Communications on Applied Nonlinear Analysis
ISSN: 1074-133X
Vol 32 No. 2s (2025)

A Creative Crossbred Scheme for Accumulation of Synchronized Validated Recovery Line for Fault-Resilient Nomadic Distributed Frameworks

Divya Sharma[1], Dr. Surendra Pal Singh[2]

[1]Research Scholar, Department of Computer Science and Engineering,

NIMS Institute of Engineering and Technology (NIET)

NIMS University, Rajasthan, Jaipur, Rajasthan, INDIA

Email: ssharma.divya29@gmail.com

[2]Research Guide, Department of Computer Science and Engineering,

NIMS Institute of Engineering and Technology (NIET)

NIMS University, Rajasthan, Jaipur, Rajasthan, INDIA

Email: drspsingh2511@gmail.com

Article History:

Received: 03-01-2025

Revised: 30-02-2025

Accepted: 20-03-2025

Abstract:

Nomadic distributed frameworks raise new concerns such as suppleness, low bandwidth of cellular pathways, discontinuations, restricted battery power and dearth of steadfast steady stowage on nomadic nodules. To refine both matrices, the VRL-agglomeration (Validated Recovery Line agglomeration) outlay and the abatement of reckoning on reclamation, we advocate an outcrossed VRL-agglomeration blueprint, wherein an all-implementation orchestrated resurgence-point is amassed after the accomplishment of narrowest-implementation orchestrated VRL-agglomeration blueprint for a fixed aggregate of times. Furthermore, we attempt to abate the information annexed onto each reckoning transmission. For narrowest-implementation VRL-agglomeration, we design an impeding blueprint, where no inoperable resurgence-points are amassed and a determination has been carried out to refine the impeding of implementations. We also manage to shorten the mislaying of VRL-agglomeration striving when any implementation misses to collate its proximate-resurgence-point in consonance with others.

Keywords: Culpability forbearance, unfailing comprehensive predicament, orchestrated Validated Recovery Line Agglomeration and nomadic frameworks.

1. Introduction

A resurgence-point is a native predicament of an implementation saved on steady stowage. In a distributed framework, since the implementations in the framework do not share reminiscence, a comprehensive predicament of the framework is defined as a set of native states, one from each implementation. The predicament of pathways corresponding to a comprehensive predicament is the set of transmissions consigned but not yet incurred. A comprehensive predicament is said to be “unfailing” if it comprehends no orphan transmission; i.e., a transmission whose acquire event is documented, but it’s consign event is lost. To recover from a disappointment, the framework restarts its accomplishment from a previous unfailing comprehensive predicament saved on the steady stowage during

culpability-free accomplishment. This protects all the reckoning done up to the last checkpointed predicament and only the reckoning done thereafter prerequisites to be redone.

A virtuous VRL-agglomeration blueprint for nomadic distributed frameworks should have low outlays on Nm_Hsts and cellular pathways and should circumvent arousing of Nm_Hsts in doze mode operation. The discontinuation of Nm_Hsts should not lead to infinite wait predicament. The blueprint should be non-stalling and should force narrowest aggregate of implementations to amass their native resurgence-points [26]. In narrowest-implementation orchestrated VRL-agglomeration blueprints, some impeding of the implementations occurs [4, 15], or some inoperable resurgence-points are amassed [5, 16, 19].

In the present study, we design an outcrossed orchestrated VRL-agglomeration blueprint for nomadic distributed frameworks, where an all-implementation resurgence-point is amassed after accomplishing narrowest-implementation blueprint for a fixed aggregate of times. By recommending an outcrossed scheme, we try to balance the VRL-agglomeration outlay and the abatement of reckoning on reclamation. We also condense the annexed information onto each reckoning transmission. For narrowest-implementation VRL-agglomeration, we advocate an impeding blueprint, where implementations are endorsed to accomplish their normal reckoning, consign transmissions and partly acquire them during the impeding interlude.

2. Basic Idea

In narrowest-implementation VRL-agglomeration, some implementations, having low interaction activity, may not be included in the narrowest set for several resurgence-point instigations and thus may not advance their reclamation line for a long time. In the case of a reclamation after a culpability, this may lead to their rollback to far earlier checkpointed predicament and the abatement of reckoning at such implementations may be outstandingly elevated. Furthermore, due to scarce possessions of Nm_Hsts, this abatement of reckoning may be undesirable. In all-implementation VRL-agglomeration, reclamation line is progressed for each implementation after every comprehensive resurgence-point but the VRL-agglomeration outlay may be outstandingly elevated, especially in nomadic environments due to frequent resurgence-points. Nm_Hsts utilize the steady stowage at the Nm_Suppt_Sts to store resurgence-points of the Nm_Hsts [1]. Thus, to balance the VRL-agglomeration outlay and the abatement of reckoning on reclamation, we design an outcrossed VRL-agglomeration blueprint for nomadic distributed frameworks, where an all-implementation resurgence-point is amassed after certain aggregate of narrowest-implementation resurgence-points. The aggregate of times, the narrowest-implementation VRL-agglomeration blueprint is accomplished, relies on the particular application and environment and can be fine-tuned.

In orchestrated VRL-agglomeration, an ever-growing integer snp_seq_no is generally annexed onto normal transmissions [9, 29]. We advocate a strategy to refine the dimension of the snp_seq_no. In order to address different VRL-agglomeration intervals, we have replaced integer snp_seq_no with k-bit CI. Integer snp_seq_no is monotonically growing, each time an

implementation amasses its resurgence-point, it increments its snp_seq_no by 1. k-bit CI is used to serve the purpose of integer snp_seq_no. The value of k can be fine-tuned. If we use p-bit CI, we will be able to distinguish only 2^p different CIs and it will be implicitly assumed that no transmission is delivered after 2^p-1 CIs. The lower limit of k is '1' which will lead to CI of '1' bit [18].

In the present study, we assume that all-implementation orchestrated resurgence-point is amassed after the accomplishment of narrowest-implementation blueprint for seven times which requires only three-bit CI. In this case, any restrain of a transmission that extends to more than seven CIs may cause a false resurgence-point [18], i.e., it may trigger a resurgence-point even if a begetter does not trigger VRL-agglomeration activity. Thus, in this blueprint, such restrain prerequisites to be evaded. The limit of supreme restrain interlude of a transmission can be extended to fifteen CIs by using four-bit CI, but it will increase the information annexed onto each reckoning transmission by 1-bit. By using four-bit CI, we have the option of accomplishing narrowest-implementation blueprint for 3, 7 or 15 aggregate of times before amassing an all-implementation resurgence-point. If we use two-bit CI, the supreme restrain of a massage should not exceed three CIs, which seems to be unreasonably small in nomadic frameworks. In this case, narrowest-implementation blueprint prerequisites to be accomplished for three times before amassing an all-implementation resurgence-point.

The narrowest-implementation VRL-agglomeration blueprint is based on upholding of straight causative-interrelationships of implementations. Similar to [4], begetter implementation collects the straight causative-interrelationship vectors of all implementations, computes narrowest set, and consigns the resurgence-point requisition along with the narrowest set to all implementations. In this way, impeding time has been significantly condensed as paralleled to [15].

During the interlude, when an implementation consigns its causative-interrelationship set to the begetter and acquires the narrowest set, may acquire some transmissions, which may amend its causative-interrelationship set, and may add new affiliates to the already computed narrowest set. In order to keep the computed narrowest set intact and to circumvent inoperable resurgence-points as in [16, 19], we advocate to block the implementations for this interlude. We have classified the transmissions, incurred during the impeding interlude, into two types: (i) transmissions that amend the causative-interrelationship set of the consignee implementation (ii) transmissions that do not amend the causative-interrelationship set of the consignee implementation. The former transmissions prerequisite to be restrained at the consignee side. The transmissions of the later type can be handled normally. All implementations can accomplish their normal reckonings and consign transmissions during their impeding interlude. When an implementation annexes a transmission of former type, it does not implementation any transmission till it acquires the narrowest set so as to keep the proper sequence of transmissions incurred. When an implementation acquires the narrowest set, it amasses the resurgence-point, if it is in the narrowest set. After this, it acquires the

cached transmissions, if any. By doing so, impeding of implementations is condensed as paralleled to [4].

In orchestrated VRL-agglomeration , if a solitary implementation washes out to grasp its proximate resurgence-point; all the VRL-agglomeration striving goes deserted, for the reason that, every single implementation has to repeal its quasi-imperishable proximate resurgence-point [2, 3, 4, 10]. Likewise, in order to collate the quasi-imperishable proximate resurgence-point, an Nm_Hst demands to transport colossal proximate resurgence-point data to its proximate Nm_Suppt_St over cellular mediums. Hence, the mislaying of VRL-agglomeration exertion may be remarkably exorbitant due to intermittent repeals, mainly, in nomadic distributed framework. In nomadic distributed framework, there persist certain concerns like: unpredicted decoupling, fatigued battery power, or collapse in cellular transmission capacity. So there rests a virtuous likelihood that some Nm_Hst may wash out to grasp and transport its proximate resurgence-point in consonance with others. For that reason, we put forward that in the first-juncture, all implementations in the meanest_colloborating_set [], grasp fugacious proximate resurgence-point only. Fugacious proximate resurgence-point is stockpiled on the reminiscence of Nm_Hst only. If some implementation miscarries to grasp its proximate resurgence-point in the first juncture, then other Nm_Hsts demand to repeal their fugacious proximate resurgence-points only. The striving of stockpiling a fugacious proximate resurgence-point is inconsequential as paralleled to the quasi-imperishable one. In other etiquettes [2, 3, 4, 10], all pertinent implementations demand to repeal their quasi-imperishable proximate resurgence-points in this predicament of affairs. Hence the mislaying of VRL-agglomeration exertion is melodramatically low in the projected tactic as paralleled to other orchestrated VRL-agglomeration strategies for nomadic distributed framework [2, 3, 4, 10].

In this second-juncture, an implementation alters its fugacious proximate resurgence-point into quasi-imperishable one. By applying this tactic, we manage to curtail the forfeit of VRL-agglomeration work in case of repeal of the tactic in the first juncture.

3. The Planned Narrowest-implementation VRL-agglomeration Blueprint

(a)Resurgence-point Inception

The begetter Nm_Supp_St consigns a requisition to all Nm_Suppt_Sts (Nm_Suppt_Sts of the nomadic framework under consideration) to consign the *SCIA* vectors of the implementations in their cells. All *SCIA* vectors are at Nm_Suppt_Sts and thus no initial VRL-agglomeration transmissions or responses travels cellular pathways. On acquiring the *SCIA*[] requisition, an Nm_Supp_St records the identity of the begetter implementation (say Nm_Suppt_St _id= Nm_Suppt_St $\_id_{in}$) and begetter Nm_Supp_St , consigns back the *SCIA*[] of the implementations in its cubicle, and sets *g_chkpt*. If the begetter Nm_Supp_St acquires a requisition for *SCIA*[] from some other Nm_Supp_St (say Nm_Suppt_St _id= Nm_Suppt_St $\_id_{in2}$) and Nm_Suppt_St $\_id_{in}$ is lower than Nm_Suppt_St $\_id_{in2}$,the, newfangled inception (having Nm_Suppt_St _id= Nm_Suppt_St $\_id_{in}$) is thrown away and the new one (having Nm_Suppt_St _id= Nm_Suppt_St $\_id_{in2}$) is continued. Correspondingly,

if an Nm_Supp_St acquires SCIA requisitions from two Nm_Suppt_Sts, then it discards the requisition of the begetter Nm_Supp_St with lower Nm_Suppt_St _id. Otherwise, on acquiring *SCIA* vectors of all implementations, the begetter Nm_Supp_St computes *narrowest_set*[], consigns resurgence-point requisition to the begetter implementation and consigns resurgence-point requisition along with the *narrowest_set*[] to all Nm_Suppt_Sts.

(b) Reception of a resurgence-point requisition

On acquiring the resurgence-point requisition along with the *narrowest_set*[], an Nm_Supp_St , say Nm_Supp_St$_j$, amasses the succeeding actions. It consigns the resurgence-point requisition to P_i only if P_i belongs to the *narrowest_set*[] and P_i is accomplishing in its cubicle. On acquiring the resurgence-point requisition, P_i amasses its quasi-imperishable resurgence-point and informs Nm_Supp_St$_j$. On acquiring positive response from P_i, Nm_Supp_St$_j$ streamlines cci_i, nci_i, resets $blocking_i$, and consigns the cached transmissions to P_i, if any. Alternatively, If P_i is not in the *narrowest_set*[] and P_i is in the cubicle of Nm_Supp_St$_j$, Nm_Supp_St$_j$ resets $blocking_i$ and consigns the cached transmission to P_i, if any. For a disengaged Nm_Hst, that is a affiliate of *narrowest_set*[], the Nm_Supp_St that has its disengaged resurgence-point, transfigures its disengaged resurgence-point into quasi-imperishable one and streamlines its CIs.

(c) Reckoning Transmission Incurred During VRL-agglomeration

During impeding interlude, P_i treats m, incurred from P_j, if succeeding conditions are met: (i) (!bufer$_i$) i.e. P_i has not cached any transmission (ii) ($m.cci$!= nci_i) i.e. P_j has not amassed its quasi-imperishable resurgence-point before consigning m (iii) ($SCIA_i$[j]=1) $\vee$ ($matd$[j, $m.cci$]= 0)) i.e. P_i is already reliant upon P_j in the newfangled CI or P_j has amassed some imperishable resurgence-point after consigning m.

Otherwise, the native Nm_Supp_St of P_i annexes m for the impeding interlude of P_i and sets $buffer_i$. On acquiring transmissions, *SCIA* vectors are streamlined as described in Section 2.3(vi).

(d) Termination

When an Nm_Supp_St learns that all of its implementations in narrowest set have amassed their quasi-imperishable resurgence-points or at least one of its implementation has failed to resurgence-point, it consigns the response transmission to the begetter Nm_Supp_St .

Finally, begetter Nm_Supp_St consigns commit or abort to all implementations. On acquiring abort, an implementation discards its quasi-imperishable resurgence-point, if any, and undoes the streamlining of data configurations. On acquiring commit, implementations, in the *narrowest_set*[], transfigure their quasi-imperishable resurgence-points into imperishable ones. On acquiring commit or abort, all implementations update their *SCIA* vectors and other data configurations.

4. All-implementation VRL-agglomeration

Our all implementation VRL-agglomeration blueprint is similar to Elnozahy et al [8]. Pioneer Nm_Supp_St consigns requisition to all implementations to resurgence-point. On acquiring the resurgence-point requisition, an implementation amasses the quasi-imperishable resurgence-point if it has not amassed the resurgence-point during newfangled inception. After amassing a resurgence-point, an implementation streamlines its CIs. An implementation, after amassing its quasi-imperishable resurgence-point or knowing its inability to amass the resurgence-point, informs its native Nm_Supp_St .

When an implementation consigns a reckoning transmission, it appends its *cci* with the transmission. When an implementation, say P_i, acquires a reckoning transmission *m* from some other implementation, say P_j, P_i amasses the quasi-imperishable resurgence-point before administering the transmission if *m.cci* equals nci_i; otherwise, it simply implementations the transmission.

When an Nm_Supp_St learns that its all implementations have amassed the quasi-imperishable resurgence-points successfully or at least one of its implementations has failed to resurgence-point, it consigns the response to the begetter Nm_Supp_St . Finally, begetter Nm_Supp_St consigns commit or abort to all Nm_Suppt_Sts.

On commit, all implementations transfigure their quasi-imperishable resurgence-points into imperishable ones and update their data configurations. For Nm_Hsts, Nm_Suppt_Sts update the data configurations. On abort, all implementations discard their quasi-imperishable resurgence-points, if any, and undo the streamlining of data configurations.

5. Example

We explain our narrowest-implementation VRL-agglomeration blueprint with the help of an example. In *Fig. 1*, at time t_1, P_1 inductees VRL-agglomeration implementation and consigns requisition to all implementations for their *SCIA* vectors. During the impeding time of an implementation, selective transmissions are cached as follows. P_2 treats m_0, because, P_1 has amassed imperishable resurgence-point after consigning m_0. P_2 treats m_6, because, $SCIA_2[3]$ is already 1 due to acquire of m_3. P_2 annexes m_7, because, $SCIA_2$ [4] is 0 due to non-receipt of any transmission from P_4 during newfangled CI. P_2 annexes m_8 to keep the proper sequence of transmissions incurred. $SCIA_4[5]$ equals 1 due to m_4, therefore, P_4 dispenses m_9. Correspondingly, P_5 dispenses m_{10}, because, $SCIA_5[4]$ equals 1 due to m_5. P_5 annexes m_{13}, because, P_3 has amassed a new resurgence-point before consigning m_{13} and P_5 has not incurred the resurgence-point requisition from P_1.

At time t_2, P_1 acquires the *SCIA[]* from all implementations [not shown in the figure], computes *narrowest_set[]* [which in case of Fig. 1 is $\{P_1, P_2, P_3\}$], sets $cci_1=nci_1$, consigns resurgence-point requisition along with the *narrowest_set[]* to all implementations, and amasses its own quasi-imperishable resurgence-point.

When P_2 acquires the resurgence-point requisition, it finds itself a affiliate of the *narrowest_set []*.It amasses the succeeding actions: (i) amass its own quasi-imperishable resurgence-

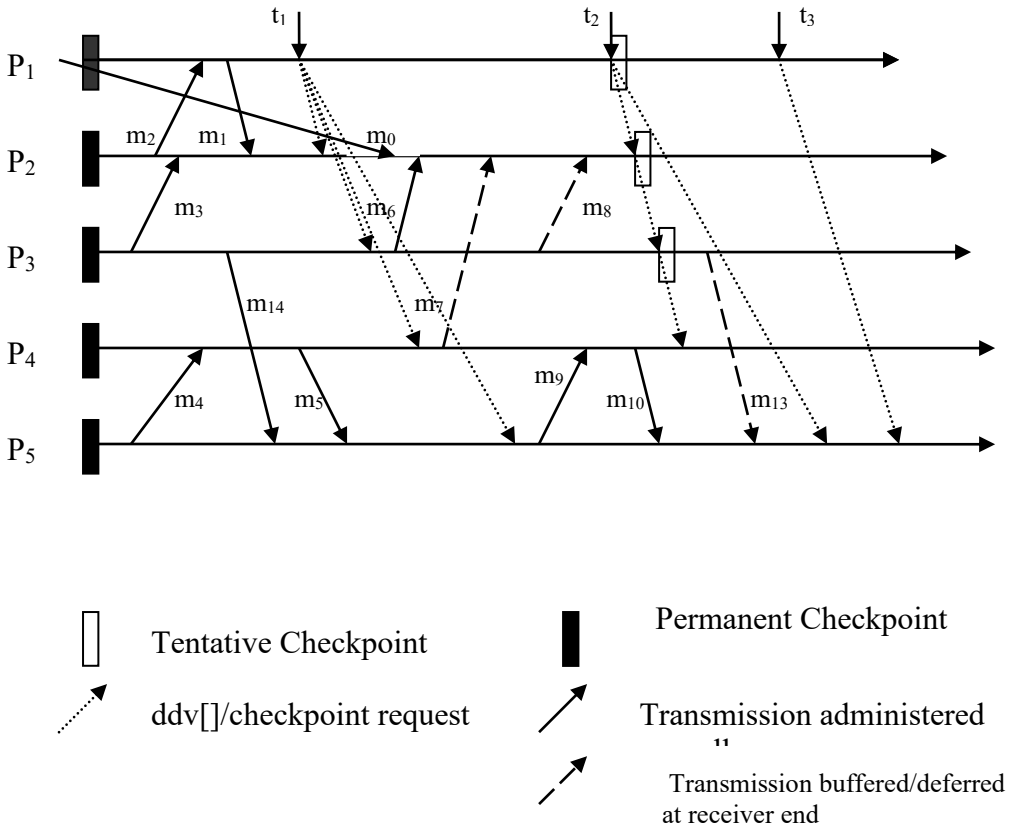

Figure 1 Illustration of narrowest-implementation VRL-agglomeration Tactic

point, (ii) set $cci_2=nci_2$, (iii) consign the response to P_1 [not shown in the figure], (iv) implementation the cached transmissions, i.e., m_7 and m_8. When P_5 acquires the resurgence-point requisition, it is not a affiliate of the *narrowest_set[]*; therefore, it does not resurgence-point but implementations the cached transmission, i.e., m_{13}. At time t_3, P_1 acquires responses, decides to commit or abort the VRL-agglomeration activity, and consigns abort or commit requisition to all implementations.

Evaluation of the planned outcrossed blueprint

In the planned outcrossed blueprint, the all-implementation blueprint planned by Elnozahy et al [9] is enforced after accomplishing planned min-implementation blueprint for m times. Therefore, the performance of the outcrossed blueprint is mainly reliant upon these two blueprints and the value of m.

As shown in Table 3, the average impeding time of the Koo-Toueg [15] blueprint is the highest, followed by Cao-Singhal [4] blueprint. The average impeding time of the planned outcrossed scheme is slightly less than [4]. The other schemes are non-impeding, [5, 9, 16, 19]. In Elnozahy et al [9] blueprint, all implementations amass resurgence-points. In the

blueprints [4, 15], only narrowest numbers of implementations record their resurgence-points. In non-stalling narrowest-implementation VRL-agglomeration scheme, [5, 16, 19], some inoperable resurgence-points may be amassed, which are thrown away on commit. The aggregate of inoperable resurgence-points in [19] is negligibly small as paralleled to [5]. In the narrowest-implementation blueprints, some implementations may starve to resurgence-point and the abatement of reckoning in the case of a reclamation after a culpability may be outstandingly elevated. In the planned blueprint, the average aggregate of implementations that amass resurgence-points in an inception is slightly greater than the narrowest requisitioned; but it condenses the abatement of reckoning on reclamation.

The average transmission outlay in the planned blueprint is slightly less than [4] and [19], but greater than [9] [Refer Table 3]. In orchestrated VRL-agglomeration, an integer snp_seq_no is generally annexed on normal transmissions [5, 9, 16, 19]. In the blueprint [4], no information is annexed on normal transmissions. In the planned blueprint, k -bit CI is annexed on normal transmissions. In the present study, we have amassed k=3. Coinciding accomplishments of the blueprint are endorsed in [5]. W. Ni et al [23] have shown that this blueprint [5] may lead to inconsistencies during coinciding accomplishments .

7. Conclusion

We have designed an orchestrated VRL-agglomeration blueprint which is an outcrossed of narrowest-implementation and all-implementation blueprints. The aggregate of implementations that amass resurgence-points is minimized to circumvent arousing of Nm_Hsts in doze mode of operation and thrashing of Nm_Hsts with VRL-agglomeration activity. Further, it protects restricted battery life of Nm_Hsts and low bandwidth of cellular pathways. Moreover, to circumvent greater abatement of reckoning in case of a reclamation after a culpability, an all-implementation resurgence-point is amassed after accomplishing narrowest-implementation VRL-agglomeration for a fixed aggregate of times, which, in fact, can be fine-tuned. VRL-agglomeration outlay in the planned scheme is slightly greater than the narrowest-implementation VRL-agglomeration but is far less than the all-implementation orchestrated VRL-agglomeration. We have introduced the k-bit sequence numbers instead of ever growing integer snp_seq_no that is annexed on normal transmissions. This also leads to reduction in the interaction outlay. We have also condensed the impeding of implementations during VRL-agglomeration. We also manage to shorten the mislaying of VRL-agglomeration striving when any implementation misses to collate its proximate-resurgence-point in consonance with others.

Reference

[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] N. Chand , R.C. Joshi , Manoj Misra, Cooperative caching in mobile ad hoc networks based on data utility, *Mobile Information Systems*, 3(1), 2007, 19-37.

[8] 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.

[9] 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.

[10] Christoph Endres, Andreas Butz, Asa MacWilliams, A survey of software infrastructures and frameworks for ubiquitous computing", *Mobile Information Systems*, 1(1), 2005, 41-80.

[11] Anders Fongen, Christian Larsen, Gheorghita Ghinea, Simon J.E. Taylor and Tacha Serif, Location based mobile computing - A tuplespace perspective, *Mobile Information Systems*, 2(2-3), 2006, 135 – 149.

[12] 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.

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

[14] James Jayaputera and David Taniar, Data retrieval for location-dependent queries in a multi-cell wireless environment, *Mobile Information Systems*, 1(2), 2005, 91-108.

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

[16] 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.

[17] 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.

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

[19] 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.

[20] 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.

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

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

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

[24] David C.C. Ong , Rytis Sileika , Souheil Khaddaj , Radouane Oudrhiri, Alternative data storage solution for mobile messaging services, *Mobile Information Systems*, 3(1), 2007, 49-54.

[25] 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.

[26] 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.

[27] Ioannis Priggouris, Dimitrios Spanoudakis, Manos Spanoudakis, Stathes Hadjiefthymiades, A generic framework for Location-Based Services (LBS) provisioning, *Mobile Information Systems*, 2(2-3), 2006, 111-133.

[28] 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.

[29] 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.

30. 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.

31. 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