International Journal of Scientific Research in Science and Technology
Print ISSN: 2395-6011 | Online ISSN: 2395-602X (www.ijsrst.com)
doi : https://doi.org/10.32628/IJSRST218427

Effectual Non-Intrusive Minimum-Process Synchronous Checkpointing Protocol for Mobile Distributed Systems

Raman Kumar[1], Er Jyoti Arora[2]
[1]Research Scholar, Desh Bhagat University Mandi Gobindgarh, Punjab, India
[2]Assistant Professor, Desh Bhagat University Mandi Gobindgarh, Punjab, India

Article Info
Volume 8, Issue 4
Page Number : 325-331

Publication Issue
July-August-2021

Article History
Accepted : 10 July 2021
Published : 20 July 2021

ABSTRACT

While dealing with Mobile Distributed systems, we come across some issues like: mobility, low bandwidth of wireless channels and dearth of stable storage on mobile nodes, disconnections, inadequate battery power and high failure rate of mobile nodes. Minimum-process coordinated checkpointing is considered an attractive methodology to introduce fault tolerance in mobile systems transparently. In this paper, we propose a non-blocking coordinated global state compilation algorithm for mobile computing systems, which requires only a minimum number of processes to take permanent recovery points. We reduce the communication complexity as compared to the Cao-Singhal algorithm [4], while keeping the number of useless recovery points unchanged. Finally, the paper presents an optimization technique, which significantly reduces the number of useless recovery points at the cost of minor increase in the communication complexity. In coordinated global state compilation, if a single process fails to take its tentative recovery point; all the recovery point effort is aborted. We try to reduce this effort by taking soft recovery points in the first phase at Mobile Hosts.

Keywords : Mobile Computing Systems, coordinated checkpointing, Consistent Checkpoints, Global Snapshot, Recovery.

I. INTRODUCTION

In mobile distributed computing system (MDCS), some processes are operating on mobile hosts (M_Hs). An MH is a computer that may retain its connectivity with the rest of the distributed-system through a wireless network while on move or it may detach. It requires integration of portable computers within existing data network. An MH can join to the network from diverse sites at dissimilar times. The infrastructure machines that interconnect directly with the Mob-Hosts are called Mobile Support Stations (M_S_Ss). A cell is a logical or geographical coverage area under an MSS [2, 8, 9, 19, 20].

Local checkpoint is the saved state of a process at a processor at a given instance. Global checkpoint is a collection of local checkpoints, 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 a failure, the system restarts its execution from a previous consistent global state saved on the stable storage during fault-free execution. This saves all the computation done up to the last check pointed state and only the computation done thereafter needs to be redone. Processes in a distributed system communicate by sending and receiving messages [1, 7, 10, 11, 14, 17, 18].

A recovery point algorithm for mobile computing systems needs to handle many new issues like: mobility, low bandwidth of wireless channels, lack of stable storage on mobile nodes, disconnections, limited battery power and high failure rate of mobile nodes. These issues make traditional global state compilation techniques unsuitable for such environments. Minimum-process coordinated global state compilation is an attractive approach to introduce fault tolerance in mobile distributed systems transparently. This approach is domino-free, requires at most two recovery points of a process on stable storage, and forces only a minimum number of processes to recovery point. But, it requires extra synchronization communications, blocking of the underlying computation or taking some useless recovery points [3, 4, 5, 6, 12, 13, 15, 16].

In this paper, we propose a nonblocking coordinated global state compilation algorithm for mobile computing systems, which requires only a minimum number of processes to take permanent recovery points. We reduce the communication complexity as compared to the Cao-Singhal algorithm [4], while keeping the number of useless recovery points unchanged. We also address the related issues like: failures during global state compilation, disconnections, concurrent initiations of the algorithm and maintaining exact dependencies among processes. Finally, the paper presents an optimization technique, which significantly reduces the number of useless recovery points at the cost of minor increase in the communication complexity. In coordinated global state compilation, if a single process fails to take its tentative recovery point; all the recovery point effort is aborted. We try to reduce this effort by taking soft recovery points in the first phase at Mobile Hosts.

In the present study, we propose a nonblocking coordinated global state compilation algorithm for mobile computing systems, which requires only a minimum number of processes to take permanent recovery points. We reduce the communication complexity as compared to [4], while keeping the number of useless recovery points unchanged.

II. The Proposed Checkpointing Algorithm

2.1 Basic Idea

The proposed global state compilation algorithm is based on keeping track of direct dependencies of processes. The initiator M_S_S computes minset [subset of the minimum set] on the basis of dependencies maintained locally; and sends the recovery point request along with the minset[] to the relevant M_S_Ss. On receiving recovery point request, an M_S_S asks concerned processes to recovery point and computes new processes for the minimum set. By using this technique, we have tried to optimize the number of communications between M_S_Ss. In case of example, given in Section 2, point (i), $M\_S\_S_1$ will send just one c_req to $M\_S\_S_2$ to recovery point P_3 and P_4.

When the initiator M_S_S commits the global state compilation process, it sends the commit request along with the exact minimum set to all M_S_Ss and every M_S_S maintains up-to-date csn[]. This enables us to maintain exact dependencies among processes. In our protocol, $ddv_i[j]=1$ only if P_i is directly dependent upon P_j in the current CI. Therefore, useless recovery point requests, are not sent in our algorithm.

When P_i sends c_req to P_j, it also piggybacks $csn_i[j]$ [4]. When P_j receives c_req, it decides, on the basis of piggybacked $csn_i[j]$, whether c_req is useful. In our protocol, no useless c_req is sent, therefore, $csn_i[j]$ is not piggybacked onto c_req.

In algorithm [4], when a process, say P_j, takes its tentative recovery point, it also finds the processes P_k such that P_j has received m from P_k in the current CI. On the basis of MR, received with the recovery point request, P_j decides the following: (i) whether any process has already sent the recovery point request to P_k (ii) whether the earlier recovery point request to P_k is useless. In our protocol, no useless recovery point request is sent, therefore, data structures MR[] is not piggybacked onto recovery point requests. The decision (i) is taken on the basis of tminset, maintained at every M_S_S. tminset maintains the local knowledge about the minimum set. In our case, instead of MR[], tminset is piggybacked onto recovery point requests. The size of the tminset is negligibly small as compared to MR[].

In the first phase, all the M_Hs take induced recovery points. When the initiator M_S_S comes to know that all the processes in the minimum set have taken their mutable recovery points successfully, it sends the request to all concerned processes to convert their mutable recovery points into tentative ones. Finally, when initiator M_S_S comes to know that all concerned processes have taken their tentative recovery points successfully, it issues commit request. In this way, if a process fails to take mutable recovery point in the first phase, then the loss of global state compilation effort is low. If all concerned M_Hs take tentative recovery points in the first phase and some process fails to take its recovery point, then the loss of global state compilation effort will be exceedingly high.

2.2 The Proposed Global state compilation Algorithm

When an M_H sends an application communication, it needs to first send to its local M_S_S over the wireless cell. The M_S_S can piggyback appropriate information onto the application communication, and then route it to the appropriate destination. Conversely, when the M_S_S receives an application communication to be forwarded to a local M_H, it first updates the relevant vectors that it maintains for the M_H, strips all piggybacked information from the communication, and then forwards it to the M_H. Thus, an M_H sends and receives application communications that do not contain any additional information; it is only responsible for global state compilation its local state appropriately and transferring it to the M_S_S.

Each process P_i can initiate the global state compilation process. Initiator M_S_S initiates and coordinates global state compilation process on behalf of M_H_i. It computes minset; and sends c_req along with minset to an M_S_S if the later supports at least one process in the minset. It also updates its tminset on the basis of minset. We assume that concurrent invocations of the algorithm do not occur.

On receiving the c-req, along with the minset from the initiator M_S_S, an M_S_S, say $M\_S\_S_i$, takes the following actions. It updates its tminset on the basis of minset. It sends the c_req to P_i if the following conditions are met: (i) P_i is running in its cell (ii) P_i is a member of the minset and (iii) c_req has not been sent to P_i. If no such process is found, $M\_S\_S_i$ ignores the c_req. Otherwise, on the basis of tminset, ddv vectors of processes in its cell, initial ddv vectors of other processes, it computes tnp_minset. If tnp_minset is not empty, $M\_S\_S_i$ sends c_req along with tminset, tnp_minset to an M_S_S, if the later supports at least one process in the tnp_minset. $M\_S\_S_i$ updates np_minset, tminset on the basis of tnp_minset and initializes tnp_minset.

On receiving c_req along with tminset, tnp_minset from some M_S_S, an M_S_S, say $M\_S\_S_j$, takes the following actions. It updates its own tminset on the basis of received tminset, tnp_minset and finds any process P_k such that P_k is running in its cell, P_k has not been sent c_req and P_k is in tnp_minset. If no such process exists, it simply ignores this request. Otherwise, it sends the recovery point request to P_k. On the basis of tminset, ddv[] of its processes and initial ddv[] of other processes, it computes tnp_minset. If tnp_minset is not empty, $M\_S\_S_j$ sends the recovery point request along with tminset, tnp_minset to an M_S_S, which supports at least one process in the tnp_minset. $M\_S\_S_j$ updates np_minset, tminset on the basis of tnp_minset. It also initializes tnp_minset.

For a disconnected M_H, that is a member of minimum set, the M_S_S that has its disconnected recovery point, converts its disconnected recovery point into tentative one. Algorithm executed at a process on the receipt of a computation communication is given in Section 3.4.

When an M_S_S learns that all of its relevant processes have taken their tentative recovery points successfully or at least one of its processes has failed to take its tentative recovery point, it sends the response communication along with the np_minset to the initiator M_S_S. If, after sending the response communication, an M_S_S receives the recovery point request along with the tnp_minset, and learns that there is at least one process in tnp_minset running in its cell and it has not taken its tentative recovery point, then the M_S_S requests such process to take recovery point. It again sends the response communication to the initiator M_S_S.

When the initiator M_S_S receives a response from some M_S_S, it updates its minset on the basis of np_minset, received along with the response. Finally, initiator M_S_S sends commit/abort to all the processes. When a process in the minimum set receives the commit request, it converts its tentative recovery point into permanent one and discards its earlier permanent recovery point, if any. On receiving commit, a process discards its mutable recovery point, if it is not a member of the minimum set.

An Example of the Proposed Algorithm

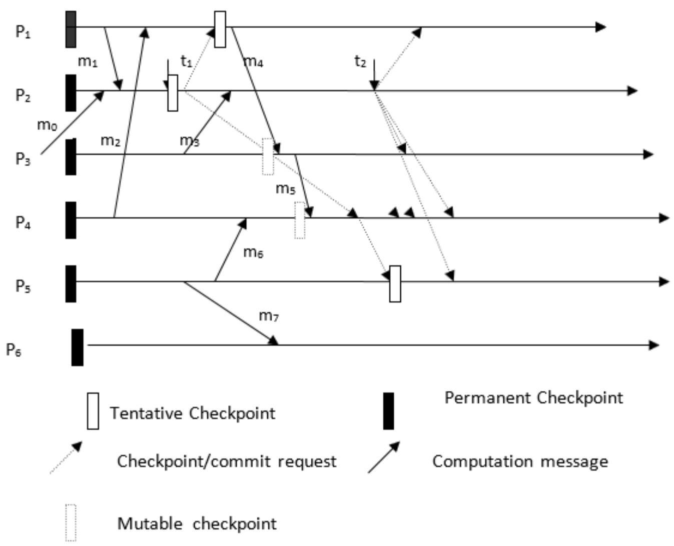

Figure 1

We explain our global state compilation algorithm with the help of an example. In Figure 1, at time t_1, P_2 initiates global state compilation process. $ddv_2[1]=1$ due to m_1; and $ddv_1[4]=1$ due to m_2. On the receipt of m_0, P_2 does not set $ddv_2[3]=1$, because, P_3 has taken permanent recovery point after sending m_0. We assume that P_1 and P_2 are in the cell of the same M_S_S, say $M\_S\_S_{in}$. $M\_S\_S_{in}$ computes minset (subset of minimum set) on the basis of ddv vectors maintained at $M\_S\_S_{in}$, which in case of figure 1 is $\{P_1, P_2, P_4\}$. Therefore, P_2 sends recovery point request to P_1 and P_4. After taking its tentative recovery point, P_1 sends m_4 to P_3. P_3 takes mutable recovery point before processing m_4. Similarly, P_4 takes mutable recovery point before processing m_5. When P_4 receives the recovery point request, it finds that it has already taken the mutable recovery point; therefore, it converts its mutable recovery point into tentative one. P_4 also finds that it was dependent upon P_5 before taking its mutable recovery point and P_5 is not in the minimum set. Therefore, P_4 sends recovery point request to P_5. At time t_2, P_2 receives responses from all relevant processes and sends the commit request along with the minimum set [{P_1, P_2, P_4, P_5}] to all processes. When a process, in the minimum set, receives the commit communication, converts its tentative recovery point into permanent one. When a process, not in the minimum set, receives the commit communication, it discards its mutable recovery point, if any. For the sake of simplicity, we have explained our algorithm with two-phase scheme.

III. General Comparison of the Proposed Algorithm with the Cao-Singhal Algorithm [4]

Some useless recovery point requests are sent in the algorithm [4]; whereas, in the proposed protocol, no such useless recovery point requests are sent. In algorithm [4], when P_i sends recovery point request to P_j, it also piggybacks $csn_i[j]$ and a data structure MR. MR is an array of n pairs and each pair contains two fields: csn and r, where csn contains the csn number and r is a bit vector of length n. MR provides information to the request receivers on recovery point request propagation decision-making. $csn_i[j]$ enables P_j to decide whether P_j inherits the request. These data structures are piggybacked onto recovery point requests to handle useless recovery point requests. In the proposed protocol, no useless recovery point request is sent; therefore, there is no need to piggyback these data structures onto recovery point requests. The $csn_i[j]$ is integer; its size is 4 bytes. In worst case the size of MR[] is (4n +n/8) bytes (n is the number of processes in the distributed system). In the proposed protocol, tminset and tnp_minset are piggybacked onto recovery point requests. Size of each data structure is: n/8 bytes. The extra bytes piggybacked onto each recovery point request in the algorithm [4] as compared to the proposed one are: (29n+32)/8. The number of useless recovery point requests in [18] depends upon the number of processes, communication sending rate, dependency pattern of processes etc. In some cases, the number of useless recovery point requests in [4] may be exceedingly high. The useless recovery point requests further increase the communication complexity of the algorithm [4]. In the proposed protocol, the exact minimum set is broadcasted on the static network along with commit request, whereas in the Cao-Singhal [4] algorithm, only commit request is broadcasted. The size of the minimum set is n/8 bytes.

Conclusions

We have proposed a nonblocking coordinated global state compilation protocol for mobile distributed systems, where only minimum number of processes takes permanent recovery points. We have reduced the communication complexity as compared to Cao-Singhal algorithm [4], while keeping the number of useless recovery points unchanged. The proposed algorithm is designed to impose low memory and computation overheads on M_Hs and low communication overheads on wireless channels. An M_H can remain disconnected for an arbitrary period of time without affecting global state compilation activity. We address the issues like: failures during global state compilation, disconnections, maintaining exact dependencies among processes, and concurrent initiations. We also devise an optimization, which leads to significant reduction in the number of useless recovery points at the cost of a slight increase in the communication overhead.

IV. REFERENCES

[1]. K.M. Chandy and L.Lamport. "Distributed Snapshots: Determining Global States of Distributed Systems" ACM Transactions Computer systems vol. 3, no.1.pp.63- 75, Feb.1985

[2]. 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.

[3]. Guohong Cao and Mukesh Singhal, "On Coordinated Checkpointing in Distributed Systems" IEEE Transaction On Parallel and Distributed Systems, vol. 9, no. 12, pp. 1213-1224, December 1998.

[4]. Guohong Cao and Mukesh Singhal, "Mutable Checkpoints: A New Checkpointing Approach for Mobile Computing Systems", IEEE Transaction On Parallel and Distributed Systems, vol. 12, no. 2, pp. 157- 171, February 2001.

[5]. Weigang Ni, Susan V. Vrbsky and Sibabrata Ray "Pitfalls in Distributed Non blocking Checkpointing", University of Alabama

[6]. Prakash R. and Singhal M. "Maximal Global Snapshot with concurrent initiators," Proc. Sixth IEEE Symp. Parallel and Distributed Processing, pp.344-351, Oct.1994.

[7]. Koo. R. and S.Toueg. "Checkpointing and Rollback- Recovery for Distributed Systems" .IEEE Transactions on Software Engineering, SE-13(1):23-31, January 1987.

[8]. Bidyut Gupta, S.Rahimi and Z.Lui. "A New High Performance Checkpointing Approach for Mobile Computing Systems". IJCSNS International Journal of Computer Science and Network Security, Vol.6 No.5B, May 2006.

[9]. 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.

[10]. Ch.D.V. Subba Rao and M.M.Naidu. "A New, Efficient Coordinated Checkpointing Protocol Combined with Selective Sender-Based Message Logging".

[11]. Nuno Neves and W. Kent Fuchs. "Adaptive Recovery for Mobile Environments",in Proc.IEEE High-Assurance Systems Engineering Workshop,October 21- 22,1996,pp.134-141.

[12]. Y.Manable. "A Distributed Consistent Global Checkpoint Algorithm With minimum number of Checkpoints". Technical Report of IEICE, COMP97-6(April1997).

[13]. J.L.Kim and T.Park. "An efficient protocol for checkpointing recovery in Distributed Systems" IEEE Transaction On Parallel and Distributed Systems,4(8):pp.955-960, Aug 1993.

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

[15]. S.Venkatesan and T.T.-Y.Juang , " Low Overhead Optimistic Crash Recovery:", Preliminary version appears in Proc. 11th Int'l Conf. Distributed Computing Systems as "Crash Recovery with Little Overhead,"pp.454- 461, 1991.

[16]. Parveen Kumar, Lalit Kumar, R K Chauhan, "A Non-intrusive Hybrid Synchronous Checkpointing Protocol for Mobile Systems", IETE Journal of Research, Vol. 52 No. 2&3, 2006.

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

[18]. Mansouri, H., Pathan, A-S.K.: Review of checkpointing and rollback recovery protocols for mobile distributed computing systems. In: Ghosh, U., Rawat D.B., Datta, R., Pathan, A-S.K (eds.) Internet of Things and Secure Smart Environments: Successes and Pitfalls, CRC Press, Taylor & Francis Group (2020).

[19]. Mansouri, H., Pathan, A.-S.K.: Checkpointing distributed application running on mobile Ad

Hoc networks. Int. J. High Perform. Comput. Networking 11(2), 95–107 (2018).

[20]. Mansouri, H., Pathan, A.-S.: A resilient hierarchical checkpointing algorithm for distributed systems running on cluster federation. In: Thampi, S.M., Martinez Perez, G., Ko, R., Rawat, D.B. (eds.) SSCC 2019. CCIS, vol. 1208, pp. 99–110. Springer, Singapore (2020).

Cite this article as :

Raman Kumar, Er Jyoti Arora , "Effectual Non-Intrusive Minimum-Process Synchronous Checkpointing Protocol for Mobile Distributed Systems", International Journal of Scientific Research in Science and Technology (IJSRST), Online ISSN : 2395-602X, Print ISSN : 2395-6011, Volume 8 Issue 4, pp. 325-331, July-August 2021. Available at doi : https://doi.org/10.32628/IJSRST218427
Journal URL : https://ijsrst.com/IJSRST218427