A Coherent Minimum-Process Dependable Reclamation Line Collation Scheme for Fault-Tolerant Mobile Distributed Systems

Dr. Deepak Dagar[1], Dr. Ajay Sharma[2], Dr. Sanjeev Kumar[3], Dr. Ajeet Kumar[4], Mohit Kumar[5]

[1]Assistant Professor,Maharaja Agrasen Institute of Management Studies,Delhi;deepakdagar.faculty@maims.ac.in

[2]Associate Professor, Kasturi Ram College of Higher Education, Delhi; ajay0202@gmail.com

[3]Assistant Professor,Maharaja Agrasen Institute of Technology, Delhi; drsanjeevsharma001@gmail.com

[4]Assistant Professor, Department of Computer Science and Engineering, Tula's Institue, Dehradun; ajeet7488@gmail.com

[5]Assistant Professor, Department of Electronic Engineering, Tula's Institute, Dehradun; mohitdodval@gmail.com

Article Info

Page Number: 1933 - 1943

Publication Issue:

Vol 71 No. 3 (2022)

Article History

***Article Received**: 12 January 2022*

***Revised**: 25 February 2022*

***Accepted**: 20 April 2022*

***Publication**: 09 June 2022*

Abstract

We advocate a minimal-process coordinated Dependable Reclamation Line Collation arrangement for non-deterministic mobile distributed interconnection; where no incompetent recuperation-points are captured. An effort has been made to moderate the intrusion of proceedings and synchronization overhead. We capture the partial transitive interdependencies during the normal accomplishment by piggybacking interdependency arrays onto computation communications.Recurrent terminations of Dependable Reclamation Line Collation arrangement may happen in mobile interconnection due to exhausted battery, non-voluntary disengagements of M_Nodules (Mobile Nodules), or poor mobile connectivity.Therefore, we advocate that in the first stage, all pertinent M_Nodules will capture evanescent recuperation-point only. Evanescent recuperation-point is stored on the memory of M_Nodule only. In this case, if some proceeding miscarries to capture recuperation-point in the first stage, then M_Nodules need to abandon their evanescent recuperation-points only. In this way, we try to moderate the loss of Dependable Reclamation Line Collation (DRL-collation) work when any proceeding miscarries to capture its recuperation-point in coordination with others

Keywords:- Fault tolerance**,** consistent global state, coordinated Dependable Reclamation Line Collation and mobile interconnection.

1. Introduction

In the mobile distributed interconnection, some of the proceedings are running on mobile nodules(M_Nodules). AM_Nodule communicates with other nodes of the interconnection via a special node calledmobile support station (Mobl_Suppt_stn) [1]. A cubicle is a geographical area around a Mobl_Suppt_stn in which it can support an M_Nodule. A M_Nodule can change its geographical position freely from one cubicle to another or even to an area covered by no cubicle. A Mobl_Suppt_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 Mobl_Suppt_stn.

Recuperation-point is defined as a designated place in a program at which normal processing is interrupted specifically to preserve the status information necessary to allow resumption of

processing at a later time. DRL-collation (**Dependable Reclamation Line Collation**) is the act of saving the status information. By periodically invoking the DRL-collation protocol,
one can save the status of a program at regular intervals. If there is a letdown one may restart computation from the last DRL thereby avoiding repeating computation from the beginning.

The process of resuming computation by rolling back to a saved state is called rollback-reclamation. The recuperation-point-restart is one of the well-known methods to realize dependable distributed interconnection. Each proceeding arrests a recuperation-point where the local state information is stored in the stable storage. Rolling back a proceeding and again recommencing its execution from a prior state involves overhead and delays the overall completion of the computation, it is needed to make a proceeding rollback to a most recent possible state. So it is at the desire of the user for arresting many recuperation-points over the whole life of the execution of the proceeding [6, 29, 30].

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-communications dispatched but not yet received. A global state is said to be "consistent" if it contains no orphan computation-communication; i.e., a computation-communication whose receive event is recorded, but its send event is lost. To recover from a letdown, the interconnection 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 DRL and only the computation done thereafter needs to be redone. In distributed interconnection, DRL-collation arrangements can be independent, coordinated [6, 11, 13] or quasi-synchronous [2]. Message Logging is also used for fault tolerance in distributed interconnection [22, 29, 30].

In coordinated or synchronous DRL-collation, proceedings arrest recuperation-points in such a manner that the resulting global state is consistent. Mostly it follows two-stage commit structure [6, 11, 23]. In the first stage, proceedings arrest partially-persistent recuperation-points and in the second stage, these are made persistent. The main advantage is that only one persistent recuperation-point and at most one partially-persistent recuperation-point is necessitated to be stored. In the case of a fault, all proceedings rollback to the last DRL.
The coordinated DRL-collation protocols can be classified into two types: intrusion and non-intrusion. In intrusion arrangements, some intrusion of proceedings takes place during DRL-collation [4, 11, 24, 25]. In non-intrusive arrangements, no intrusion of proceedings is necessitated for DRL-collation [5, 12, 15, 21]. The coordinated DRL-collation arrangements can also be classified into following two categories: minimum-proceeding and all proceeding arrangements. In all-proceeding coordinated DRL-collation arrangements, every proceeding is necessitated to arrest its recuperation-point in a commencement [6], [8]. In minimum-proceeding arrangements, minimum interacting proceedings are necessitated to arrest their recuperation-points in a commencement [11].

In minimum-proceeding coordinated DRL-collation arrangements, a proceeding P_i arrests its recuperation-point only if it a member of the minimum set (a subset of interacting proceeding). A proceeding P_i is in the minimum set only if the recuperation-point initiator proceeding is transitively dependent upon it. $P_j$is directly dependent upon $P_k$only if there exists *m* such that $P_j$receives*m* from $P_k$in the current DRL-collation interval [CI] and P_k00000000000 not arrested its persistent recuperation-point after sending *m*. The i^{th} CI of a proceeding denotes all the computation performed between its $i^{th}$and $(i+1)^{th}$ recuperation-point , including the i^{th} recuperation-point but not the $(i+1)^{th}$ recuperation-point .

In minimum-proceeding DRL-collation protocols, some unserviceable recuperation-points are arrested or intrusion of proceedings takes place. In this paper, we advocate a minimum-proceeding coordinated DRL-collation arrangement for non-deterministic mobile distributed interconnection, where no unworkable recuperation-points are arrested. An effort has been made to minimize the intrusion of proceedings and the loss of DRL-collation effort when any proceeding miscarries to arrest its recuperation-point in coordination with others.

Rao and Naidu [26] advocated a new coordinated DRL-collation protocol combined with selective sender-based computation-communication logging. The protocol is free from the problem of lost computation-communications. The term ‘selective’ implies that computation-communications are logged only within a specified interval known as active interval, thereby reducing computation-communication logging overhead. All proceedings arrest recuperation-points at the end of their respective active intervals forming a consistent global recuperation-point.Biswas&Neogy [27] advocated a DRL-collationand letdown reclamation arrangement where mobile hosts save recuperation-points based on mobility and movement patterns. Mobile hosts save recuperation-points when number of hand-offs exceed a predefined handoff threshold value. Neves& Fuchs [18] designed a time based loosely synchronized coordinated DRL-collationprotocol that removes the overhead of synchronization and piggybacks integer csn (recuperation-point sequence number). Gao et al [28] developed an index-based arrangement which uses time-coordination for consistently DRL-collation in mobile computing environments. In time-based DRL-collation protocols, there is no need to send extra coordination computation-communications. However, they have to deal with the synchronization of timers. This class of protocols suits to the applications where proceedings have high computation-communication sending rate.

2. Basic Idea

All Communications to and from M_Nodule pass through its resident Mobl_Supp_St. The Mobl_Supp_St maintains the interdependency information of the M_Nodules which are in its cubicle. The interdependency information is kept in Boolean array R_i for proceeding P_i. The array has n bits for n proceedings. When $R_i[j]$ is set to 1, it repredispatcheds 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 proceeding P_i running on a M_Nodule , say M_Nodule$_p$, acquires a communication from a proceeding P_j, M_Nodule$_p$'s resident Mobl_Supp_St should set $R_i[j]$ to 1.If P_j has captured its persistent recuperation-point after forwarding m, $R_i[j]$ is not updated.

Suppose there are proceedings P_i and P_j running on M_Nodules, M_Nodule$_i$ and M_Nodule$_j$ with interdependency arrays R_i and R_j. The interdependency arrays of M_Nodules, M_Nodule$_i$ and M_Nodule$_j$ are maintained by their resident Mobl_Supp_Sts, Mobl_Supp_St$_i$ and Mobl_Supp_St$_j$. Process P_i running on M_Nodule$_i$ forwards communication m to proceeding P_j running on M_Nodule$_j$. The communication is first dispatched to Mobl_Supp_St$_i$ (resident Mobl_Supp_St of M_Nodule$_i$). Mobl_Supp_St$_i$ maintains the interdependency array R_i of M_Nodule$_i$. Mobl_Supp_St$_i$ appends R_i with communication m and forwards it to Mobl_Supp_St$_j$(resident Mobl_Supp_St of M_Nodule$_j$). Mobl_Supp_St$_j$ maintains the interdependency array R_j of M_Nodule$_j$. Mobl_Supp_St$_j$ replaces R_j with bitwise logical OR of interdependency arrays R_i and $R_j$and forwards m to P_j.

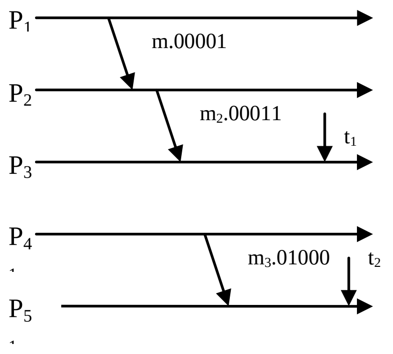

Figure 1 Upkeep of Interdependency Vectors

In Figure 1, there are five proceedings P_1, P_2, P_3, P_4, P_5 with interdependency arrays R_1, R_2, R_3, R_4, R_5 initialized to 00001, 00010, 00100, 01000, and 10000 respectively. Initially, every proceeding depends upon itself. Now proceeding P_1 forwards 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 forwards m2 to P_3 and appends R_2 (00011) with m_2. Before acquiring m_2, the value of R_3 at P_3 was 00100. After acquiring 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 forwards m_3 along with R_4 (01000) to P_5. After acquiring m_3, R_5 becomes (11000).In this case, if P_3 starts DRL-collation at t_1, it will compute the tentative minimal set equivalent to R_3(00111), which comes out to be {P_1, P_2, P_3}. In this way, partial transitive interdependencies are captured during normal computations.

In coordinated DRL-collation, if a single proceeding miscarries to capture its recuperation-point; all the DRL-collation work goes waste, because, each proceeding has to abandon its partially-persistent recuperation-point. Furthermore, in order to capture the partially-persistent recuperation-point, a M_Nodule prerequisites to transfer large recuperation-point data to its resident Mobl_Supp_St over mobile channels. Hence, the loss of DRL-collation work may be exceedingly high due to frequent terminations of DRL-collation strategies especially in mobile interconnection. In mobile distributed interconnection, there remain certain concerns like: abrupt cessation, exhausted battery power, or letdown in mobile bandwidth. So there remains a good probability that some M_Nodule may fail to capture its recuperation-point in harmonization with others. Therefore, we advocate that in the first stage, all proceedings in the minimal set, capture evanescent recuperation-point only. Evanescent recuperation-point is stored on the memory of M_Nodule only. If some proceeding miscarries to capture its recuperation-point in the first stage, then other M_Nodules need to abandon their evanescent recuperation-points only.

The work of arresting an evanescent recuperation-point is negligible as compared to the partially-persistent one. In other strategies, all pertinent proceedings need to abandon their partially-persistent recuperation-points in this situation. Hence the loss of DRL-collation work in case of an abandon of the DRL-collation arrangement is dramatically low in the advocated scheme as compared to other coordinated DRL-collation schemes for mobile distributed interconnection.

In this second stage, a proceeding renovates its evanescent recuperation-point into partially-persistent one. By using this scheme, we try to moderate the loss of DRL-collation work in case of abandon of DRL-collation arrangement in the first stage.

A non-intrusion DRL-collation arrangement does not require any proceeding to interrupt its underlying computation. When proceedings do not suspend their computation, it is possible for a proceeding to acquire a computation communication from another proceeding, which is already running in a new DRL-collation interval. If this situation is not properly dealt with, it may result in an inconsistency. During the DRL-collation arrangement, a proceeding Pi may acquire m from Pj such that Pj has captured its recuperation-point for the current instigation whereas Pi has not. Suppose, Pi proceedings m, and it acquires recuperation-point request later on, and then it captures its recuperation-point. In that case, m will become orphan in the captured comprehensive status. We advocate that only those communications, which can become orphan, should be buffered at the forwarder's end. When a proceeding captures its evanescent recuperation-point, it is not allowed to forward any communication till it acquires the partially-persistent recuperation-point request. However, in this interval, the proceeding is allowed to perform its normal computations and acquire the communications. When a proceeding acquires the partially-persistent recuperation-point request, it is confirmed that every pertinent proceeding has captured its evanescent recuperation-point. Hence, a communication generated for forwarding by a proceeding after getting partially-persistent recuperation-point request cannot become orphan. Hence, a proceeding can forward the buffered communications after getting the partially-persistent recuperation-point request from the inaugurator.

3. The Proposed Coordinated DRL-collation Arrangement

First stage of the arrangement:When a proceeding, say P_i, running on a M_Nodule, say M_Nodulei, triggers a DRL-collation, it forwards a recuperation-point instigation request to its resident Mobl_Supp_St, which will be the proxy Mobl_Supp_St (if the inaugurator runs on an Mobl_Supp_St, then the Mobl_Supp_St is the proxy Mobl_Supp_St). The proxy Mobl_Supp_St maintains the interdependency array of Pi say Ri. On the basis of Ri, the set of dependent proceedings of Pi is formed, say Smin. The proxy Mobl_Supp_St broadcasts ckpt (Smin) to all Mobl_Supp_Sts. When an Mobl_Supp_St acquire ckpt (Smin) communication, it checks, if any proceedings in Smin are in its cubicle. If so, the Mobl_Supp_St forwards evanescent recuperation-point request communication to them. Any proceeding acquiring an evanescent recuperation-point request captures an evanescent recuperation-point and forwards a response to its resident

Mobl_Supp_St. After an Mobl_Supp_St received all response communications from the proceedings to which it dispatched evanescent recuperation-point request communications, it forwards a response to the proxy Mobl_Supp_St. It should be noted that in the first stage, all proceedings capture the evanescent recuperation-points. For a proceeding running on a static host, evanescent recuperation-point is equivalent to partially-persistent recuperation-point. But, for a M_Nodule, evanescent recuperation-point is divergent from partially-persistent recuperation-point. In order to capture a partially-persistent recuperation-point, a M_Nodulehas to record its resident status and has to transfer it to its resident Mobl_Supp_St. But, the evanescent recuperation-point is stored on the resident disk of the M_Nodule. It should be noted that the work of arresting an evanescent recuperation-point is very small as compared to the partially-persistent one. For a disconnected M_Nodule that is a member of minimal set, the Mobl_Supp_St that has its disconnected recuperation-point, considers its disconnected recuperation-point as the necessitated come.

Second Stage of the arrangement:After the proxy Mobl_Supp_St has received the response from every Mobl_Supp_St, the arrangement enters the second stage. If the proxy Mobl_Supp_St learns that all applicable proceedings have captured their evanescent recuperation-points successfully, it directs them to convert their evanescent recuperation-points into partially-persistent ones and also forwards the exact minimal set along with this request. Alternatively, if inaugurator Mobl_Supp_St comes to know that some proceeding has failed to capture its recuperation-point in the first stage, it sends abandon request to all Mobl_Supp_St. In this way the M_Nodules need to abandon only the evanescent recuperation-points, and not the partially-persistent ones. In this way we try to reduce the loss of DRL-collation work in case of abandon of DRL-collation arrangement in first stage.

When an Mobl_Supp_St acquires the partially-persistent recuperation-point request, it directs all the proceeding in the minimal set, which are also running in itself, to convert their evanescent recuperation-points into partially-persistent ones. When an Mobl_Supp_St learns that all applicable proceeding in its cubicle have captured their partially-persistent recuperation-points successfully, it forwards response to proxy Mobl_Supp_St. If any M_Nodule miscarries to transfer its recuperation-point data to its resident Mobl_Supp_St, then the letdown response is dispatched to the proxy Mobl_Supp_St; which in turn, issues the abandon communication.

Third Stage of the arrangement : Finally, when the proxy Mobl_Supp_St learns that all proceedings in the minimal set have captured their partially-persistent recuperation-points successfully, it sends commit request to all Mobl_Supp_Sts. When a proceeding in the minimal set gets the commit request, it renovates its partially-persistent recuperation-point into persistent one and discards its earlier persistent recuperation-point, if any.

4. Communication Handling During DRL-collation:

When a proceeding captures its evanescent recuperation-point, it does not forward any communication till it acquires the partially-persistent recuperation-point request. This time interval of a proceeding is called its improbability period. Suppose, P_i forwards m to P_j after arresting its evanescent recuperation-point and P_j has not captured its evanescent recuperation-point at the time of acquiring m. In this case, if P_j captures its evanescent recuperation-point after working m, then m will become orphan. Therefore, we do not allow Pi to forward any communication unless and until every proceeding in the minimal set have captured its evanescent recuperation-point in the first stage. P_i can forward communications when it acquires the partially-persistent recuperation-point request; because, at this moment every pertinent proceeding has captured its evanescent recuperation-point and m cannot become orphan. The communications to be dispatched are buffered at forwarders end. In this interval, a proceeding is allowed to continue its normal computations and acquire communications.

Suppose, Pj gets the evanescent recuperation-point request at Mobl_Supp_Stn. Now, we find any proceeding Pk such that Pk does not belong to Smin and Pk belongs to Rj[]. In this case, Pk is also encompassed in the minimal set; and Pj forwards evanescent recuperation-point request to Pk. It should be noted that the Smin, computed on the basis of interdependency array of inaugurator proceeding is only a subset of the minimal set. Due to zigzag interdependencies, inaugurator proceeding may be transitively dependent upon some more proceedings which are not encompassed in the Smin computed initially.

5. An Example of the Proposed Protocol

The advocated arrangement can be better understood by the example shown in Figure 2. There are six proceedings (P0 to P5) denoted by straight lines. Each proceeding is assumed to have initial persistent recuperation-points with chkpt_seq_no equal to “0”. Cix denotes the xth recuperation-points of Pi. Initial interdependency arrays of P0, P1, P2, P3, P4, P5 are [000001], [000010] [000100], [001000], [010000], and [100000], respectively.

P0 forwards m2 to P1 along with its interdependency array [000001]. When P1 acquires m2, it computes its interdependency array by arresting bitwise logical OR of interdependency arrays of P0 and P1, which comes out to be [000011]. Similarly, P2 updates its interdependency array on acquiring m3 and it comes out to be [000111]. At time t1, P2 triggers DRL-collation arrangement with its interdependency array is [000111]. At time t1, P2 discovers that it is transitively dependent upon P0 and P1. Therefore, P2 computes the partially-persistent minimal set [Smin= {P0, P1, P2}]. P2 forwards the evanescent recuperation-point request to P1 and P0 and captures its own evanescent recuperation-point C21. For a M_Nodule the evanescent recuperation-point is stored on the disk of M_Nodule. It should be noted that Smin is only a subset of the minimal set. When P1 captures its evanescent recuperation-point C11, it discovers that it is dependent upon P3 due to m4, but P3 is not a member of Smin; therefore, P1 forwards evanescent recuperation-point request to P3. Consequently, P3 captures its evanescent recuperation-point C31.

After arresting its evanescent recuperation-point C21, P2 generates m8 for P3. As P2 has already captured its evanescent recuperation-point for the current instigation and it has not received the partially-persistent recuperation-point request from the inaugurator; therefore P2 buffers m8 on its resident disk. We define this interval as the improbability period of a proceeding during which a proceeding is not allowed to forward any communication. The communications generated for forwarding are buffered at the resident disk of the forwarder’s proceeding. P2 can forwards m8 only after getting partially-persistent recuperation-point request or abandon communications from the inaugurator proceeding. Similarly, after arresting its evanescent recuperation-point P0 buffers m10 for its improbability period. It should be noted that P1 acquires m10 only after arresting its evanescent recuperation-point. Similarly, P3 acquires m8 only after arresting its evanescent recuperation-point C31.A proceeding is allowed to acquire all the communications during its improbability period; for example, P3 acquires m11. A proceeding is also allowed to perform its normal computations during its improbability period.

At time t2, P2 acquires responses to evanescent recuperation-points requests from all proceeding in the minimal set (not shown in the Figure 2) and discovers that they have captured their evanescent recuperation-points successfully; therefore, P2 sends partially-persistent recuperation-point request to all proceedings. On getting partially-persistent recuperation-point request, proceedings in the minimal set [P0, P1, P2, P3] convert their evanescent recuperation-points into partially-persistent ones and forward the response to inaugurator proceeding P2; these proceeding also forward the communications, buffered at their resident disks, to the destination proceedings For example, P0 forwards m10 to P1 after getting partially-persistent recuperation-point request [not shown in the figure]. Similarly, P2 forwards m8 to P3 after getting partially-persistent recuperation-point request. At time t3, P2 acquires responses from the proceeding in minimal set [not shown in the figure] and discovers that they have captured their partially-persistent recuperation-points successfully, therefore, P2 directs commit request to all proceeding. A proceeding in the minimal set renovates its partially-persistent recuperation-point into persistent recuperation-point and abandons it old persistent recuperation-point if any.

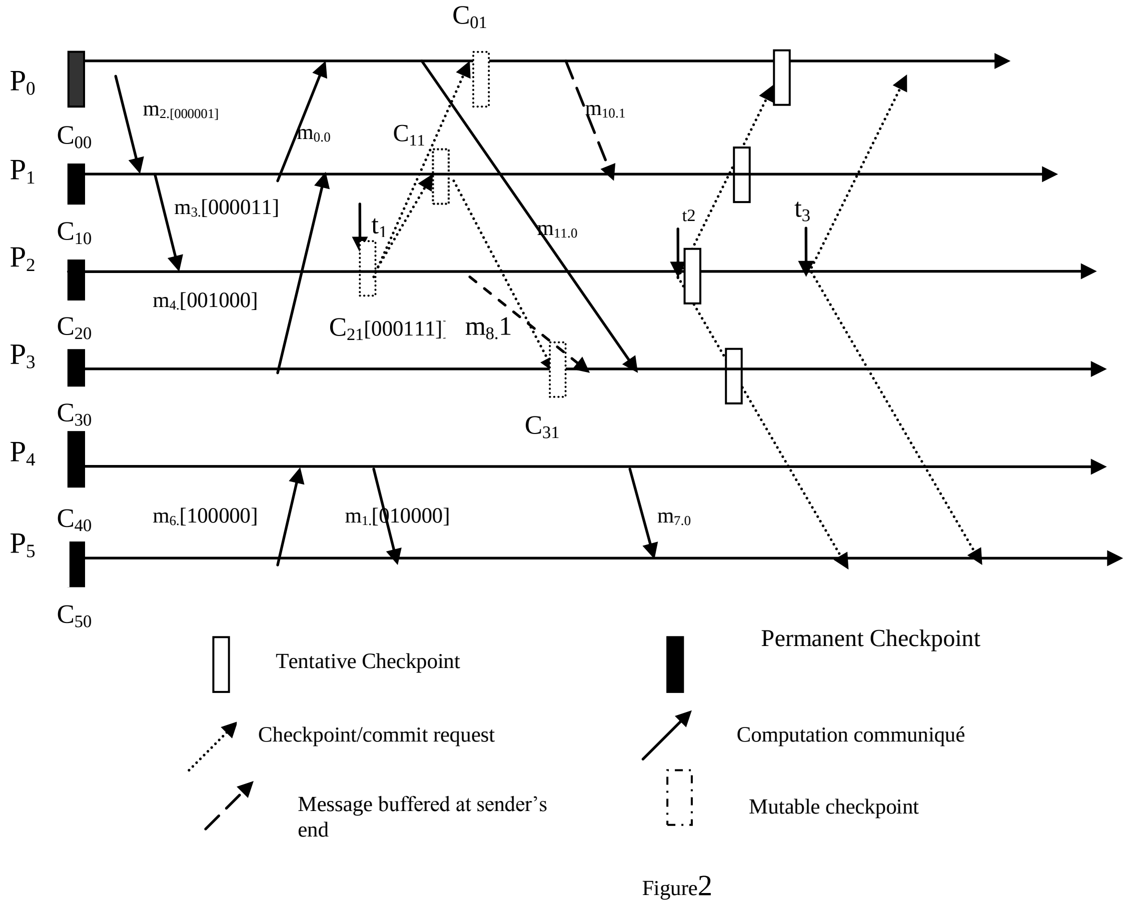

Figure2

6. Conclusion

We have designed a minimal-proceeding synchronous DRL-collation mechanism for mobile distributed interconnection. We try to minimize the intrusion of proceedings during DRL-collation . The intrusion time of a proceeding is quite minimal. During intrusion period, proceedings can do their normal working outs, forward computation-communications and process selective computation-communications. The number of proceedings that seize recuperation-points is minimized to circumvent awakening of M_Nodules in doze mode of processing and thrashing of M_Nodules with DRL-collation activity. It also saves limited battery life of M_Nodules and low bandwidth of wireless channels. We try to reduce the loss of DRL-collation effort when any proceeding miscarries to seize its recuperation-point in synchronization with others. We also try to minimize the synchronization computation-communications during DRL-collation.

References:-

[1] Acharyaand 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] R. Baldoni, J-M Hélary, A. Mostefaoui and M. Raynal, A Communication-Induced Checkpointing Protocol that Ensures Rollback-Dependency Tractability, In Proceedings of the International Symposium on Fault-Tolerant-Computing Systems, 1997, 68-77.

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

[6] 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.

[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 NonblockingCheckpointing, 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] Gao Y., Deng C., &Che, Y.,“ An Adaptive Index-Based Algorithm Using Time-Coordination in Mobile Computing”, International Symposiums on Information Processing, pp.578-585,

2008.

[29] Praveen Choudhary, Parveen Kumar,” Minimum-Process Global-Snapshot Accumulation Etiquette for Mobile Distributed Systems ”, International Journal of Advanced Research in Engineering and Technology” Vol. 11, Issue 8, Aug 20, pp.937-948

[30] 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