Volume 5, Issue 7, July 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Piggybacking based Clock Synchronous Approach for Determining Global State in Mobile Distributed Systems

Raman Kumar
Research Scholar, Deptt. of CSE,
Mewar University, Chittorgarh (Raj.), India

Parveen Kumar
Deptt. of Computer Science & Engg.
NIT, Hamirpur (H.P), India

Abstract— This paper presents an efficient piggybacking based clock synchronization approach which is probabilistic and can guarantee a much smaller bound on the clock skew than most existing algorithms. The proposed algorithm is designed to impose low memory and computation overheads on MHs and low communication overheads on wireless channels.

Keywords— Clock synchronization; Checkpointing; Mobile Distributed Systems, Fault Tolerance

I. INTRODUCTION

Checkpointing/rollback recovery is an attractive and popular technique which gives fault tolerance without additional efforts in DSs [1]. A checkpoint is a global state of a process stored on stable storage. In a DS, since the processes in the system do not share memory and have not any synchronized clock, a global state of the system is defined as a set of LSs, one from each process. A global state is said to be "consistent" if it contains no orphan message; i.e., a message whose receive event is recorded, but its send event is lost. To recover from failure, the system restarts its execution from a previous CGS saved on the stable storage during fault-free execution [Figure 1]. Checkpointing algorithms for DSs have been extensively studied in the literature (e.g., [2], [3], [4], [5], [6], [7], [8]). Due to the emerging challenges of the MDS as low bandwidth, mobility, lack of stable storage, frequent disconnections and limited battery life, the fault tolerance technique designed for distributed system cannot directly implemented on mobile distributed systems(MDSs).

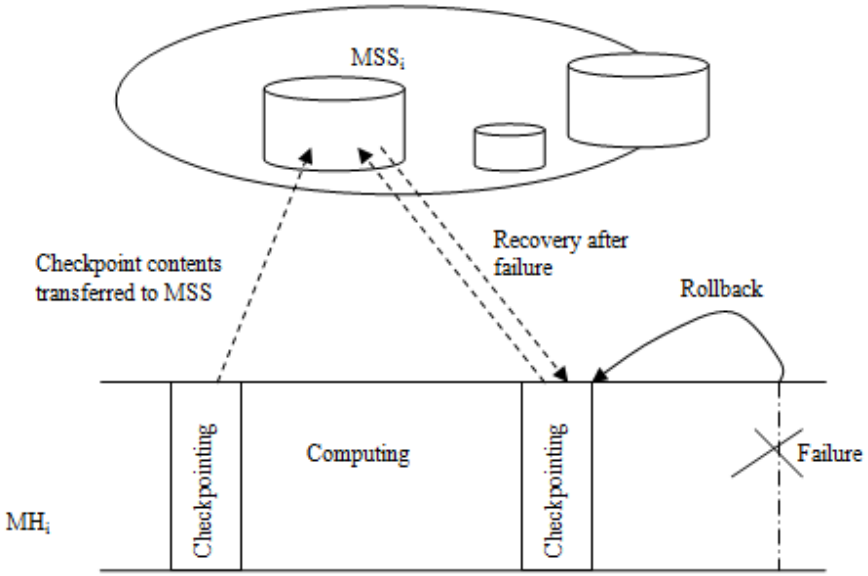

Fig. 1 Checkpointing and rollback recovery in MDS

A common goal of checkpointing algorithm for MDSs is to reduce the checkpointing cost by taking minimum number of checkpoints and reducing the coordinated messages. Hence, the checkpointing algorithms having lesser number of coordinated messages and fewer checkpoints nearly to minimum are preferred for mobile environment.

II. PROPOSED CLOCK SYNCHRONIZATION MECHANISM

A. Piggybacking:

As shown in Figure 2, MSS is known as the primary server and its timer are used as a reference because it has more reliable timer than those in MHs. In our approach three types of messages are transferred between MH and MSS; computation message, acknowledgement message and checkpoint reply message. Every application and acknowledgement message is piggybacked with the local clock time of the MSS which is known as CLK_M, and that is, time to next checkpoint.

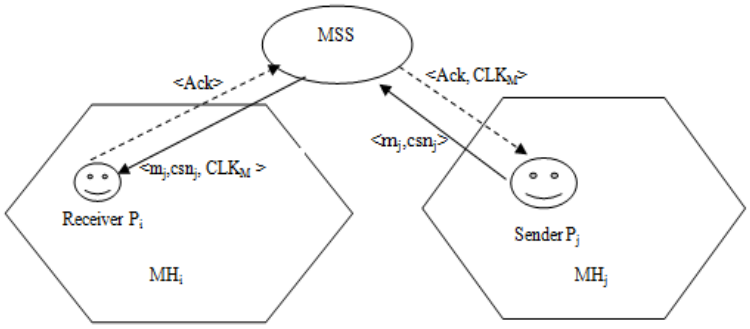

Fig. 2 Message passing between MHs

When a process receives the timer with the message, it compares its local time with the one just received. If there are differences in both the timer (received and own), it resets it with the received value. After receiving the piggybacked message, MHs resets its time on the basic of $CLK_{M.}$ We will explains by using an example that how processes are resynchronized in Figure 3.Distributed system uses in Figure 3 uses three processes P_1, P_2, P_3 where simple message communication are displayed and acknowledgement are not represented. As the time of all the process are not perfectly synchronized, so each process takes their checkpoints at different time instants.

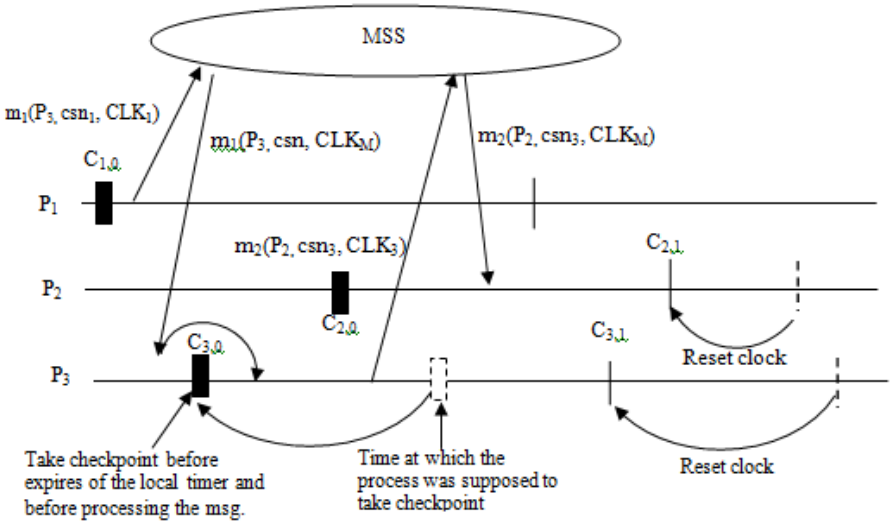

Fig. 3 Clocks resynchronization through piggybacked messages

All the messages are communicated through each other message passing through their local MSS. When process P_1 sends message m_1 to process P_3, it first sends it to its local MSS with its own csn, and then MSS transferred the message piggybacks with its own CLK_M to destination process P_3. When m_1 arrives, process P_3 does not take any checkpoint and still in $C_{2,0}$ -1^{th} CI. If m_1 is not properly handled it can causes inconsistency and become orphan message. To remove inconsistency, P_3 first takes its checkpoint and then process the message m_1. P_3 also resets the clock for the next checkpoint. Message m_2 is a normal message that indirectly resynchronizes the timer of process $P_{2.}$

B. Multicasting:

MSS may also update its subsidiary MHs by using multicasting [Figure 4]. In multicasting, MSS periodically multicasts the time to all Base Station (BS) under it and further BS updates the time of all its MHs which are running under it. This mode can achieve only relatively low accuracies, but ones that nonetheless are considered sufficient for many purposes.

In both modes, messages are delivered unreliably, using the standard internet algorithm. First mode generates fewer overheads compared to the second one. In our approach we use the second approach to synchronization of clocks.

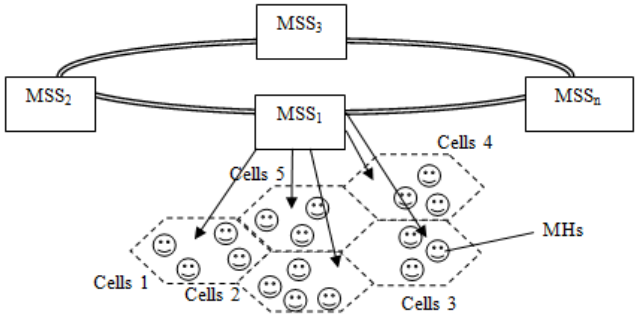

Figure 4 Clocks synchronization using multicasting

III. IMPLEMENTATION DETAILS

Proposed algorithms is similar to [9], but main disadvantages of the approach is that a global consistent state consists all the Nth checkpoints of every process, where N≥0. Proposed approach does not awakes to all the MHs; rather only those minimum numbers of MHs that have sent or receives some messages after their last checkpoint will takes checkpoints during checkpointing.

A. Initial state

All the process in the system takes their periodically independent from the other processes. Before a checkpointing process starts, a predefined checkpoint period T is set by the local MSS, on the timer of each mobile hosts (MHs) running under it.

However, due the varying drift rates of local clock and initial timer inaccuracy the timer at different MHs are not perfectly synchronized. Hence, the checkpoint may not be consistent because of orphan message and there is a great need to resynchronize the local clock. Here we resynchronize the initial or predefined value of T by using the resynchronization mechanism. When the local time of the process expires and if it sends any computation message during this current CI, then it takes checkpoint without the coordination of the other processes, increments in checkpoint sequence number (csn) and reply to its local MSS. On the other hands, if no message has been sent in the current CI then it will takes soft-checkpoint for the current CI and sends reply. Soft checkpoint only shows that process does not send any message during its current CI and its current CI has finished. It has not any transferring cost and stored locally on MHs and discarded when processes takes its checkpoint during next CI. When the MSS knows that it receives checkpoint reply from all the process in minimum set, it removes the previous global state and set new GS or consistent set.

B. Selection of checkpoint interval

Checkpoint interval (CI) is the duration of the time between completion of a checkpoint and start of the next checkpoint i.e. in Figure 5 (a) CI is equal to t_2-t_1. The CI plays an important role in checkpointing process. If CI is too small [Figure 5 (b)] then it has a high checkpointing cost and increases the execution time, as checkpoints takes time to stored on the permanent storage and also during recovery most of the checkpoints are not used for recovery. On the other hand, if CI is too large [Figure 5 (c)] then there may be large time interval between failure and recently saved checkpoint which leads to large amount of computational loss during rollback and recovery. Hence, selection of optimal CI can minimize the checkpoint overheads and play important role in reducing overheads incurred due to checkpointing and rollback recovery.

As checkpoints are stored on a stable storage and required time to save it which is called checkpoint latency. Thus, CI should be greater then the checkpoint latency. Proposed algorithm adapts its behavior with the characteristics of the environment. In starting, it will adapt on the basic of quality of service of network. If the network has poor quality of service, the algorithm takes checkpoint frequently else infrequently. The checkpoint period depends upon the quality of service and failure rate of the system. Further, after some time of processing it will learn from the environment and takes decision on the basic of failure rate, average overheads per checkpoints, average latency per checkpoint, and average recovery time as in [10].

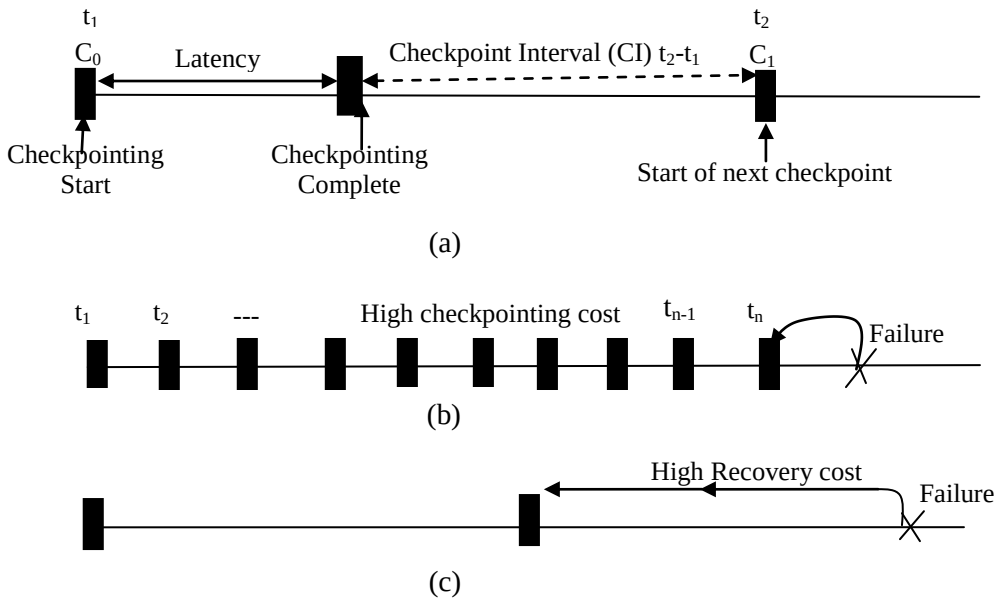

Fig. 5 (a) Checkpoint interval (CI) (b) Frequent checkpoints (c) Infrequent checkpoints

C. Maintaining minimum set

For maintaining the minimum set in proposed checkpointing approach, Each MSS maintain the following information's which are shown in Table 1:

TABLE 1 TABLE MAINTAINED AT EACH MSS

Msg_recvdfrom (1)	**Msg_sent_to (2)**	**Msg_status (3)**	**Ack_status (4)**	**Chkrply_recvd_from (5)**
Name of the source process from which msg received)	Name of the destination process to whom recvd msg to be sent	0- msg sent to destination, 1-Ack for sent-msg received 2- msg sent during previous CI	0- Ack not sent 1-Ack sent 2- recvd during previous CI	Name of the process from which it received checkpoint or soft checkpoint reply.

Minset [] = (msg_recevdfrom) U (msg_sent_to); Hence, in our approach minset[] is computed by taking the union of column (1) and (2) which contains all the processes from which it received computation message or sent any message, received from the other process.

D. Maintaining consistent global state

As memory of MHs is not considered stable, when processes take checkpoints, they transferred their checkpoint information including with reply, to its local MSS, where it is stored on the stable storage of MSS. If process does not send any message during its current CI and checkpoint interval expires processes takes soft checkpoint and reply to its local MSS. if MSS receives the reply from all the processes in minset, then a global consistent state is updated. On the other hands if a processes is belongs to minset and does not reply; it sends the reminder and forces it to takes it's soft or permanent checkpoint, and wait for reply.

IV. WORKING MODEL

A process Pi takes its checkpoint on two ways:

a) On the expires of its local timer CLKi : if its timer has expires, then it checks the status of ci. if ci=1, then it takes a permanent checkpoint, in another case if ci=0, it takes soft checkpoint(SC).
b) On the receipt of piggybacked message in between current CI: in such case it will checks and compares the received csn with its own csni. There may be following two possibilities:

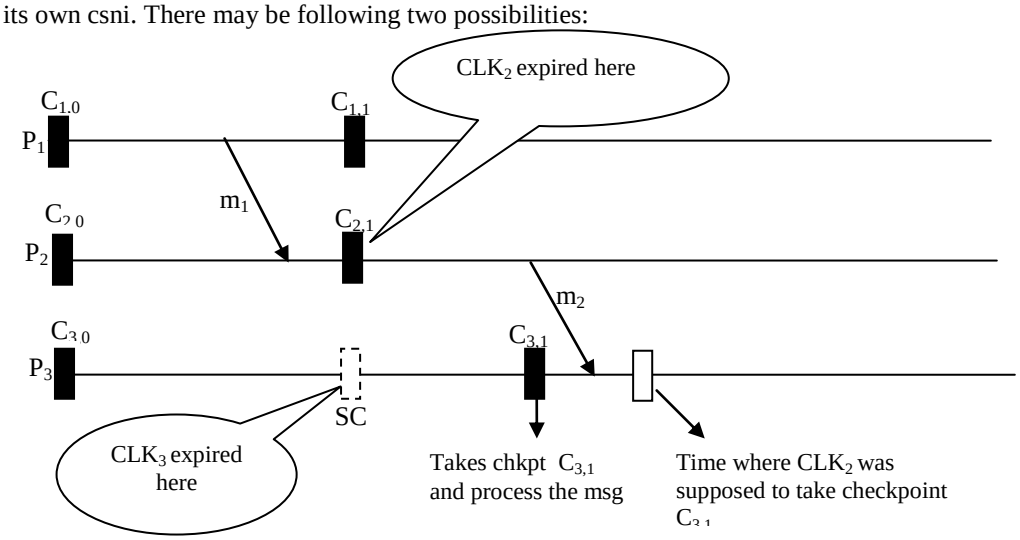

Fig. 6 Working model of proposed approach

msg_csn < = own_csn$_i$: if it is true, process P_i receives message as a normal without taking any checkpoint. In Fig. 6, csn of process P_1 received with message m_1 is equal to the csn_2. Hence, checkpoint is not taken after receiving the message. However process P_2 takes a checkpoint, after expires of its current CI it sends any message.

msg_csn > own_csn$_i$: it is true, process P_i takes checkpoint first and then precedes the message. If process P_i taken any soft checkpoint, it will discards it and set the checkpoint state (CS) is equal to 1. Furthermore P_i does not takes checkpoint, in the current CI, after expires of its local clock CLK_i, if it does not send any message after taking the checkpoint and before expires of the clocks. In Figure 6, process P_3 takes a checkpoint before receiving the message m_2 as csn received with m_3 is greater than its own csn. Process P_3 receives the message after taking the checkpoint $C_{3,1}$ and it does not takes checkpoint after expires of its local clock.

V. CONCLUSION

Checkpointing approach having lesser number of coordinated messages and lesser number of checkpoints are preferred in mobile distributed systems. Proposed ppiggybacking based clock synchronization algorithm doesn't require any extra message as it uses timer to indirectly synchronization for determining consistent global state. This proposed approach can be implemented and compared with other approaches for a future research work.

REFERENCES

[1] Candy K.M. and Lamport L., "*Distributed Snapshots: Determining Global State of Distributed Systems*", ACM Trans. on Computing Systems, Vol. 3, No. 1,pp. 63-75, Feb.1985.

[2] Koo R. and Toueg S., "*Checkpointing and Roll-Back Recovery for Distributed Systems*", IEEE Trans. on Software Engg., Vol.13, No.1, pp.23-31, Jan. 1987.

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

[4] Elnozahy E.N., Johson D.B. and Zwaenepoel W., "*The Performance of Consistent Checkpointing*", in the Proc. of the 11th Symp. on Reliable Distributed Systems, pp. 39-47, Oct. 1992.

[5] Johnson, D.B., Zwaenepoel, W., "*Sender-based message logging*", In the Proc. of the 17th Int'l Symp. on Fault-Tolerant Computing, pp. 14-19, 1987.

[6] Neves N. and Fuchs W. K., "*Adaptive Recovery for Mobile Environments*", ACM Communication, Vol. 40, No.1, pp. 68-74, January 1997.

[7] Silva L.M. and Silva J.G., "*Global checkpointing for distributed programs*", in the Proc. of the 11th Symp. Reliable Distributed Systems, pp. 155-62, Oct. 1992.

[8] Sistla A.P. and Welch J.L., "*Efficient Distributed Recovery Using Message Logging*", in the Proc. of the 18th Symp. on Principles of Distributed Computing", pp. 223-238, Aug. 1989.

[9] Singh A.K., "*On Mobile Checkpointing using Index and Time Together*", World Acdemy of Science, Engineering and Technology, Vol 32, pp. 144-151, 2007.

[10] Vaidya N.H., "*Impact of Checkpoint Latency on Overhead Ratio of a Checkpointing Scheme*", IEEE Trans. Computers, Vol. 46, No. 8, pp. 942-947, August 1997.

