International Journal of Computer Applications (0975 – 8887)
Volume 17– No.4, March 2011

Minimum-Process Synchronous Checkpointing in Mobile Distributed Systems

Anil Kumar Panghal
HCTM, Kaithal

Mukesh Kumar Rana
HCTM,Kaithal

Parveen kumar
MIET,Meerut

ABSTRACT
Checkpointing is an efficient fault tolerance technique used in distributed systems. Due to the emerging challenges of the mobile distributed system as low bandwidth, mobility, lack of stable storage, frequent disconnections and limited battery life, the fault tolerance technique designed for distributed system can not directly implemented on mobile distributed systems(MDSs). This research paper presents an efficient low cost synchronous checkpointing algorithm which fit into the mobile environment.

Keywords
Checkpointing, Distributed system, Mobile computing, Synchronous

1. INTRODUCTION
One of the most challenging and interesting trend in computer and a telecommunication industry is the integration of mobile communication and computing. The resulting distributed network is referred to as mobile computing system [7]. Mobile computing is a new era that enhances the functionality of computing equipment by freeing communication from the location constraints of the wireless infrastructure.

Checkpointing is an important feature in distributed computing systems. It gives fault tolerance without requiring additional efforts from the programmer. A *checkpoint* is a snapshot of the current state of a process. It saves enough information in non-volatile stable storage such that, if the contents of the volatile storage are lost due to process failure, one can reconstruct the process state from the information saved in the non-volatile stable storage [1]. If the processes communicate with each other through messages, rolling back a process may cause some inconsistency. In the time since its last checkpoint, a process may have sent some messages. If it is rolled back and restarted from the point of its last checkpoint, it may create *orphan messages*, i.e., messages whose receive events are recorded in the states of the destination processes but the send events are lost. Similarly, messages received during the rolled back period, may also cause problem. Their sending processes will have no idea that these messages are to be sent again. Such messages, whose send events are recorded in the state of the sender process but the receive events are lost, are called *missing messages*. A set of checkpoints, with one checkpoint for every process, is said to be *Consistent Global checkpointing State (CGS)*, if it does not contain any *orphan message* or *missing message*. However, generation of missing messages may be acceptable, if messages are logged by sender. In a distributed system, each process has to take checkpoints periodically on non-volatile stable storage. In case of a failure, the system rolls back to a consistent set of checkpoints. If all the processes take checkpoints at the same time instant, the set of checkpoints would be consistent. But since globally synchronized clocks are very difficult to implement, processes may take checkpoints within an interval. In order to achieve synchronization, sometimes processes take temporary checkpoints. When all processes agree, these checkpoints are made permanent. The focus of this paper is on checkpointing techniques, which do not require any intervention on the part of the application or the programmer. The system automatically takes checkpoints according to some specified policy, and recovers automatically after the failed process restarts.
This approach has the advantages of relieving the application programmers from the complex and error-prone chores of implementing fault tolerance and of offering fault tolerance to existing applications written without consideration of reliability concerns.

2. RELATED WORK
Prakash - Singhal proposed first non-blocking minimum process checkpointing protocol for MDCSs which tries to minimizing the number of number of processes participating without blocking, that algorithm may produce inconsistencies in some situations, and they further proves that there does not exists a minimum-process non-blocking checkpointing algorithms in reality.

Cao-Singhal showed that algorithm proposed causes inconsistencies when there are multiple forced checkpoints. The correctness proof fails to handle the situation when the sending process P takes multiple forced checkpoints as only the latest one’s initiator information is attached with the computation message sent to process Q.

Cao and Singhal achieved non-intrusiveness in minimum-process algorithm by introducing the concept of mutable checkpoints to adapt to mobile environments. Proposed protocol is modified version of the algorithm proposed in mutable checkpoints that need not be saved on the stable storage and can be store on the main memory of the MHs and has not any transferring cost.

3. SYTEM MODEL
The system model of the distributed mobile system used is as follows:
(i)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.
(ii)The mobile distributed system can be considered as consisting of “n” Mobile Hosts (MHs) and “m” Mobile Support Stations (MSSs). All the MSSs are connected through static wired network. A cell is a small geographical area around the MSS supports a MH only within this are and there is a wireless link between a MH to MSS. A MH can

communicate to another MH only through their reachable MSS.
(iii)The processes do not share memory or clock and message passing is the only way for processes to communicate with each other.
(iv)Each process progresses at its own speed and messages are exchanged through reliable channels, whose transmission delays are finite but arbitrary.
(v)The communication links are FIFO.
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. We also assume that the processes are non-deterministic.

4. PROPOSED CHECKPOINTING PROTOCOL

This research paper proposes a new checkpointing protocol for mobile environments. Our synchronous minimum process checkpointing algorithm has the following characteristics:
(i)It maintains the correct dependencies among processes.
(ii)It piggybacks the necessary information onto normal messages and ignores the non-existent dependencies;
(iii)It also maintains the information related to location of MHs, dependency and forced/permanent record on the MSS level for reducing the searching cost
(iv)To reduce the useless checkpoint, the trigger tuples are piggybacked with the application message sent after taking the forced checkpoint.
(v)It also maintains the record of multiple forced checkpoints (FC) after converting the forced checkpoint into permanent one.
(vi)At the time of starting a checkpointing algorithm, a predefined checkpoint period T is set on all the process. All process in the system take their permanent checkpoints (PC) after this time period T; however a process may not take PC if it takes forced checkpoint (FC) before the permanent checkpoint.
(vii)As there is no common clock and processes do not share a common memory but every MH and MSS contains a system clock, with typical clock drift rate p in the order of 10^{-5} to 10^{-6}. The system clocks of MSSs can be synchronized using the network time protocol (NTP). Due to the reliability of MSSs, and work as mediator of MHs, time in MSSs is used to as reference. The MSS closest to the receiver, is timer
(viii)A process involves in global state, sends the acknowledgement directly to the initiator.
(ix)In our proposed checkpointing approach, checkpoints are taken only if any event occurs between the timeout period to back up copy of previous checkpointed state are still valid as shown in Figure 1

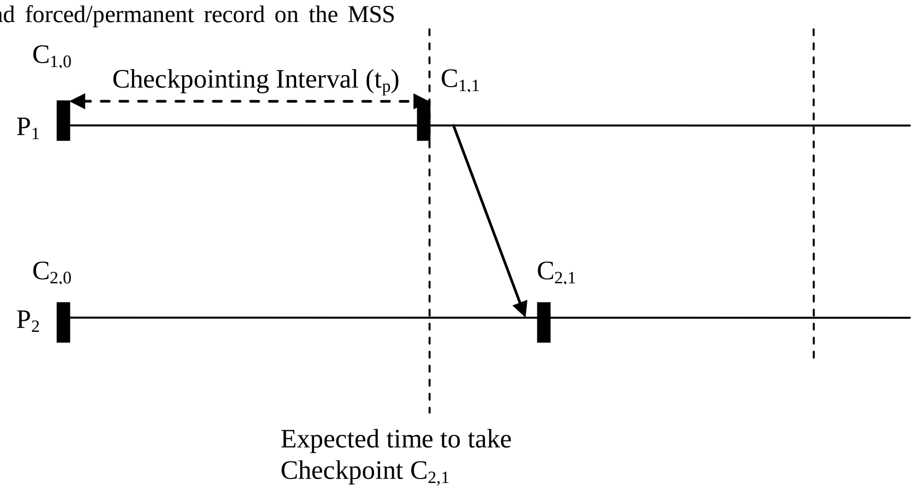
$C_{1,0}$
Checkpointing Interval (t_p)
$C_{1,1}$
P_1
$C_{2,0}$
$C_{2,1}$
P_2
Expected time to take Checkpoint $C_{2,1}$

Figure 1 Taking Checkpoints

Process Pi is responsible for piggybacking its local time, in every application message. On receiving message from process Pi:

```
    If (current_time () > get_time ()
        Resetimer (get_time)
```

As shown in figure 1 there are two processes P1, P2 here expected time to take checkpoint > checkpoint interval

We explain our checkpointing algorithm with the help of an example. Consider the distributed system as shown in Figure 2. Note that when a computation message is sent after taking the checkpoint it piggybacked with minset []. The entire computation message is piggybacked with the csn and dependency vector. Assume that process P_4 initiate checkpointing process in Figure 2. First process P_4 takes its checkpoint and increment its csn number from $C_{4,0}$ to $C_{4,1}$, compute minset[](which in case of Figure 2 is {P_1, P_3, P_5}). This means that the initiator process is directly or transitively dependent on these processes. Hence, when P_2 initiate a checkpoint all of these processes should take their checkpoints in order to maintain global consistent state. Therefore P_2 sends the checkpoint request along with minset [] to process P_1, P_3 and P_5. When P_3 receives the checkpoint request it takes the tentative checkpoint and sends message M4 by attaching minset [1011100], trigger set(P_4, $C_{4,1}$), and csn_3=1. After receiving message M4, P_2 first compare m.csn_3 (which is 1) with its old_csn_2 [3] (which is 0). As P_2 does not belongs to minset, not sent any message to the processes which are in minimum set and m.csn_3 > csn_2 [3]. Hence, P_2 takes forced checkpoint, update its trigger set to (P_4, $C_{4,1}$), increment its csn_2 from 1 to 2, and updates the csn_2[3] from $C_{4,0}$ to $C_{4,1}$.

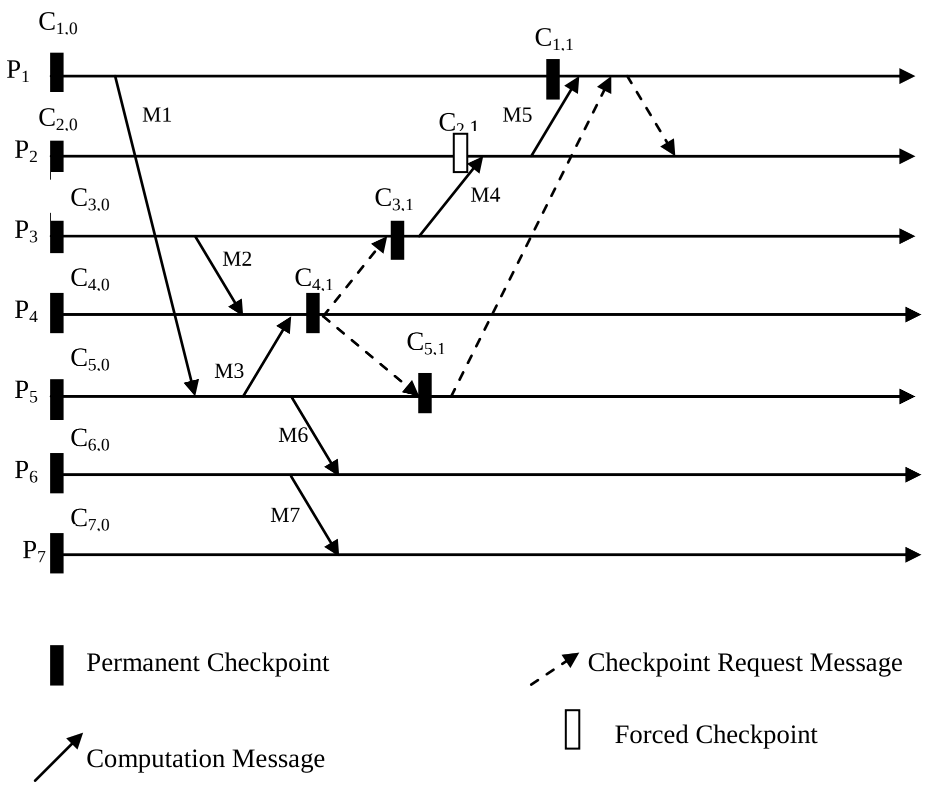

Figure 2. An example of our checkpointing approach

After taking forced checkpoint it sends message M5 to P_1. P_1 takes tentative checkpoint directly due to minset $[P_1]$= =1 and set c_state ==1(as P_1 knows that it is the part of minset and get the checkpoint request from the initiator in future and when it get the checkpoint request it ignore the request).P_2 check its dependency and find out that it receives computation message from P_2 since its last checkpoints.

So, it sends checkpoint request to the process P1 with weight and reply with remaining weight and new_ddv_2 [P1] ==1 to the initiator. After receiving the checkpointing request from P_1, P_2 converts its forced checkpoints in to tentative one and reply to the initiator. Initiator compute the Uminset [P_1, P_2, P_3, P_4, P_5] by taking the union of minset {P1, P3, P4, P5} and new_ddv_1 {P_2}.

At last, when P_2 receives positive responses from all relevant processes(weight = =1) it issues commit request along with the exact minimum set [P_0, P_1, P_2, P_3, P_4] to all processes. On receiving commit following actions are taken. A process, in the minimum set, converts its tentative checkpoint into permanent one and discards its earlier permanent checkpoint, if any.

On the other hand if it receive the negative response from any one of the processes which belongs to the minset, it sends the abort message to all processes which belongs to Uminset []. On receiving abort, processes discard the tentative checkpoint, if any; reset c_state, tentative, g_chkpt etc and update ddv [] and minset [].The system is consistent.

Performance Analysis:

In this section we analyze our checkpointing algorithm by comparing with different existing algorithms in different context. We use following notations to compare our algorithm with others.

C_{air}: Cost of sending a message m from any P_i to P_j;
C_{broad}: Cost of broadcasting a message to all processes;
N_{min}: Number of processes that belongs to minset;
n: total number of MHs in the system;
n_{dep}: Average no. of processes on which a process depends;
$T_{ch:}$ Total time taken to store the checkpoint on stable storage

Figure shows the average searching time on y-axis, no. of MHs on the X-axis for algorithms proposed in [9] and our proposed protocol. As depicted figure 4, with the number of MHs increases, our checkpointing protocol have approximately two times less searching cost in the comparison of [9] and changes very slightly on increasing the number of MHs..

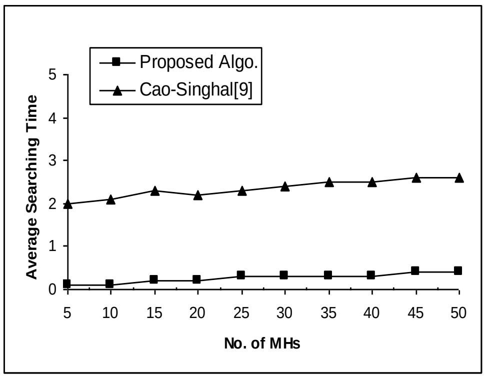

Figure 3. Average searching time comparison

5. CONCLUSION

This research paper presents an efficient minimum process synchronous checkpointing algorithm which fit into the mobile environment. It forces only those minimum number of MHs which are directly or transitively dependent, to take their

checkpoints during checkpointing. This helps in power saving as if a MH is in active mode it consumes more power.
In our algorithm MHs take very less number of checkpoints which is nearest to minimum. Reduction in the number of checkpoints helps in the efficient use of resources of mobile systems as message sending and state saving consumes power and bandwidth. It has very less searching cost. It requires very minimum interaction between initiator MH and others. Instead of above all, our protocol is non-blocking, adaptive, uses very simple data structure and each MH takes its checkpointing decision on their own basic. These all features make our algorithm more suitable for mobile computing environment.

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] 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.

[3] 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.

[4] 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.

[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] 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] G.H. Forman and Zahorjan, "The Challenges of Mobile Computing" Computer, Vol. 27, no.4, pp.38-47, Apr. 1994.

[8] Singhal, M. Shivaratri, N.-G.: Advanced Concept in Operating System. McGraw Hill,(1994)

[9] 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.

[10] Kshemkalyani Ajay D, Singhal, M.: Distributed Computing Principles, Algorithms, and System (2008)