academicresearchJournals

Vol. 4(5), pp. 158-162, November 2016
DOI: 10.14662/IJARER2016.026

ISSN: 2360-7866
http://www.academicresearchjournals.org/IJARER/Index.htm

International Journal of Academic Research in Education and Review

Full Length Research

Handling Recurrent Failures in Coordinated Checkpointing for Mobile Distributed Systems

Maridul Kothari and Parveen Kumar

Department of Computer Science, Nims University, Jaipur

Accepted 27 November 2016

We propose a minimum-process coordinated checkpointing algorithm for non-deterministic mobile distributed systems, where no useless checkpoints are taken. An effort has been made to minimize the blocking of processes and synchronization message overhead. We capture the partial transitive dependencies during the normal execution by piggybacking dependency vectors onto computation messages. Frequent aborts of checkpointing procedure may happen in mobile systems due to exhausted battery, non-voluntary disconnections of MHs, or poor wireless connectivity. Therefore, we propose that in the first phase, all concerned MHs will take mutable checkpoint only. Ad hoc checkpoint is stored on the memory of MH only. In this case, if some process fails to take checkpoint in the first phase, then MHs need to abort their ad hoc checkpoints only. In this way, we try to minimize the loss of checkpointing effort when any process fails to take its checkpoint in coordination with others.

Keyword: Fault Tolerance, Mobile Computing Systems, Coordinated Checkpointing, Rollback Recovery, Distributed Systems.

Cite This Article As: Kothari M, Kumar P (2016). Handling Recurrent Failures in Coordinated Checkpointing for Mobile Distributed Systems. Inter. J. Acad. Res. Educ. Rev. 4(5): 158-162

INTRODUCTION

A distributed system is a collection of independent entities that cooperate to solve a problem that cannot be individually solved. A mobile computing system is a distributed system where some of processes are running on mobile hosts (MHs), whose location in the network changes with time. 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. In this chapter, we propose a minimum-process coordinated checkpointing algorithm for non-deterministic mobile distributed systems, where no useless checkpoints are taken. An effort has been made to minimize the blocking of processes and synchronization message overhead. We capture the partial transitive dependencies during the normal execution by piggybacking dependency vectors onto computation messages. Frequent aborts of checkpointing procedure may happen in mobile systems due to exhausted battery, non-voluntary disconnections of MHs, or poor wireless connectivity. Therefore, we propose that in the first phase, all concerned MHs will take mutable checkpoint only. Ad hoc checkpoint is stored on the memory of MH only. In this case, if some process fails to take checkpoint in the first phase, then MHs need to abort their ad hoc checkpoints only. In this way, we try to

minimize the loss of checkpointing effort when any process fails to take its checkpoint in coordination with others.

Most of the existing coordinated checkpointing algorithms [37, 63] rely on the two-phase protocol and save two kinds of checkpoints on the stable storage: tentative and permanent. In the first phase, the initiator process takes a tentative checkpoint and requests all or selective processes to take their tentative checkpoints. If all processes are asked to take their checkpoints, it is called all-process coordinated checkpointing [3]. Alternatively, if selective communicating processes are required to take checkpoints, it is called minimum-process checkpointing. Each process informs the initiator whether it succeeded in taking a tentative checkpoint. After the initiator has received positive acknowledgments from all relevant processes, the algorithm enters the second phase. Alternatively, if a process fails to take its tentative checkpoint in the first phase, the initiator process requests all or concerned processes to abort their tentative checkpoint.

If the initiator learns that all concerned processes have successfully taken their tentative checkpoints, the algorithm enters in the second phase and the initiator asks the relevant processes to make their tentative checkpoints permanent.

In order to record a consistent global checkpoint, when a process takes a checkpoint, it asks (by sending checkpoint requests to) all relevant processes to take checkpoints. Therefore, coordinated checkpointing suffers from high overhead associated with the checkpointing process [11, 12]. Much of the previous work [4, 5, 6] in coordinated checkpointing has focused on minimizing the number of synchronization messages and the number of checkpoints during the checkpointing process. However, some algorithms (called blocking algorithm) force all relevant processes in the system to block their computations during the checkpointing process. Checkpointing includes the time to trace the dependency tree and to save the states of processes on the stable storage, which may be long. Moreover, in mobile computing systems, due to the mobility of MHs, a message may be routed several times before reaching its destination. Therefore, blocking algorithms may dramatically reduce the performance of these systems [31]. Recently, non-blocking algorithms have received considerable attention. In these algorithms, processes need not block during the checkpointing by using a checkpointing sequence number to identify orphan messages. Moreover, these algorithms require all processes in the system to take checkpoints during checkpointing, even though many of them may not be necessary.

System Model

Our system model consists of a number of MHs which communicate through mobility support stations (MSSs). Each MSS is a fixed network host which provides wireless communication support for a fixed geographical area, called a cell. MSSs are linked together over the wired data networks. The distributed system consisting of n processes, running on MHs or MSSs. The MHs can communicate with the MSS through wireless channels. We assume that wireless channels and logical channels are all FIFO order. If a MH moves to the cell of another base station, a wireless channel to the old MSS is disconnected and a wireless channel in the new MSS is allocated. However, its checkpoint related information is still with the old MSS. A MH may voluntarily disconnect from mobile computing networks. The MH does not send and receive any message when it is in a disconnected state. We also assume a closed system that consists of nodes, links, and disks. Input is stored on disk before operation begins. Output is stored on disk when the job ends.

There is no common clock, shared memory or central coordinator. Message passing is the only mode of communication between any pair of processes. The messages originated from a source Mh, are received by the local Mobile support stations and then forwarded to the destination MH. Any process can initiate checkpointing. It is assumed that processes may be failed during processing but there is no communication link failure. Messages are exchanged with finite but arbitrary delays. In our algorithm, we consider that the processes which are running in the distributed mobile systems are non-deterministic.

Basic Idea

All Communications to and from MH pass through its local MSS. The MSS maintains the dependency information of the MHs which are in its cell. The dependency information is kept in Boolean vector R_i for process P_i. The vector has n bits for n processes. When $R_i[j]$ is set to 1, it represents P_i depends upon P_j. For every P_i, R_i is initialized to 0 except $R_i[i]$, which is initialized to l. When a process P_i running on an MH, say MH_p, receives a message from a process P_j, MH_p's local MSS should set $R_i[j]$ to 1.If P_j has taken its permanent checkpoint after sending m, $R_i[j]$ is not updated.

Suppose there are processes P_i and P_j running on MHs, MH_i and MH_j with dependency vectors R_i and R_j. The dependency vectors of MHs, MH_i and MH_j are maintained by their local MSSs, MSS_i and MSS_j. Process P_i running on MH_i sends message m to process P_j running on MH_j. The message is first sent to MSS_i (local MSS of MH_i). MSS_i maintains the dependency vector R_i of MH_i. MSS_i appends R_i with message m and sends it to MSS_j (local MSS of MH_j). MSS_j maintains the dependency vector R_j of MH_j. MSS_j replaces R_j with bitwise logical OR of dependency vectors R_i and R_j and sends m to P_j.

In Figure 1, there are five processes P_1, P_2, P_3, P_4, P_5

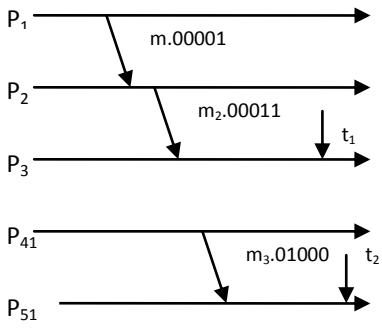

Figure 1. Maintenance of Dependency Vectors

with dependency vectors R_1, R_2, R_3, R_4, R_5 initialized to 00001, 00010, 00100, 01000, and 10000 respectively. Initially, every process depends upon itself. Now process P_1 sends m to P_2. P_1 appends R_1 with m. P_2 replaces R_2 with the bitwise logical OR of R_1(00001) and R_2(00010), which comes out to be (00011). Now P_2 sends m2 to P_3 and appends R_2 (00011) with m_2. Before receiving m_2, the value of R_3 at P_3 was 00100. After receiving m_2, P_3 replaces R_3 with the bitwise logical OR of R_2 (00011) and R_3 (00100) and R_3 becomes (00111). Now P_4 sends m_3 along with R_4 (01000) to P_5. After receiving m_3, R_5 becomes (11000).In this case, if P_3 starts checkpointing at t_1, it will compute the tentative minimum set equivalent to R_3(00111), which comes out to be {P_1, P_2, P_3}. In this way, partial transitive dependencies are captured during normal computations.

In coordinated checkpointing, if a single process fails to take its checkpoint; all the checkpointing effort goes waste, because, each process has to abort its tentative checkpoint [28, 62, 72, 77, 90, 92]. Furthermore, in order to take the tentative checkpoint, an MH needs to transfer large checkpoint data to its local MSS over wireless channels. Hence, the loss of checkpointing effort may be exceedingly high due to frequent aborts of checkpointing algorithms especially in mobile systems. In mobile distributed systems, there remain certain issues like: abrupt disconnection, exhausted battery power, or failure in wireless bandwidth. So there remains a good probability that some MH may fail to take its checkpoint in coordination with others. Therefore, we propose that in the first phase, all processes in the minimum set, take ad hoc checkpoint only. Ad hoc checkpoint is stored on the memory of MH only. If some process fails to take its checkpoint in the first phase, then other MHs need to abort their ad hoc checkpoints only. The effort of taking an ad hoc checkpoint is negligible as compared to the tentative one. In other protocols [3, 4], all concerned processes need to abort their tentative checkpoints in this situation. Hence the loss of checkpointing effort in case of an abort of the checkpointing procedure is dramatically low in the proposed scheme as compared to other coordinated checkpointing schemes for mobile distributed systems [5, 6].

In this second phase, a process converts its ad hoc checkpoint into tentative one. By using this scheme, we try to minimize the loss of checkpointing effort in case of abort of checkpointing algorithm in the first phase.

A non-blocking checkpointing algorithm does not require any process to suspend its underlying computation. When processes do not suspend their computation, it is possible for a process to receive a computation message from another process, which is already running in a new checkpointing interval. If this situation is not properly dealt with, it may result in an inconsistency. During the checkpointing procedure, a process P_i may receive m from P_j such that P_j has taken its checkpoint for the current initiation whereas P_i has not. Suppose, P_i processes m, and it receives checkpoint request later on, and then it takes its checkpoint. In that case, m will become orphan in the recorded global state. We propose that only those messages, which can become orphan, should be buffered at the sender's end. When a process takes its ad hoc checkpoint, it is not allowed to send any message till it receives the tentative checkpoint request. However, in this duration, the process is allowed to perform its normal computations and receive the messages. When a process receives the tentative checkpoint request, it is confirmed that every concerned process has taken its ad hoc checkpoint. Hence, a message generated for sending by a process after getting tentative checkpoint request cannot

become orphan. Hence, a process can send the buffered messages after getting the tentative checkpoint request from the initiator.

The Proposed Algorithm

First phase of the algorithm:

When a process, say P_i, running on an MH, say MH_i, initiates a checkpointing, it sends a checkpoint initiation request to its local MSS, which will be the proxy MSS (if the initiator runs on an MSS, then the MSS is the proxy MSS). The proxy MSS maintains the dependency vector of P_i say R_i. On the basis of R_i, the set of dependent processes of P_i is formed, say S_{minset}. The proxy MSS broadcasts ckpt (S_{minset}) to all MSSs. When an MSS receive ckpt (S_{minset}) message, it checks, if any processes in S_{minset} are in its cell. If so, the MSS sends ad hoc checkpoint request message to them. Any process receiving a ad hoc checkpoint request takes a ad hoc checkpoint and sends a response to its local MSS. After an MSS received all response messages from the processes to which it sent ad hoc checkpoint request messages, it sends a response to the proxy MSS. It should be noted that in the first phase, all processes take the ad hoc checkpoints. For a process running on a static host, ad hoc checkpoint is equivalent to tentative checkpoint. But, for an MH, ad hoc checkpoint is different from tentative checkpoint. In order to take a tentative checkpoint, an MH has to record its local state and has to transfer it to its local MSS. But, the ad hoc checkpoint is stored on the local disk of the MH. It should be noted that the effort of taking a ad hoc checkpoint is very small as compared to the tentative one. For a disconnected MH that is a member of minimum set, the MSS that has its disconnected checkpoint, considers its disconnected checkpoint as the required come.

Second Phase of the Algorithm:

After the proxy MSS has received the response from every MSS, the algorithm enters the second phase. If the proxy MSS learns that all relevant processes have taken their ad hoc checkpoints successfully, it asks them to convert their ad hoc checkpoints into tentative ones and also sends the exact minimum set along with this request. Alternatively, if initiator MSS comes to know that some process has failed to take its checkpoint in the first phase, it issues abort request to all MSS. In this way the MHs need to abort only the ad hoc checkpoints, and not the tentative ones. In this way we try to reduce the loss of checkpointing effort in case of abort of checkpointing algorithm in first phase.
When an MSS receives the tentative checkpoint request, it asks all the process in the minimum set, which are also running in itself, to convert their ad hoc checkpoints into tentative ones. When an MSS learns that all relevant process in its cell have taken their tentative checkpoints successfully, it sends response to proxy MSS. If any MH fails to transfer its checkpoint data to its local MSS, then the failure response is sent to the proxy MSS; which in turn, issues the abort message.

Third Phase of the Algorithm:

Finally, when the proxy MSS learns that all processes in the minimum set have taken their tentative checkpoints successfully, it issues commit request to all MSSs. When a process in the minimum set gets the commit request, it converts its tentative checkpoint into permanent one and discards its earlier permanent checkpoint, if any.

Massage Handling During Checkpointing:

When a process takes its ad hoc checkpoint, it does not send any massage till it receives the tentative checkpoint request. This time duration of a process is called its uncertainty period. Suppose, P_i sends m to P_j after taking its ad hoc checkpoint and P_j has not taken its ad hoc checkpoint at the time of receiving m. In this case, if P_j takes its ad hoc checkpoint after processing m, then m will become orphan. Therefore, we do not allow Pi to send any massage unless and until every process in the minimum set have taken its ad hoc checkpoint in the first phase. P_i can send massages when it receives the tentative checkpoint request; because, at this moment every concerned process has taken its ad hoc checkpoint and m cannot become orphan. The massages to be sent are buffered at senders end. In this duration, a process is allowed to continue its normal computations and receive massages.

CONCLUSION

In this paper, we have proposed a minimum-process checkpointing protocol for non-deterministic mobile distributed systems, where no useless checkpoints are taken and an effort has been made to minimize the blocking of processes. We try to reduce the checkpointing time and blocking time of processes by limiting checkpointing tree which may be formed in other algorithms [28, 37]. We captured the transitive dependencies during the normal execution by piggybacking dependency vectors onto computation messages. The Z-dependencies are well taken care of in this protocol. We also try to reduce the loss of checkpointing effort when any process fails to take its checkpoint in coordination with others.

REFERENCES

[1] Chandy K.M. and Lamport L., “Distributed snapshots : Determining Global State of Distribited Systems, “ ACM Transaction on Computing Systems, vol., 3 No. 1, pp 63-75, February, 1985

[2] Koo R. and Tueg S., “Checkpointing and Rollback recovery for Distributed Systems”, IEEE Trans. On Software Engineering, Vol. 13 no. 1, pp 23-31, January 1987.

[3] Elonzahy 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.

[4] L. Alvisi,“ Understanding the Message Logging Paradigm for Masking Process Crashes,“ Ph.D. Thesis, Cornell Univ., Dept. of Computer Science, Jan. 1996. Available as Technical Report TR-96-1577.

[5] Lalit Kumar P. Kumar “A synchronous ckeckpointing protocol for mobile distributed systems: probabilistic approach” Int Journal of information and computer security 2007.

[6] Cao, M.Singhal, “Mutable Checkpoints: A New Checkpointing Approach for Mobile Computing Systems”, IEEE Transactions on Parallel and Distributed system, vol.12, Issue 2, Feb., 2001, pages: 157-172, ISSN: 1045-9219.

[7] 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.

[8] M. Singhal and N. Shivaratri, Advanced Concepts in Operating Systems, New York, McGraw Hill, 1994.

[9] 9. 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.

[10] 10. 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.