local MSS. Therefore, the current local MSS can easily log messages of the MHs in its cell. In case of an MH failure, it can recover independently by using message logs and checkpoints.

In order to address different checkpointing intervals (CIs), we have replaced integer csn (checkpoint sequence number) used in [6], [9], [14] with three bits CI. We have considered only eight different checkpointing intervals and so the information piggybacked onto application messages is just three bits as compared to ever increasing csn. We have not considered CI of one bit [13] or two bits suitable, because by using one bit CI we shall be able to distinguish only two CIs, and by using two bits we shall be able to distinguish four CIs, which we assume to be insufficient. In case of three bits, we shall be able to distinguish eight different CIs, and we infer that it is just sufficient for all practical purposes.

Whenever a fixed host takes its checkpoint, it updates its current CI and piggybacks current CI onto application messages. When a process receives the message and finds that sender's CI is higher than its own CI, it concludes that sender has taken the checkpoint for the new initiation; therefore, it takes the checkpoint before processing the message.

3.2 Maintenance of Different Checkpointing Intervals

Every process maintains current checkpoint interval (cci) and next checkpointing interval (nci).

Initially for a process cci and nci are [000] and [001] respectively. When a process updates its CIs, it sets cci = nci; nci = modulo 8(nci^{++}). When no checkpointing process is going on, all the processes are running in the same CIs.

3.3 Handling Node Mobility

We explain the handling of node mobility in our protocol with reference to Figure 1. Initially, MH_i is connected with MSS_j. MSS_j sends messages m_1 and m_2 to MH_i. MSS_j receives the acknowledgement of m_1 and not of m_2 from MH_i. After this, MH_i disconnects from MSS_j and connects to MSS_k. During disconnection period, MSS_j receives m_3 and m_4 for MH_i and buffers them. As MSS_j is one hop away from MSS_k, therefore, MSS_j transfers only in-transit messages (m_2) and buffered messages (m_3, m_4) to MSS_k. It doen not transfer tentative state log (tsl_{ji}) and message log (tml_{ji}) of MH_i to MSS_k. MSS_k sends in-transit and buffered messages to MH_i. MH_i processes the in-transit messages if it has not processed them; otherwise, it simply sends the acknowledgement. After processing in-transit and buffered

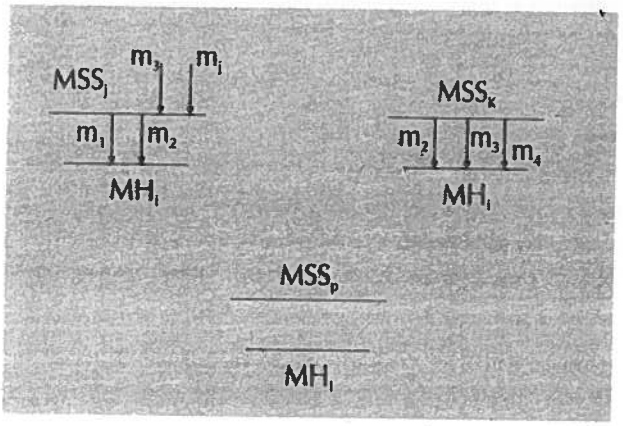

Figure 1

messages, MH_i starts normal functioning. Now, MH_i disconnects from MSS_k and connects to MSS_p. As MSS_k is one hop away from MSS_p, therefore, it only transfers in-transit and buffered messages to MSS_p. Now, MSS_j

becomes two hops away from MSS_p, therefore, it sends tsl_{ji} and tml_{ji} to MSS_p. In this way, all tentative state and message logs of an MH are on its current local MSS or on an MSS at most one hop away from the current MSS.

3.4 Checkpointing Protocol

We assume that a single process on an MSS initiates checkpointing.

(a) Algorithm executed at initiator MSS (say MSS_{in})

(i)	Initiator Process takes the checkpoint; updates its CIs;
(ii)	Initiator MSS sends tentative checkpoint request along with current CI of initiator process to all MSSs;
(iii)	Set $timer_i$; // $timer_i$ is a flag, set to 1 when maximum allowable time for collecting // hybrid checkpoint expires.
(iv)	wait for response;
(v)	At $timer_i$ or on receiving unsuccessful response from some MSS: Send message Abort checkpoints to all MSSs;
(vi)	On receiving successful responses from all MSSs: Send message Commit checkpoints to all MSSs.

(b) Algorithm executed at any MSS:

On receiving tentative checkpoint request from initiator MSS or on receiving a message for any process in its cell of higher cci then the local cci:

(i) processes running on the MSS take tentative checkpoints;
// processes on MSSs process the messages of higher cci only after taking checkpoints.
all tentative state and message logs of MHs are stored in the stable storage;
// processes on MHs or MSSs process the messages of higher cci only after completion of this task.

(ii) Inform the initiator MSS about the failure or successful execution of the above steps;

On receiving Abort message from the initiator MSS:	
(i)	All MSSs discard their tentative checkpoints of the current initiation;
(ii)	Undo the updating on CIs;
(iii)	All the state and message logs transferred to stable storage in the current initiation are removed from the stable storage; // they remain on temporary storage as priors to this initiation;

On receiving Commit message from the initiator MSS:	
(i)	All MSSs discard their earlier committed checkpoints;
(ii)	If an MH_i has taken checkpoint in the current checkpoint interval, then all previous committed message and state logs are discarded;

(c) Independent Checkpointing and Message logging of MHs

Each MH_i has to restart the computation from a state consistent with Coordinated Checkpoint (CC) taken by all MSSs. MH_i's checkpoint (say CM_i) may not be consistent with CC because MH_i takes CM_i independently of the other hosts. Hence, MH_i has to restart the computation by using message log. Here, the messages sent and received after CM_i by MH_i are stored in the message log in the stable storage of MSS. If MH_i restarts the computation, MH_i rollbacks to CM_i and replays the received messages after CM_i to get the state consistent with CC.

Let MH_i be in the cell of MSS_j. All the to and fro application messages of MH_i are logged in the volatile storage at MSS_j. Whenever MH_i wants to take its checkpoint; it takes its checkpoint and transfers it to MSS_j. If MH_i fails to take its checkpoint and transfer it to MSS_j, it can try later. If MH_i successfully transfers its checkpoint to MSS_j, all previous temporary message and state logs of MH_i are discarded. Previous temporary messages include messages that are received or sent by the MHi before taking the checkpoint and are stored in the temporary or volatile storage of different MSSs. These are available on MSS_j or some other MSSs that are situated at most one hop away from MSS_j.

When MSSs make their local checkpoints permanent, MH_i also makes its most recent checkpoint (temporary or permanent) permanent (say c_i) and store its message logs (temporary or permanent), after c_i, in the stable storage. The entire message logs and state logs before c_i are discarded. If MH_i has not taken any checkpoint, its all-previous message logs are stored in the stable

storage. It can reach to the state consistent with CC by starting from the initial state and re-computing the received messages, in the order, they were actually processed.

The order of messages received and sent by the MH may be different at the local MSS that logs the messages on behalf of the MH. The exact order of the messages at the MH can easily be known by the MSS if the MH gives monotonically increasing sequence number to each sent or received message and the MSS is made aware of these sequence numbers.

4. Comparisons with Existing Schemes

The Chandy-Lamport [7] algorithm is the earliest non-blocking coordinated checkpointing algorithm. During checkpointing, markers are sent along all channels in the network, which leads to a message complexity of O (N^2). It requires all processes to synchronously take checkpoints and channels need to be FIFO.

Elnozahy et al. [9] proposed an all process non-blocking coordinated checkpointing algorithm. They have used ever-increasing integer csn to be piggybacked onto normal messages. We use three bits CI instead of integer csn for the same purpose.

The Koo and Toueg [12] algorithm is a minimum process coordinated checkpointing algorithm. It requires processes to be blocked during checkpointing. Checkpointing includes the time to find the minimum interacting set of processes and to save their state of processes on stable storage, which may be too long. Therefore, this extensive blocking of processes may significantly reduce the performance of the system in mobile environments where some of the MHs may not be available due to disconnections. Each process uses monotonically increasing labels in its outgoing messages. It forces only interacting processes to synchronously take checkpoints.

Prakash et al. proposed a minimum process non-blocking coordinated checkpointing scheme for MDCSs [18]. But this protocol may lead to inconsistencies in some situations [5], [6]. They have used a bit array of length n, for n processes, to be piggybacked onto normal messages.

Cao and Singhal [5], proposed minimum process non-blocking checkpointing schemes for MDCSs by introducing the concept of mutable checkpoints. In their protocol, only minimum number of processes takes permanent checkpoints, but the actual number of processes that take checkpoints can be exceedingly high than the minimum required. The useless checkpoints are discarded on commit. L. Kumar et al. [14] and P. Kumar et al. [20] proposed a non-blocking minimum process checkpointing protocol for MDCS and significantly reduced the number of useless checkpoints as compared to [5]. These protocols use integer csn to be piggybacked onto normal messages and require interacting processes to synchronously take checkpoints. We use three bits CI in place of integer csn.

All the above-mentioned protocols i.e. [5], [6], [8], [9], [12], [14], [18], [20], require all or interacting processes to synchronously take checkpoints. In these protocols, if a single process fails to checkpoint, all processes abort their checkpoints taken in the current initiation. In our case, only MSSs take checkpoints synchronously. In our protocol, if an MH fails to checkpoint, it can try later. During synchronous checkpointing, MHs are not forced to checkpoint .We use three bits CI, to be piggybacked onto normal messages, to realize non-intrusiveness during checkpointing, instead of integer csn, used in literature [6], [9], [14], [20].

Acharya and Badrinath [1] gave the first checkpointing protocol for mobile systems based on quasi-synchronous approach. In this protocol, an MH is forced to take a local checkpoint whenever a message receipt is preceded by message sent at that MH. This algorithm has no control over checkpointing activity on MHs and depends totally on communication patterns. In worst case, the number of local checkpoints may be twice the number of computation messages, which may degrade the system performance. In our scheme, there is a strict control on checkpointing activity.

Higaki and Takizawa [11] have proposed a hybrid checkpointing protocol, where checkpoints are taken asynchronously by MHs and synchronously by MSSs. During checkpointing, MSSs coordinate to checkpoint. All processes are blocked during checkpointing that can significantly degrade the system performance. In our protocol, no blocking of processes takes place.

Bin Yao et al [3] proposed receiver based message logging for MHs and MSSs. In MDCS, the MSSs can

easily take consistent checkpoints since they can communicate with each over static network and have good stable storage.

5. Conclusions

In this paper, we have presented a non-intrusive checkpointing protocol for mobile distributed computing systems. It may be difficult for multiple mobile hosts to synchronously take their checkpoints due to their specific characteristics. In our protocol, the mobile hosts take their checkpoints asynchronously and MSSs take their checkpoints synchronously. Thus, it relives the MHs from taking the checkpoints in synchronization with MSSs. The protocol for checkpointing MSSs is non-intrusive. In order to realize non-intrusiveness, we use three bits to identify different checkpointing interval that is piggybacked onto normal messages, instead of ever-increasing integer checkpoint sequence number used in many protocols in the literature. The proposed protocol is optimal in using the scare resources of the MHs.

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. 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, pp. 68-77, June 1997.
3. Bin Yao, Kuo-Feng Ssu, W. Kect Fuchs, "Message Logging in Mobile Computing" Proceedings of international conference on FTCS, pp 294-301, 1999.
4. Briatico D., Ciuffoletti A. and Simoncini L., "A Distributed Domino-Effect Free Recovery Algorithm," Proceedings of International Symposium on Reliable Distributed Software and database, pp. 207-215, December 1984.
5. 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.
6. 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.
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. 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.
10. Hélary J. M., Mostefaoui A. and Raynal M., "Communication-Induced Determination of Consistent Snapshots," Proceedings of the 28th International Symposium on Fault-Tolerant Computing, pp. 208-217, June 1998.
11. 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.
12. 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.
13. L. Kumar, M. Misra, R.C. Joshi, "Checkpointing in Distributed Computing Systems" Book Chapter "Concurrency in Dependable Computing", pp. 273-92, 2002.
14. L. Kumar, M. Misra, R.C. Joshi, "Low overhead optimal checkpointing for mobile distributed systems"

Proceedings. 19th International Conference on IEEE Data Engineering, pp 686 88, 2003.

15. Pradhan D.K., Krishana P.P. and Vaidya N.H., "Recovery in Mobile Wireless Environment: Design and Trade-off Analysis," Proceedings 26th International Symposium on Fault-Tolerant Computing, pp. 16-25, 1996.
16. Richard G. G. and Singhal M., "Using Logging and Asynchronous Checkpointing to Implement Recoverable Distributed Shared Memory", Proceedings of the 12th symposium on Reliable Distributed Systems, pp. 58-67, October 1993.
17. Neves N. and Fuchs W. K., "Adaptive Recovery for Mobile Environments," Communications of the ACM, vol. 40, no. 1, pp. 68-74, January 1997.
18. 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.
19. 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.
20. 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.

* Dept. of CSE, National Institute of Technology, Hamirpur (HP), India.
**Dept of Computer Sc & Applications, K.U. Kurukshetra (HRY), India
lalitdec@yahoo.com; pk223475@yahoo.com