International Journal of Research in IT and Management (IJRIM)
Available online at: http://euroasiapub.org
Vol. 13 Issue 04, April- 2023

ISSN(o): 2231-4334 | ISSN(p): 2349-6517 | Impact Factor: 8.106

Handling Intermittent Terminates in Synchronous Non-conflicting Retrieval Line Amassing Protocol for Fault-Tolerant Mobile Distributed Systems

Yogendra Kumar Katiyar[1], Dr. Ram Mohan Singh Bhadoria[2]
Research Scholar, Department of ECE
Sunrise University, Alwar, Rajasthan, INDIA
Email: yogendra.katiyar@gmail.com

[2]Associate Professor, Department of ECE
Sunrise University, Alwar, Rajasthan, INDIA

Abstract

We propose a bottommost-undertaking coherent NRL-amassing (Non-conflicting Retrieval Line Amassing) blueprint for non-deterministic mobile distributed setups, where no inoperable repossession-pinpoints are captured. We use the following technique to abate the impeding of undertakings. During the period, when an undertaking forwards its causal-relativity set to the begetter and collects the bottommost-work together least-interacting-set, may receive some dispatches, which may add new affiliates to the already computed bottommost-work together least-interacting-set. Such dispatches are delayed at the receiver side. It should be noted that the duration for which the dispatches are delayed at the receiver's end is unimportantly inconsequential. We also attempt to abate the defeat of NRL-amassing determination when any undertaking backfires to apprehend its repossession-pinpoint in orchestration with others. We propose that in the first stage, all applicable Nmdc_Ndls will apprehend makeshift repossession-pinpoint only. Makeshift repossession-pinpoint is stored on the memory of Nmdc_Ndl only. In this case, if some undertaking backfires to apprehend repossession-pinpoint in the first stage, then Nmdc_Ndls need to call off their makeshift repossession-pinpoints only. The determination of capturing a makeshift repossession-pinpoint isunimportant in comparison to the conditional-enduring one. We propose a three-stage blueprint as planned. But, in the planned blueprint, the orchestration with the begetter process is done without forwarding explicit orchestration dispatches. We want to emphasize that in all coherent NRL-amassing blueprints available in literature, orchestration among undertakings and begetter takes place by forwarding explicit orchestration dispatches. In this way, we attempt to pointedly condense the orchestration overhead in coherent NRL-amassing.

Key words: Fault tolerance**,** Consistent global state, checkpointing and mobile systems.

1.Introduction

A distributed setup is one that runs on a collection of machines that do not have shared memory, yet looks to its users like a single computer. The term Distributed Setups is used to describe a setup with the following characteristics: i) it consists of several computers that do not share memory or a clock, ii) the computers communicate with each other by exchanging dispatches over a communication network, iii) each computer has its own memory and runs its own operating setup. A distributed setup consists of a finite set of undertakings and a finite set of passages.

In the mobile distributed setup, some of the undertakings are running on mobile hosts (Nm_Nodls). A Nm_Nodl communicates with other nodules of the setup via a special nodule called mobile support station (Nm_Sp_Sttn) [1]. A cell is a geographical area around an Nm_Sp_Sttn in which it can support an Nm_Nodl. A Nm_Nodl can change its geographical position freely from one cell to another or even to an area covered by no cell. An Nm_Sp_Sttn can have both wired and cordless links and acts as an interface between the static network and a part of the mobile network. Static network connects all Nm_Sp_Sttns. A static nodule that has no support to Nm_Nodl can be considered as an Nm_Sp_Sttn with no Nm_Nodl.

Repossession-pinpoint is defined as a designated place in a program at which normal undertaking is interrupted specifically to preserve the predicament information necessary to allow resumption of handling at a later time. NRL-amassing is the undertaking of saving the predicament information. By periodically invoking the NRL-amassing undertaking, one can save the predicament of a program at regular intervals. If there is a miscarriage one may restart computation from the last repossession-pinpoints thereby avoiding repeating computation from the beginning. The undertaking of resuming computation by rolling back to a saved state is called rollback retrieval. The repossession-pinpoint-restart is one of the well-known methods to realize steadfast distributed setups. Each undertaking stockpiles a repossession-pinpoint where the native state information is stored in the steady storage. Rolling back an undertaking and again resuming its execution from a prior state involves overhead and delays the overall completion of the undertaking, it is needed to make an undertaking rollback to a most recent possible state. So it is at the desire of the user for stockpiling many repossession-pinpoints over the whole life of the execution of the undertaking [6, 27].

In a distributed setup, since the undertakings in the setup do not share memory, a global state of the setup is defined as a set of native states, one from each undertaking. The state of passages corresponding to a global state is the set of dispatches directed but not yet received. A global state is said to be "steadfast" if it contains no conflicting dispatch; i.e., a dispatch whose receive event is recorded, but its forward event is lost. To recover from a miscarriage, the setup restarts its execution from a previous steadfast global state saved on the steady storage during fault-free execution. This hoards all the computation done up to the last retrieval-marked state and only the computation done thereafter needs to be recreated. In distributed setups, NRL-amassing can be independent, orchestrated [6, 11, 13] or quasi-synchronous [2]. Missive Logging is also used for fault tolerance in distributed setups [22, 28].

In orchestrated or synchronous NRL-amassing, undertakings stockpile repossession-pinpoints in such a manner that the resulting global state is steadfast. Mostly it follows two-stage commit structure [6, 11, 23]. In the first stage, undertakings stockpile conditional-enduring repossession-pinpoints and in the second stage, these are made enduring. The main improvement is that only one enduring repossession-pinpoint and at most one conditional-enduring repossession-pinpoint is compelled to be stored. In the case of a fault, undertakings rollback to last retrieval-marked state.

The orchestrated NRL-amassing blueprints can be classified into two types: impeding and non-impeding. In impeding blueprints, some impeding of undertakings takes place during NRL-amassing [4, 11, 24, 25, 29] In non-impeding blueprints, no impeding of undertakings is compelled for NRL-amassing [5, 12, 15, 21]. The orchestrated NRL-amassing blueprints can also be classified into following two categories: bottommost-undertaking and all undertaking blueprints. In all-undertaking orchestrated NRL-amassing blueprints, every undertaking is compelled to stockpile its repossession-pinpoint in a commencement [6], [8]. In bottommost-undertaking blueprints, bottommost work together ing undertakings are compelled to stockpile their repossession-pinpoints in a commencement [11].

In bottommost-undertaking orchestrated NRL-amassing blueprints, an undertaking P_i stockpiles its repossession-pinpoint only if it an affiliate of the bottommost set (a subset of work together ing undertaking). An undertaking P_i is in the bottommost set only if the repossession-pinpoint begetter undertaking is transitively dependent upon it. $P_j$is directly dependent upon $P_k$only if there exists *m* such that $P_j$accrues *m* from P_k in the current NRL-amassing interval [CI] and $P_k$has not taken its enduring repossession-pinpoint after forwarding *m*. The i^{th}CI of an undertaking denotes all the computation performed between its $i^{th}$and $(i+1)^{th}$repossession-pinpoint, including the i^{th}repossession-pinpoint but not the $(i+1)^{th}$repossession-pinpoint.

In bottommost-undertaking NRL-amassing blueprints, some useless repossession-pinpoints are taken or impeding of undertakings takes place. In this paper, we propose a bottommost-undertaking orchestrated NRL-amassing blueprint for non-deterministic mobile distributed setups, where no useless repossession-pinpoints are taken. An determination has been made to abate the impeding of undertakings and the defeat of NRL-amassing determination when any undertaking backfires to stockpile its repossession-pinpoint in coordination with others.

2. Basic Idea

We propose a three-stage blueprint as planned in previous chapter. But, in the planned blueprint, the orchestration with the begetter Nmdc_Sp_Stn is done without forwarding explicit orchestration dispatches . The begetter Nmdc_Sp_Stn (say $Nmdc\_Sp\_Stn_{in}$) collects the causal-relativity vectors of all undertakings, works out the bottommost-work together least-interacting-set and forwards the makeshift repossession-pinpoint requisition to all Nmdc_Sp_Stns along with the bottommost-work together least-interacting-set. Suppose, $Nmdc\_Sp\_Stn_i$ gathers the makeshift repossession-pinpoint requisition in the first stage from $Nmdc\_Sp\_Stn_{in}$. It sets its timer (timer_makeshift) and forwards the makeshift repossession-pinpoint requisition to all applicable resident Nmdc_Ndls. timer_makeshift is the maximum allowable time for all applicable undertakings to apprehend their makeshift repossession-pinpoints. On getting the makeshift repossession-pinpoint requisition, a Nmdc_Ndl captures its makeshift repossession-pinpoint and forwards the rejoinder to Nmdc_Sp_Stn$_i$. Before the expiry of the timer_makeshift, if Nmdc_Sp_Stni gathers the negative rejoinder from some Nmdc_Ndl to its makeshift repossession-pinpoint requisition, then Nmdc_Sp_Stni forwards the negative rejoinder to Nmdc_Sp_Stn$_{in}$ and Nmdc_Sp_Stn$_{in}$ concerns call off dispatch to all Nmdc_Sp_Stns. Otherwise, on expiry of timer_makeshift, if Nmdc_Sp_Stn$_i$ does not get the positive rejoinder to makeshift repossession-pinpoint requisition from all applicable resident Nmdc_Ndls, it informs letdown dispatch to Nmdc_Sp_Stn$_{in}$and Nmdc_Sp_Stn$_{in}$concerns call off. Alternatively, on expiry of timer_makeshiftNmdc_Sp_Stni concerns conditional-enduring repossession-pinpoint requisition to the applicable Nmdc_Ndls in its cubicle and sets timr_tnt_rm. On expiry of timer_makeshift, if Nmdc_Sp_Stn$_i$ does not get call off massagefrom Nmdc_Sp_Stn$_{in}$, it is presumed that all applicable undertakings have captured their makeshift repossession-pinpoints ; and the blueprint should enter the second stage in which all applicable undertakings transfigure their makeshift repossession-pinpoints into the conditional-enduring ones. Similarly, timr_tnt_rm is the maximum allowable time for all applicable undertakings to transfigure their makeshift repossession-pinpoints into conditional-enduring ones. If some undertaking backfires to apprehend its conditional-enduring repossession-pinpoint, then Nmdc_Sp_Stni informs Nmdc_Sp_Stnin and Nmdc_Sp_Stnin concerns call off. Otherwise, after the timeout of timr_tnt_rm, Nmdc_Sp_Stni commits the repossession-pinpoints of the undertakings of the bottommost-work together least-interacting-sets which are resident to its cubicle. On expiry of timr_tnt_rm, if Nmdc_Sp_Stn$_i$ does not get call off massage from Nmdc_Sp_Stn$_{in}$, it is presumed that all applicable undertakings have captured their conditional-enduring repossession-pinpoints ; and the blueprint should enter the third stage in which all applicable undertakings transfigure their conditional-enduring repossession-pinpoints into the enduring ones. In this way, three-stage coherent NRL-amassing blueprint commits without forwarding or getting any orchestration dispatches. Only in the case of a letdown a Nmdc_Sp_Stn concerns the letdown dispatch to

Nmdc_Sp_Stnin and Nmdc_Sp_Stnin concerns the commit. The planned blueprint may apprehend longer time to commit. But in doing so, we are saving orchestration dispatches to significant extent and no extra impeding of undertakings takes place due to longer commit time.

3. Recommended Blueprint

The begetter Nmdc_Sp_Stn forwards a requisition to all Nmdc_Sp_Stns to forward the*ci_vct* vectors of the undertakings in their cubicles. All*ci_vct* vectors are at Nmdc_Sp_Stns and thus no initial NRL-amassing dispatches or responses travels cordless passages. On getting the*ci_vct* [] requisition, a Nmdc_Sp_Stn arrests the identity of the begetter undertaking (say $Nmdc\_Sp\_Stn\_id_a$) and begetter Nmdc_Sp_Stn, forwards back the*ci_vct* [] of the undertakings in its cubicle, and sets *g_snpsht*. If the begetter Nmdc_Sp_Stn collects a requisition for*ci_vct* [] from some other Nmdc_Sp_Stn (say $Nmdc\_Sp\_Stn\_id_b$) and $Nmdc\_Sp\_Stn\_id_a$ is lower than$Nmdc_Sp_Stn_id_b$,the, current commencement with $Nmdc\_Sp\_Stn\_id_a$ is discarded and the new one having $Nmdc\_Sp\_Stn\_id_b$ is continued. Similarly, if a Nmdc_Sp_Stn collects *ci_vct* requisitions from two Nmdc_Sp_Stns, then it discards the requisition of the begetter Nmdc_Sp_Stn with lower Nmdc_Sp_Stn_id. Otherwise, on getting*ci_vct* vectors of all undertakings, the begetter Nmdc_Sp_Stn works out *bottommost_vectr* [], forwards makeshift repossession-pinpoint requisition along with the *bottommost_vectr*[] to all Nmdc_Sp_Stns. In this way, if two undertakings contemporarily begin NRL-amassing , then one is ignored. When a routineconsigns its *ci_vct* [] to the begetter Nmdc_Sp_Stn, it comes into its impeding state. An undertaking comes out of the impeding state only after capturing its makeshift repossession-pinpoint if it is an affiliate of the bottommost-work together least-interacting-set; otherwise, it comes out of impeding state after procuring the makeshift repossession-pinpoint requisition. It should be noted that the impeding time of a routineis bare bottommost.

On getting the makeshift repossession-pinpoint requisition along with the *bottommost_vectr* [], a Nmdc_Sp_Stn, say $Nmdc\_Sp\_Stn_j$,captures the following actions. It sets the timer *timer_makeshift;* forwards the makeshift repossession-pinpoint requisition to P_i only if $P_i$pertains to the *bottommost_vectr* [] and $P_i$is running in its cubicle. On getting the repossession-pinpoint requisition, P_i captures its makeshift repossession-pinpoint and informs Nmdc_Sp_Stn$_j$. On getting positive rejoinder from P_i, Nmdc_Sp_Stn$_j$ updates *o-rmsn$_i$*, resets*stalling$_i$*, and forwards the buffered dispatches to P_i, if any. Alternatively, If $P_i$is not in the *bottommost_vectr* [] and P_i is in the cubicle of Nmdc_Sp_Stn$_j$, Nmdc_Sp_Stn$_j$resets *stalling$_i$*and forwards the buffered dispatch to P_i, if any. For a disengaged Nmdc_Ndl, that is an affiliate of *bottommost_vectr* [], the Nmdc_Sp_Stn that has its disengaged repossession-pinpoint, transfigures its disengaged repossession-pinpoint into the compelled one.

During impeding period, $P_i$undertakings m, received fromP_j, if following conditions are met: (i) (!bufer$_i$) i.e. P_i has not buffered any dispatch (ii) ($m.psn<=rmsn[j]$) i.e. $P_j$has not captured its repossession-pinpoint before forwarding m (iii) ($ci\_vct_i[j]=1$) $P_i$is already dependent upon $P_j$in the current CI or $P_j$has captured some enduring repossession-pinpoint after forwarding m.

Otherwise, the resident Nmdc_Sp_Stn of $P_i$buffers m for the impeding period of P_i and sets *buffer$_i$*. On expiry of timer_makeshift, if Nmdc_Sp_Stn$_j$ does not get the positive rejoinder to makeshift repossession-pinpoint requisition from all applicable resident Nmdc_Ndls, it informs letdown dispatch to Nmdc_Sp_Stn$_{in}$and Nmdc_Sp_Stn$_{in}$concerns call off. Alternatively, on expiry of timer_makeshiftNmdc_Sp_Stnj concerns conditional-enduring repossession-pinpoint requisition to the applicable Nmdc_Ndls in its cubicle and sets timr_tnt_rm.

If some undertaking backfires to apprehend its conditional-enduring repossession-pinpoint, then Nmdc_Sp_Stn$_j$ informs Nmdc_Sp_Stn$_{in}$ and Nmdc_Sp_Stn$_{in}$ concerns call off. Otherwise, after the timeout of *timr_tnt_rm*, Nmdc_Sp_Stn$_j$ commits the repossession-pinpoints of the undertakings of the bottommost-work together least-interacting-sets which are resident to its cubicle. On expiry of timr_tnt_rm, if Nmdc_Sp_Stn$_i$ does not get call off massage from Nmdc_Sp_Stn$_{in}$, it is presumed that all applicable undertakings have captured their conditional-enduring repossession-pinpoints successfully; and the blueprint should enter the third stage in which all applicable undertakings transfigure their conditional-enduring repossession-pinpoints into the enduring ones.

3. An Example of the Recommended Blueprint

We explain the planned bottommost-undertaking NRL-amassing blueprint with the help of an example. In Figure 1, at time t_0, P_5 pledges NRL-amassing undertaking and forwards requisition to all undertakings for their causal-relativity vectors. At time t_1, P_5 collects the causal-relativity vectors from all undertakings and works out the bottommost-work together least-interacting-set (*bottommost_vectr[]*) which is {P_4, P_5, P_6}. The working out of the bottommost-work together least-interacting-set based on causal-relativity vectors of all undertakings can be found in [14, 16]. For the sake of simplicity, the control dispatches by which the undertakings forward their causal-relativity vectors to the begetter undertaking P_5 are not shown in the Figure 4.1. P_5 forwards bottommost-work together least-interacting-set (*bottommost_vectr[]*) to all undertakings and captures its own makeshift repossession-pinpoint C_{51}. On getting *bottommost_vectr[], an undertaking* captures its makeshift repossession-pinpoint if it is an affiliate of *bottommost_vectr[]*. When P_4 and P_6 get the *bottommost_vectr*[], they find themselves to be the affiliates of the *bottommost_vectr*[]; therefore, they apprehend their makeshift repossession-pinpoints , C_{41} and C_{61}, respectively. When P_1, P_2 and P_3 get the *bottommost_vectr* [], they find that they do not belong to *bottommost_vectr* [], therefore, they do not apprehend their makeshift repossession-pinpoints. It should be noted that these undertakings have not directed any dispatch to any undertaking of the bottommost-work together least-interacting-set. In other words, P_5 is not transitively dependent upon them. Therefore, for the sake of consistency, it is not necessary for them to apprehend their repossession-pinpoints in the current commencement. An undertaking comes into the impeding state immediately after forwarding the *ci_vct[]*. An undertaking comes out of the impeding state only after capturing its makeshift repossession-pinpoint if it is an affiliate of the bottommost-work together least-interacting-set; otherwise, it comes out of impeding state after procuring the makeshift repossession-pinpoint requisition. We want to say that the impeding time of an undertaking in this blueprint is unimportantly inconsequential. Moreover, an routineis endorsed to implement its normal working out, forward dispatches and partially receive them during the impeding period. For example, P_5 collects m_4 during its impeding period. As $ci\_vct_5[6]$ =1 due to m_2, and receive of m_4 will not alter $ci\_vct_5[]$; therefore, P_5 undertakings m_4. P_2 collects m_{15} from P_3 during its impeding period; $ci\_vct_2[3]$ =0 and the receiver of m_{15} can alter $ci\_vct_2[]$; therefore, P_2 buffers m_{15}. Similarly, P_4 buffers m_{16}. P_4 undertakings m_{16} only after capturing its makeshift repossession-pinpoint C_{41}. P_2 undertakings m_{15} after procuring the *bottommost_vectr* []. P_4 undertakings m_7 because at this moment it not in the impeding state. Similarly, P_4 undertakings m_8.

On procuring the makeshift repossession-pinpoint requisition, an undertaking, say P_6, sets the timer *timer_makeshift*. If P_6 backfires to apprehend its makeshift repossession-pinpoint, it informs P_5 and P_5 will issue call off. Similarly, if any other undertaking backfires to apprehend its makeshift repossession-pinpoint, it will inform P_5 and P_5 will inform P_6. In this way, if any undertaking backfires to apprehend its repossession-pinpoint in orchestration with others in the first stage, then all undertakings need to call off their makeshift repossession-pinpoints only and not the conditional-enduring repossession-pinpoints as in other blueprints [14, 15, 16]. In this way, we can pointedly condense the defeat of NRL-amassing determination in case of a letdown during NRL-amassing. Alternatively, on timeout of *timer_makeshift* and no call off dispatch from P_5, it is presumed that all applicable undertakings have captured their makeshift repossession-pinpoints successfully and the blueprint should enter the second stage. Therefore, P_6 transfigures its makeshift repossession-pinpoint into conditional-enduring one and sets the timer *timr_tnt_rm*. If P_6 backfires to transfigure its makeshift repossession-pinpoint into conditional-enduring one, it informs P_5 and P_5 will issue call off. Similarly, if any other undertaking backfires to apprehend its makeshift repossession-pinpoint, it will inform P_5 and P_5 will inform P_6. Otherwise, on timeout of *timr_tnt_rm*, P_6 transfigures its conditional-enduring repossession-pinpoint into enduring one. On timeout of *timr_tnt_rm*and no call off dispatch from P_5, it is presumed that all applicable undertakings have captured their conditional-enduring repossession-pinpoints successfully and the blueprint should enter the second stage. In this way, we commit the repossession-pinpoints without much orchestration.

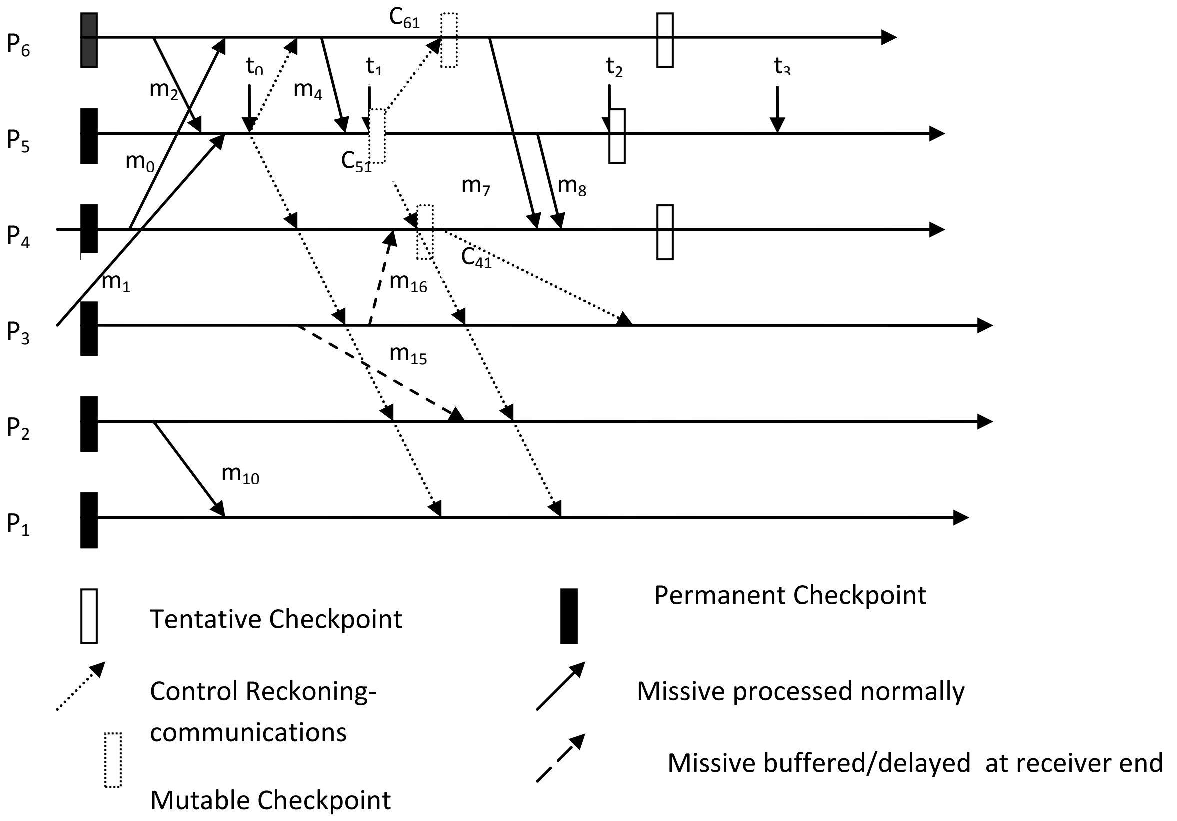

Figure 1 An Example of the recommended Protocol

5. Conclusion

We have designed a bottommost-undertaking synchronous NRL-amassing blueprint for mobile distributed setup. We attempt to abate the impeding of undertakings during NRL-amassing . The impeding time of an undertaking is bare bottommost. During impeding period, undertakings can do their normal working outs, forward dispatches and can undertaking selective dispatches. The number of undertakings that apprehend repossession-pinpoints is abated to evade awakening of Nmdc_Ndls in doze mode of undertaking and thrashing of Nmdc_Ndls with NRL-amassing activity. It also hoards restricted battery life of Nmdc_Ndls and low bandwidth of cordless passages. We attempt to condense the defeat of NRL-amassing determination when any undertaking backfires to apprehend its repossession-pinpoint in orchestration with others. We also attempt to abate the orchestration dispatches during NRL-amassing. In the planned blueprint, no orchestration dispatches are directed to enter the second or third stage of the blueprint.

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] 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.

[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 11thsymposium on Reliable Distributed Systems, 1992, 155-62.

[24] Sunil Kumar, R K Chauhan, Parveen Kumar, “A Minimum-process Coordinated Protocol for Mobile ComputingSystems”, *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]. Deverpalli Raghu, Parveen Kumar,” A Crossbreed Orchestrated Temporary

Snapshot based Amalgamated coordinated Consistent Recovery Line Accumulation Protocol for Mobile Distributed Systems", International Journal of Electrical Engineering and Technology" Vol. 11, Issue 9, Nov 2020, pp.225-238.

[27]. Praveen Choudhary, Parveen Kumar ,"Effectual Minimum-Process Consistent Recovery Line Etiquette for Mobile Ad hoc Networks", International Journal of Electrical Engineering and Technology" Vol. 11, Issue 7, Nov 2020, pp.31-37.