Mathematical Statistician and Engineering Applications
ISSN: 2094-0343
2326-9865

Accomplished Minimum-process Synchronized Consistent Recovery Line Aggregation Algorithm for Fault-Tolerant Mobile Computing Environments

Dr. Sanjiv Sharma[1], Mr. Munish Kumar[2], Dr. Keerti Shrivastva[3], Dr. Sanjeev Kumar[4], Dr. Deepak Chandra Uprety[5]

[1]Senior Faculty, IT Department, University of Technology and Applied Sciences, Ibra, Sultanate of Oman,

[2]Business Consultant(IT),University of the Cumberlands, United State,

[3] Assistant Professor, CSE, ITM University, Gwailor, M.P, India,

[4] Professor, CSA, ITM University, Gwalior, M.P, India,

[5]Associate Professor, Department of Computer Science and Engineering, NIET, Alwar Rajasthan, India,

[1]profsanjiv@gmail.com,[2]munish2012@gmail.com,[3]kirtidev_01@rediffmail.com,

[4]drsanjeevsolanki@gmail.com,[5]deepak.glb@gmail.com

Article Info
Page Number: 9265 - 9273
Publication Issue:
Vol 71 No. 4 (2022)

Abstract

We scheme a minimal-collaborating-proceeding coordinated CRL-aggregation (Consistent Recovery Line Aggregation) arrangement for non-deterministic mobile distributed interconnections, where no inoperable retrieval-points are arrested. We use the following technique to minimize the intrusion of proceedings. During the timeline, when a proceeding dispatches its causal-interrelationship set to the instigator and acquires the minimal-collaborating-set, may acquire some computation-messages, which may add new members to the already computed minimal-collaborating-set. Such computation-messages are delayed at the receiver side. It should be noted that the duration for which the computation-messages are delayed at the receiver's end is negligibly small. We also try to minimize the loss of CRL-aggregation effort when any proceeding flops to arrest its retrieval-point in harmonization with others. We scheme that in the first phase, all pertinent Mbl_Nods (Mobile Modes) will arrest evanescent retrieval-point only. Evanescent retrieval-point is stored on the memory of Mbl_Nod only. In this case, if some proceeding flops to arrest retrieval-point in the first phase, then Mbl_Nods need to discard their evanescent retrieval-points only. The effort of arresting an evanescent retrieval-point is negligible as compared to the quasi-persistent one. In the schemed arrangement, the harmonization with the instigator Mbl_Suppt_Stn is done without dispatching explicit control-messages. We want to emphasize that in all coordinated CRL-aggregation schemes available in literature, harmonization among proceedings and instigator takes place by dispatching explicit control-messages. In this way, we try to significantly reduce the harmonization overhead in coordinated CRL-aggregation .

Article History
Article Received: *15 September 2022*
Revised: *25 October 2022*
Accepted: *14 November 2022*
Publication: *21 December 2022*

Keywords: - Fault tolerance, distributed systems, consistent recovery line, coordinated checkpointing, and mobile computing

1. Introduction

In the mobile distributed interconnection, some of the proceedings are running on mobile hosts (Mob_Nodes). A Mob_Node converses with other nodes of the interconnection via a special node called mobile support station (Mob_Supp_Stn) [1]. A cell is a geographical area around a Mobl_Suppt_stn in which it can support a Mob_Node. A Mob_Node can change its geographical position freely from one cell to another or even to an area covered by no cell. A Mob_Supp_Stn can have 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 Mob_Supp_Stns. A static node that has no support to Mob_Node can be considered as a Mob_Supp_Stn with no Mob_Node.

Checkpoint/retrieval-mark is defined as a designated place in a program at which normal proceeding is interrupted specifically to preserve the status information necessary to allow resumption of handling at a later time. CRL-aggregation is the process of saving the status information. By invoking the CRL-aggregation algorithm, one can save the status of a program at regular intervals. If there is a failure one may restart computation from the last retrieval-mark thereby avoiding repeating computation from the beginning. The proceeding of resuming computation by rolling back to a saved state is called rollback recovery. The retrieval-mark-restart is one of the well-known methods to realize reliable distributed interconnections. Each proceeding arrests a retrieval-mark where the local state information is stored in the stable storage. Rolling back a proceeding and again resuming its accomplishment from a prior state involves overhead and delays the overall completion of the proceeding, it is needed to make a proceeding rollback to a most recent possible state. So it is at the desire of the user for taking many retrieval-marks over the whole life of the accomplishment of the proceeding [6, 27].

In a distributed interconnection, since the proceedings in the interconnection do not share memory, a global state of the interconnection is defined as a set of local states, one from each proceeding. The state of channels corresponding to a global state is the set of computation-messages consigned but not yet acquired. A global state is said to be “dependable” if it contains no orphan computation-message; i.e., a computation-message whose acquire event is recorded, but its forward event is lost. To recover from a failure, the interconnection restarts its accomplishment from a previous dependable global state saved on the stable storage during fault-free accomplishment. This saves all the computation done up to the last retrieval-marked state and only the computation done thereafter needs to be redone. In distributed interconnections, CRL-aggregation can be independent, synchronized [6, 11, 13] or quasi-synchronous [2]. Message Logging is also used for fault tolerance in distributed interconnections [22, 28].

In synchronized or synchronous CRL-aggregation, proceedings arrest retrieval-marks in such a manner that the resulting global state is dependable. Mostly it follows two-phase commit structure [6, 11, 23]. In the first phase, proceedings arrest quasi-persistent retrieval-marks and in the second phase, these are made persistent. The main advantage is that only one persistent retrieval-mark and

at most one quasi-persistent retrieval-mark is prerequisite to be stored. In case of a fault, proceedings rollback to last retrieval-marked state.

The synchronized CRL-aggregation protocols can be classified into two types: intrusive and non-intrusive. In intrusive schemes, some intrusive of proceedings arrests place during CRL-aggregation [4, 11, 24, 25, 29]. In non-intrusive schemes, no intrusion of proceedings is prerequisite for CRL-aggregation [5, 12, 15, 21]. The synchronized CRL-aggregation schemes can also be classified into following two categories: minimum-collaborating-proceeding and all proceeding schemes. In all-proceeding synchronized CRL-aggregation schemes, every proceeding is prerequisite to arrest its retrieval-mark in a commencement [6], [8]. In minimum-collaborating-proceeding schemes, minimum interacting proceedings are prerequisite to arrest their retrieval-marks in a commencement [11].

In minimum-collaborating-proceeding synchronized CRL-aggregation schemes, a proceeding Pi arrests its retrieval-mark only if it a member of the minimum set (a subset of interacting proceeding). A proceeding Pi is in the minimum set only if the retrieval-mark instigator proceeding is transitively dependent upon it. Pj is directly dependent upon Pk only if there exists m such that Pj accepts m from Pk in the current CRL-aggregation interval [CI] and Pk has not arrested its persistent retrieval-mark after forwarding m. The ith CI of a proceeding denotes all the computation performed between its ith and (i+1)th retrieval-mark, including the ith retrieval-mark but not the (i+1)th retrieval-mark.

In minimum-collaborating-proceeding CRL-aggregation protocols, some useless retrieval-marks are arrested or intrusive of proceedings arrests place. In this paper, we scheme a minimum-collaborating-proceeding synchronized CRL-aggregation methodology for non-deterministic mobile distributed interconnections, where no useless retrieval-marks are arrested. An effort has been made to minimize the intrusive of proceedings and the loss of CRL-aggregation effort when any proceeding fails to arrest its retrieval-mark in coordination with others.

2. Basic Idea

We scheme a three phase arrangement. But, in the schemed arrangement, the harmonization with the instigator Mbl_Suppt_Stn is done without dispatching explicit control-messages. The instigator Mbl_Suppt_Stn (say Mbl_Suppt_Stn) collects the interdependency arrays of all proceedings, computes the minimal-collaborating-set and dispatches the evanescent retrieval-point request to all Mbl_Suppt_Stn along with the minimal-collaborating-set. Suppose, Mbl_Suppt_Stn gets the evanescent retrieval-point request in the first phase from Mbl_Suppt_Stn. It sets its timer (timer_evanescent) and dispatches the evanescent retrieval-point request to all pertinent resident Mbl_Nods. The timer_evanescent is the maximum allowable time for all pertinent proceedings to arrest their evanescent retrieval-points. On acquiring the evanescent retrieval-point request, a Mbl_Nod arrests its evanescent retrieval-point and dispatches the response to $Mbl\_Suppt\_Stn_i$. Before the expiry of the timer_evanescent, if $Mbl\_Suppt\_Stn_{in}$ gets the negative response from some Mbl_Nod to its evanescent retrieval-point request, then $Mbl\_Suppt\_Stn_{in}$ dispatches the negative response to $Mbl\_Suppt\_Stn_{in}$ and $Mbl\_Suppt\_Stn_{in}$ issues discard computation-message to

all Mbl_Suppt_Stns. Otherwise, on expiry of timer_evanescent, if Mbl_Suppt_Stn$_i$ does not get the positive response to evanescent retrieval-point request from all pertinent resident Mbl_Nods, it informs letdown computation-message to Mbl_Suppt_Stn$_{in}$ and Mbl_Suppt_Stn$_{in}$ issues discard. Alternatively, on expiry of timer_evanescent Mbl_Suppt_Stni issues quasi-persistent retrieval-point request to the pertinent Mbl_Nods in its cubicle and sets tim_tentv_rm. On expiry of timer_evanescent, if Mbl_Suppt_Stn$_i$ does not get discard massage from Mbl_Suppt_Stn$_{in}$, it is presumed that all pertinent proceedings have arrested their evanescent retrieval-points ; and the arrangement should enter the second phase in which all pertinent proceedings convert their evanescent retrieval-points into the quasi-persistent ones. Similarly, tim_tentv_rm is the maximum allowable time for all pertinent proceedings to convert their evanescent retrieval-points into quasi-persistent ones. If some proceeding flops to arrest its quasi-persistent retrieval-point, then Mbl_Suppt_Stn$_i$ informs Mbl_Suppt_Stn$_{in}$ and Mbl_Suppt_Stnin issues discard. Otherwise, after the timeout of tim_tentv_rm, Mbl_Suppt_Stnin commits the retrieval-points of the proceedings of the minimal-collaborating-sets which are resident to its cubicle. On expiry of tim_tentv_rm, if Mbl_Suppt_Stn$_i$ does not get discard massage from Mbl_Suppt_Stn$_{in}$, it is presumed that all pertinent proceedings have arrested their quasi-persistent retrieval-points; and the arrangement should enter the third phase in which all pertinent proceedings convert their quasi-persistent retrieval-points into the persistent ones. In this way, three-phase coordinated CRL-aggregation arrangement commits without dispatching or acquiring any control-messages. Only in the case of a letdown a Mbl_Suppt_Stn issues the letdown computation-message to Mbl_Suppt_Stnin and Mbl_Suppt_Stnin issues the commit. The schemed arrangement may arrest longer time to commit. But in doing so, we are saving control-messages to significant extent and no extra intrusive of proceedings arrests place due to longer commit time.

3. The Proposed Minimum-process Synchronized Consistent Recovery Line Aggregation Algorithm

The instigator Mbl_Suppt_Stn dispatches a request to all Mbl_Suppt_Stns to dispatch the *cci_vect* vectors of the proceedings in their cubicles. All *cci_vect* vectors are at Mbl_Suppt_Stns and thus no initial CRL-aggregation computation-messages or responses travels wireless channels. On acquiring the *cci_vect* [] request, a Mbl_Suppt_Stn records the identity of the instigator proceeding (say Mbl_Suppt_Stn_id$_a$) and instigator Mbl_Suppt_Stn, dispatches back the *cci_vect* [] of the proceedings in its cubicle, and sets *g_chkpt*. If the instigator Mbl_Suppt_Stn acquires a request for *cci_vect* [] from some other Mbl_Suppt_Stn (say Mbl_Suppt_Stn_id$_b$) and Mbl_Suppt_Stn_id$_a$ is lower tha Mbl_Suppt_Stn_id$_b$,the, current commencement with Mbl_Suppt_Stn_id$_a$ is discarded and the new one having Mbl_Suppt_Stn_id$_b$ is continued. Similarly, if a Mbl_Suppt_Stn acquires *cci_vect* requests from two Mbl_Suppt_Stns, then it discards the request of the instigator Mbl_Suppt_Stn with lower Mbl_Suppt_Stn_id. Otherwise, on acquiring *cci_vect* vectors of all proceedings, the instigator Mbl_Suppt_Stn computes *min_coll_vectr*[], dispatches evanescent retrieval-point request along with the *min_coll_vectr*[] to all Mbl_Suppt_Stns. In this way, if two proceedings contemporaneously instigate CRL-aggregation , then one is ignored. When a proceeding dispatches its *cci_vect*[] to the instigator Mbl_Suppt_Stn, it comes into its intrusive state. A proceeding comes out of the intrusive state only after arresting its evanescent retrieval-point

if it is a member of the minimal-collaborating-set; otherwise, it comes out of intrusive state after acquiring the evanescent retrieval-point request. It should be noted that the intrusive time of a proceeding is bare least.

On acquiring the evanescent retrieval-point request along with the min_coll_vectr[], a Mbl_Suppt_Stn, say $Mbl\_Suppt\_Stn_j$, arrests the following actions. It sets the timer timer_evanescent; dispatches the evanescent retrieval-point request to Pi only if Pi belongs to the min_coll_vectr [] and Pi is running in its cubicle. On acquiring the retrieval-point request, Pi arrests its evanescent retrieval-point and informs $Mbl\_Suppt\_Stn_j$. On acquiring positive response from Pi, $Mbl\_Suppt\_Stn_j$ updates o-rmsni, resets intrusivei, and dispatches the buffered computation-messages to Pi, if any. Change natively, If Pi is not in the min_coll_vectr [] and Pi is in the cubicle of Mbl_Suppt_Stnj, Mbl_Suppt_Stnj resets intrusivei and dispatches the buffered computation-message to Pi, if any. For a disengaged Mbl_Nod, that is a member of min_coll_vectr [], the Mbl_Suppt_Stn that has its disengaged retrieval-point, transforms its disengaged retrieval-point into the prerequisite one.

During intrusive timeline, Pi proceedings m, acquired from Pj, if following conditions are met:

(i) (!buferi) i.e. Pi has not buffered any computation-message

(ii) (m.psn <=rmsn[j]) i.e. Pj has not arrested its retrieval-point before dispatching m

(iii) (cci_vecti[j]=1) Pi is already dependent upon Pj in the current CI or Pj has arrested some persistent retrieval-point after dispatching m.

Otherwise, the resident Mbl_Suppt_Stn of Pi buffers m for the intrusive timeline of Pi and sets bufferi.

On expiry of timer_evanescent, if Mbl_Suppt_Stnj does not get the positive response to evanescent retrieval-point request from all pertinent resident Mbl_Nods, it informs letdown computation-message to $Mbl\_Suppt\_Stn_{in}$ and $Mbl\_Suppt\_Stn_{in}$ issues discard. Change natively, on expiry of timer_evanescent Mbl_Suppt_Stnj issues quasi-persistent retrieval-point request to the pertinent Mbl_Nods in its cubicle and sets tim_tentv_rm.

If some proceeding flops to arrest its quasi-persistent retrieval-point, then Mbl_Suppt_Stnj informs Mbl_Suppt_Stnin and Mbl_Suppt_Stnin issues discard. Otherwise, after the timeout of tim_tentv_rm, Mbl_Suppt_Stnj commits the retrieval-points of the proceedings of the minimal-collaborating-sets which are resident to its cubicle. On expiry of tim_tentv_rm, if $Mbl\_Suppt\_Stn_i$ does not get discard massage from Mbl_Suppt_Stnin, it is presumed that all pertinent proceedings have arrested their quasi-persistent retrieval-points efficaciously; and the arrangement should enter the third phase in which all pertinent proceedings convert their quasi-persistent retrieval-points into the persistent ones.

4. An Example of the Schemed Scheme

We explain the schemed minimal-collaborating-proceeding CRL-aggregation arrangement with the help of an example. In Figure 1, at time t0, P5 instigates CRL-aggregation proceeding and dispatches request to all proceedings for their interdependency arrays. At time t1, P5 acquires the interdependency arrays from all proceedings and computes the minimal-collaborating-set (min_coll_vectr[]) which is {P4, P5, P6}. The computation of the minimal-collaborating-set on the basis of interdependency arrays of all proceedings can be found in [14, 16]. For the sake of simplicity, the control computation-messages by which the proceedings dispatch their interdependency arrays to the instigator proceeding P5 are not shown in the Figure 1. P5 dispatches minimal-collaborating-set (min_coll_vectr[]) to all proceedings and arrests its own evanescent retrieval-point C51. On acquiring min_coll_vectr[], a proceeding arrests its evanescent retrieval-point if it is a member of min_coll_vectr[]. When P4 and P6 get the min_coll_vectr[], they find themselves to be the members of the min_coll_vectr[]; therefore, they arrest their evanescent retrieval-points , C41 and C61, respectively. When P1, P2 and P3 get the min_coll_vectr [], they find that they do not belong to min_coll_vectr [], therefore, they do not arrest their evanescent retrieval-points. It should be noted that these proceedings have not consigned any computation-message to any proceeding of the minimal-collaborating-set. In other words, P5 is not transitively dependent upon them. Therefore, for the sake of consistency, it is not necessary for them to arrest their retrieval-points in the current commencement.

A proceeding comes into the intrusive state immediately after dispatching the cci_vect[]. A proceeding comes out of the intrusive state only after arresting its evanescent retrieval-point, if it is a member of the minimal-collaborating-set; otherwise, it comes out of intrusive state after acquiring the evanescent retrieval-point request. We want to say that the intrusive time of a proceeding in this arrangement is insignificantly small. Moreover, a proceeding is allowed to perform its normal computation, dispatch computation-messages and partially acquire them during the intrusive timeline. For example, P5 acquires m4 during its intrusive timeline. As cci_vect5[6]=1 due to m2, and acquire of m4 will not change cci_vect5[]; therefore P5 proceedings m4. P2 acquires m15 from P3 during its intrusive timeline; cci_vect2[3]=0 and the acquire of m15 can change cci_vect2[]; therefore, P2 buffers m15. Similarly, P4 buffers m16. P4 processes m16 only after arresting its evanescent retrieval-point C41. P2 processes m15 after acquiring the min_coll_vectr []. P4 processes m7 because at this moment it not in the intrusive state. Similarly, P4 processes m8.

On acquiring the evanescent retrieval-point request, a proceeding, say P6, sets the timer timer_evanescent. If P6 flops to arrest its evanescent retrieval-point, it informs P5 and P5 will issue discard. Similarly, if any other proceeding flops to arrest its evanescent retrieval-point, it will inform P5 and P5 will inform P6. In this way, if any proceeding flops to arrest its retrieval-point in harmonization with others in the first phase, then all proceedings need to discard their evanescent retrieval-points only and not the quasi-persistent retrieval-points as in other arrangements [14, 15, 16]. In this way, we are able to significantly reduce the loss of CRL-aggregation effort in case of a letdown during CRL-aggregation. Change natively, on timeout of timer_evanescent and no discard computation-message from P5, it is presumed that all pertinent proceedings have arrested their evanescent retrieval-points efficaciously and the arrangement should enter into the second phase.

Therefore, P6 transforms its evanescent retrieval-point into quasi-persistent one and sets the timer tim_tentv_rm. If P6 flops to convert its evanescent retrieval-point into quasi-persistent one, it informs P5 and P5 will issue discard. Similarly, if any other proceeding flops to arrest its evanescent retrieval-point, it will inform P5 and P5 will inform P6. Otherwise, on timeout of tim_tentv_rm, P6 transforms its quasi-persistent retrieval-point into persistent one. On timeout of tim_tentv_rm and no discard computation-message from P5, it is presumed that all pertinent proceedings have arrested their quasi-persistent retrieval-points efficaciously and the arrangement should enter into the second phase. In this way, we commit the retrieval-points without much harmonization.

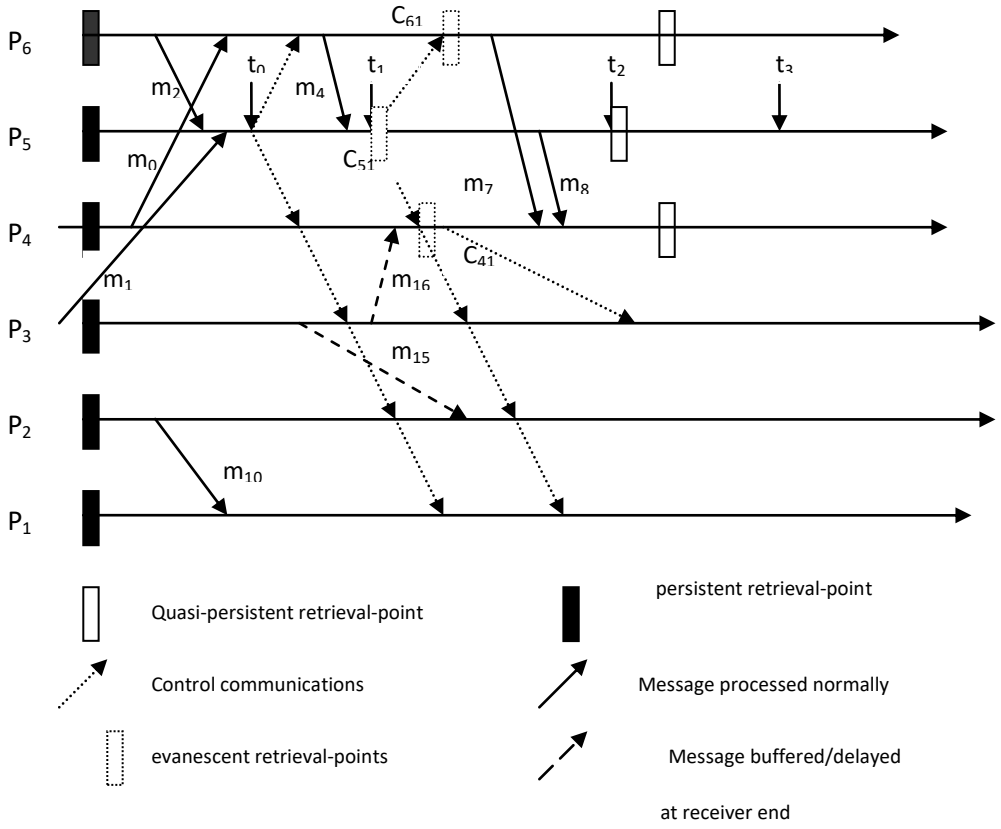

Figure 1 An Example of the proposed Protocol

5. Conclusion

We have designed a minimal-collaborating-proceeding synchronous CRL-aggregation arrangement for mobile distributed interconnection. We try to minimize the intrusion of proceedings during CRL-aggregation . The intrusive time of a proceeding is bare least. During intrusive timeline, proceedings can do their normal computations, dispatch computation-messages and can process selective computation-messages. The number of proceedings that arrest retrieval-points is diminished to avoid awakening of Mbl_Nods in doze mode of operation and thrashing of Mbl_Nods with CRL-aggregation activity. It also saves limited battery life of Mbl_Nods and low bandwidth of wireless channels. We try to reduce the loss of CRL-aggregation effort when any

proceeding flops to arrest its retrieval-point in harmonization with others. We also try to minimize the control-messages during CRL-aggregation. In the schemed scheme, minimal control-messages are consigned in order to enter the second or third phase of the arrangement.

References:-

[1] A. Acharya and B. R. Badrinath, *Checkpointing Distributed Applications on Mobile Computers*, In Proceedings of the 3rd International Conference on Parallel and Distributed Information Systems (PDIS 1994), 1994, 73-80..

[2] A. Acharya and B. R. Badrinath, *Checkpointing Distributed Applications on Mobile Computers*, In Proceedings of the 3rd International Conference on Parallel and Distributed Information Systems (PDIS 1994), 1994, 73-80..

[3] G. Cao and M. Singhal, On coordinated checkpointing in Distributed Systems, *IEEE Transactions on Parallel and Distributed Systems*, 9 (12), 1998, 1213-1225.

[4] G. Cao and M. Singhal, "*On the Impossibility of Min-process Non-blocking Checkpointing and an Efficient Checkpointing Algorithm for Mobile Computing Systems*," In Proceedings of International Conference on Parallel Processing, 1998, 37-44.

[5] G. Cao and M. Singhal, Mutable Checkpoints: A New Checkpointing Approach for Mobile Computing systems, *IEEE Transaction On Parallel and Distributed Systems*, 12(2), 2001, 157-172.

[6] K.M. Chandy and L.Lamport, "Distributed Snapshots: Determining Global State of Distributed Systems," *ACM Transaction on Computing Systems*, 3(1), 1985, 63-75.

[7] E. N. Elnozahy, L. Alvisi, Y. M. Wang and D. B. Johnson, "A Survey of Rollback-Recovery Protocols in Message-Passing Systems," *ACM Computing Surveys*, 34(3), 2002, 375-408.

[8] E.N. Elnozahy, D.B. Johnson and W. Zwaenepoel, *The Performance of Consistent Checkpointing*, In Proceedings of the 11th Symposium on Reliable Distributed Systems, 1992, 39-47.

[9] J.M. Hélary, A. Mostefaoui and M. Raynal, *Communication-Induced Determination of Consistent Snapshots*, In Proceedings of the 28th International Symposium on Fault-Tolerant Computing, 1998, 208-217..

[10] H. Higaki and M. Takizawa, Checkpoint-recovery Protocol for Reliable Mobile Systems, *Transactions of Information processing Japan*, 40(1), 1999, 236-244.

[11] R. Koo and S. Toueg, Checkpointing and Roll-Back Recovery for Distributed Systems, *IEEE Transactions on Software Engineering*, 13(1), 1987, 23-31.

[12] P. Kumar, L. Kumar, R. K. Chauhan and V. K. Gupta, *A Non-Intrusive Minimum Process Synchronous Checkpointing Protocol for Mobile Distributed Systems*, In Proceedings of IEEE ICPWC-2005, 2005.

[13] J.L. Kim and T. Park, An efficient Protocol for checkpointing Recovery in Distributed Systems, *IEEE Transactions on Parallel and Distributed Systems*, 1993, 955-960.

[14] L. Kumar, M. Misra, R.C. Joshi, Checkpointing in Distributed Computing Systems, In Concurrency in Dependable Computing, 2002, 273-92.

[15] L. Kumar, M. Misra, R.C. Joshi, *Low overhead optimal checkpointing for mobile distributed systems*, In Proceedings of 19th IEEE International Conference on Data Engineering, 2003, 686

– 88.

[16] L. Kumar and P.Kumar, A Synchronous Checkpointing Protocol for Mobile Distributed Systems: Probabilistic Approach, *International Journal of Information and Computer Security*, 1(3), 2007, 298-314.

[17] L. Lamport, Time, clocks and ordering of events in a distributed system, *Communications of the ACM*, 21(7), 1978, 558-565.

[18] N. Neves and W.K. Fuchs, Adaptive Recovery for Mobile Environments, *Communications of the ACM*, 40(1), 1997, 68-74.

[19] W. Ni, S. Vrbsky and S. Ray, Pitfalls in Distributed Nonblocking Checkpointing, *Journal of Interconnection Networks*, 1(5), 2004, 47-78.

[20] D.K. Pradhan, P.P. Krishana and N.H. Vaidya, *Recovery in Mobile Wireless Environment: Design and Trade-off Analysis*, In Proceedings of 26th International Symposium on Fault-Tolerant Computing, 1996, 16-25..

[21] R. Prakash and M. Singhal, Low-Cost Checkpointing and Failure Recovery in Mobile Computing Systems, *IEEE Transaction On Parallel and Distributed Systems*, 7(10), 1996, 1035-1048.

[22] K.F. Ssu, B. Yao, W.K. Fuchs and N.F. Neves, Adaptive Checkpointing with Storage Management for Mobile Environments, *IEEE Transactions on Reliability*, 48(4), 1999, 315-324.

[23] L.M. Silva and J.G. Silva, *Global checkpointing for distributed programs*, In Proceedings of the 11th symposium on Reliable Distributed Systems, 1992, 155-62.

[24] Sunil Kumar, R K Chauhan, Parveen Kumar, "A Minimum-process Coordinated Checkpointing Protocol for Mobile Computing Systems", International Journal of Foundations of Computer science,Vol 19, No. 4, pp 1015-1038 (2008).

[25] Parveen Kumar, "A Low-Cost Hybrid Coordinated Checkpointing Protocol for mobile distributed systems", Mobile Information Systems. pp 13-32, Vol. 4, No. 1, 2007.

[26] Rao, S., & Naidu, M.M, "A New, Efficient Coordinated Checkpointing Protocol Combined with Selective Sender-Based Message Logging", IEEE/ACS International Conference on Computer Systems and Applications, 2008.

[27] Biswas S, &Neogy S,"A Mobility-Based Checkpointing Protocol for Mobile Computing System", International Journal of Computer Science & Information Technology, Vol.2, No.1,pp135-15,2010.

[28] Praveen Choudhary, Parveen Kumar," Low-Overhead Minimum-Method Global-Snapshot Compilation Protocol for Deterministic Mobile Computing Systems ", International Journal of Emerging Trends in Engineering Research" Vol. 9, Issue 8, Aug 2021, pp.1069-1072.