Journal of Data Acquisition and Processing
ISSN: 1004-9037
https://sjcjycl.cn/
DOI: 10.5281/zenodo.777645

INSTANTANEOUS SNAPSHOT-BASED MINIMUM-PROCESS ACCORDANT RECOVERY LINE AGGLOMERATION BLUEPRINT FOR FAULT-RESILIENT MOBILE DISTRIBUTED SYSTEMS

Ashwini Patil R K
Research Scholar, Department of Computer Science & Engineering, Sunrise University, Alwar, Rajasthan, India, Ashwinrk1115@gmail.com

Dr. Rajeev Yadav
Professor, Department of Computer Science & Engineering, Sunrise University, Alwar, Rajasthan, India, Ashwinrk1115@gmail.com

Abstract:
We propound a rock-bottom transaction Accordant Recovery Line agglomeration (ARL-agglomeration) blueprint for Nomadic systems, where no inoperative snapshots (recapture-pinpoints) are hoarded and an effort has been made to moderate the intrusion of transactions. We propose to delay the processing of selective missives at the receiver end only during the ARL-agglomeration period. A transaction is indorsed to carry out its normal computations and ship missives during its intrusion period. In this way, we try to keep intrusion of transactions to bare rock-bottom. In order to keep the intrusion time rock-bottom, we grab the causal-interrelationships vectors and compute the accurate rock-bottom set in the beginning of the blueprint. In orchestrated ARL-agglomeration, if a solitary transaction flops to retain its recapture-pinpoint; all the ARL-agglomeration effort goes waste, because, each transaction has to terminate its partially-perpetual recapture-pinpoint. In order to retain its partially-perpetual recapture-pinpoint, a Nomadic Node needs to transfer large recapture-pinpoint data to its native Nm_Spp_St (Nomadic Support Station) over wireless channels. The ARL-agglomeration effort may be remarkably high due to recurrent forsakes especially in nomadic systems. We try to curtail the loss of ARL-agglomeration effort when any transaction flops to retain its recapture-pinpoint in coordination with others. In the first phase, we retain instantaneous recapture-pinpoints only. In this case, if any transaction flops to retain its recapture-pinpoint in the first phase, all concerned transactions need to terminate their instantaneous recapture-pinpoints only and not the partially-perpetual ones.
Keywords: Culpability Immunity, Nomadic Computing Systems, orchestrated checkpointing, Rollback Recovery, Distributed Systems.

I. INTRODUCTION

A distributed system (DS) is a coalition of self-regulating entities that collaborate to elucidate a problem that cannot be individualistically resolved. A Nomadic system is a DS, where some of transactions are executing on nomadic hosts, whose position in the network changes with time. The number of transactions that retain recapture-pinpoints in a particular instigation is curtailed to 1) evade awakening of Nm_Nds in doze manner of transaction, 2) curtail thrashing of Nm_Nds with ARL-agglomeration activity, 3) retain inadequate battery life of Nm_Nds and stumpy bandwidth of wireless channels. In least transaction (rock-bottom interacting

transaction) ARL-agglomeration etiquettes, some inoperative recapture-pinpoints are recorded or impeding of transactions takes place. In this paper, we propose a least transaction orchestrated ARL-agglomeration etiquette for non-deterministic Nomadic system, where no unworkable recapture-pinpoints are hoarded. Prakash Singhal [12] endorsed that a good ARL-agglomeration blueprint for nomadic systems should have low reminiscence overheads on Nm_Nds, low overheads on wireless channels and should avoid awakening of a Nm_Nd in doze manner. The Disjointedness of Nm_Nds should not lead to infinite wait state. The blueprint should be non-intrusive, orchestrated, and should force rock-bottom number of transactions to retain their native recapture-pinpoints.

Rock-bottom-transaction orchestrated ARL-agglomeration is an attractive methodology to introduce culpability immunity in nomadic systems transparently. It avoids domino-effect, minimizes stable storage requirements, and forces only rock-bottom interacting transactions to retain recapture-pinpoint. To recuperate from a disappointment, the system simply restarts its execution from a previous consistent all-inclusive recapture-pinpoint hoarded on the stable storage. But, it has the following disadvantages. Some intrusion of transactions takes place or some inoperative recapture-pinpoints are hoarded. In order to record a consistent all-inclusive recapture-pinpoint, transactions must synchronize their ARL-agglomeration activities. In other words, when a transaction pledges ARL-agglomeration tactic, it asks all applicable transactions to retain their recapture-pinpoints. Therefore, orchestrated ARL-agglomeration suffers from high overhead associated with the ARL-agglomeration coordination. Sometimes, recapture-pinpoint order numbers are piggybacked along with computation missives. If a solitary transaction flops to retain recapture-pinpoint, the whole ARL-agglomeration effort of the commencement goes waste [13, 14, 15, 16, 17].

While handling nomadic systems, we come across some issues like: Suppleness, low bandwidth of wireless channels and dearth of stable storage on nomadic nodes, disconnections, inadequate battery power and high disappointment rate of nomadic nodes. These issues make traditional ARL-agglomeration techniques planned for Distributed systems unbefitting for Nomadic environments. In this paper, we propound a rock-bottom transaction blueprint for Nomadic systems, where no inoperative recapture-pinpoints are hoarded and an effort has been made to moderate the intrusion of transactions. We propose to delay the processing of selective missives at the receiver end only during the ARL-agglomeration period. A transaction is allowed to carry out its normal computations and ship missives during its intrusion period. In this way, we try to keep intrusion of transactions to bare rock-bottom. In order to keep the intrusion time rock-bottom, we grab the causal-interrelationships vectors and compute the accurate rock-bottom set in the beginning of the blueprint. In orchestrated ARL-agglomeration, if a solitary transaction flops to retain its recapture-pinpoint; all the ARL-agglomeration effort goes waste, because, each transaction must terminate its partially-perpetual recapture-pinpoint. In order to retain its partially-perpetual recapture-pinpoint, a Nm_Nd needs to transfer large recapture-pinpoint data to its native Nm_Spp_St over wireless channels. The ARL-agglomeration effort may be remarkably high due to recurrent forsakes especially in nomadic systems. We try to curtail the loss of ARL-agglomeration effort when any transaction flops to retain its recapture-pinpoint in coordination with others. In the first

phase, we retain instantaneous recapture-pinpoints only. In this case, if any transaction flops to retain its recapture-pinpoint in the first phase, all concerned transactions need to terminate their instantaneous recapture-pinpoints only and not the partially-perpetual ones as in [9, 10].

II. PROPOSED RESEARCH IDEA

The endorsed propound is contingent on keeping track of direct dependencies of transactions. Like [10], motivator transaction grabs the direct causal-interrelationships vectors of all transactions, computes rock-bottom set, and ships the recapture-pinpoint requisition along with the rock-bottom set to all transactions. In this way, intrusion time has been expressively reduced as compared to Koo_Toueg blueprint [2].

During the period, when a transaction ships its causal-interrelationships set to the motivator and obtains the rock-bottom set, may receive some missives, which may add new members to the already computed rock-bottom set. We define this period as the uncertainty period or the intrusion period of a transaction. This period is negligibly trivial. Hence the intrusion time of a transaction in the endorsed propound is quite low. In order to keep the computed rock-bottom set intact, we have classified the missives at a transaction, received during its uncertainty period, into two types: (i) missives that alter the causal-interrelationships set of the receiver transaction (ii) missives that do not alter the causal-interrelationships set of the receiver transaction. The missives in point (i) need to be delayed at the receiver side. The missives in point (ii) can be treated normally. All transactions can carry out their normal computations and ship missives during their intrusion period. When a transaction buffers an application-message of former type, it does not transaction any application-message till it obtains the rock-bottom set so as to keep the proper order of missives received. When a transaction gets the rock-bottom set, it saves the recapture-pinpoint, if it is in the rock-bottom set. After this, it obtains the buffered missives, if any. A transaction, not in the rock-bottom set, comes out of the intrusion state immediately after getting the rock-bottom set. The endorsed rock-bottom-transaction intrusion blueprint forces zero inoperative recapture-pinpoints at the cost of very trivial intrusion.

In rock-bottom-transaction orchestrated ARL-agglomeration, the motivator transaction asks all communicating transactions to retain partially-perpetual recapture-pinpoints. In this propound, if a solitary transaction flops to retain its recapture-pinpoint; all the ARL-agglomeration effort goes waste, because, each transaction has to terminate its partially-perpetual recapture-pinpoint. In order to retain the partially-perpetual recapture-pinpoint, a Nm_Nd needs to transfer large recapture-pinpoint data to its native Nom_Suppt_St over wireless channels. Due to recurrent forsakes, total ARL-agglomeration effort may be remarkably high, which may be undesirable in nomadic systems due to scarce resources. Recurrent forsakes may happen in nomadic systems due to fatigued battery, abrupt Disjointedness, or bad wireless connectivity. Therefore, we propose that in the first phase, all concerned Nm_Nds will retain instantaneous recapture-pinpoint only. Instantaneous recapture-pinpoint is stored on the reminiscence of Nm_Nd only. In this case, if some transaction flops to retain recapture-pinpoint in the first phase, then Nm_Nds need to terminate their instantaneous recapture-pinpoints only. The effort of arresting an instantaneous recapture-pinpoint is insignificant as compared to the partially-

perpetual one. Hence, in case of a disappointment during ARL-agglomeration, the loss of ARL-agglomeration effort is expressively reduced. When the motivator discovers that all applicable transactions have hoarded their instantaneous recapture-pinpoints, it asks all applicable transactions to come into the second phase, in which, a transaction transforms its instantaneous recapture-pinpoint into partially-perpetual one. In this way, by increasing trivial orchestration application-message overhead, we are able to handle recurrent forsakes during ARL-agglomeration due to disappointment of some node or application-message channel and, in turn, try to reduce the total ARL-agglomeration effort.

III. AN EXAMPLE OF THE ENDORSED ETIQUETTE

We explain the endorsed rock-bottom-transaction ARL-agglomeration blueprint with the help of an example. In Figure 1, at time t_1, P_4 pledges ARL-agglomeration transaction and ships requisition to all transactions for their causal-interrelationships vectors. At time t_2, P_4 obtains the causal-interrelationships vectors from all transactions (not shown in the Figure 1) and computes the rock-bottom set (*rock_bott_vtr[]*) which is {P_3, P_4, P_5}.

P_4 ships *rock_bott_vtr[]*to all transactions and saves its own instantaneous recapture-pinpoint. A transaction saves its instantaneous recapture-pinpoint if it is a member of *rock_bott_vtr[]*. When P_3 and P_5 get the *rock_bott_vtr[]*, they find themselves in the *rock_bott_vtr*[]; therefore, they retain their instantaneous recapture-pinpoints. When P_0, P_1 and P_2 get the *rock_bott_vtr* [], they find that they do not belong to *rock_bott_vtr []*, therefore, they do not retain their instantaneous recapture-pinpoints.

A transaction comes into the intrusion state immediately after shipping the causal-interrelationships vector to the motivator. A transaction comes out of the intrusion state only after arresting its instantaneous recapture-pinpoint if it is a member of the rock-bottom set; otherwise, it comes out of intrusion state immediately after getting the instantaneous recapture-pinpoint requisition. P_4 obtains m_4 during its intrusion period. *_st[]* is a direct causal-interrelationships vector maintained at every transaction. As *ci_vctr_st*$_4$[5] =1 due to m_3, and receive of m_4 will not alter *ci_vctr_st*$_4$[]; therefore, P_4 transactions m_4. P_1 obtains m_5 from P_2 during its intrusion period; *ci_vctr_st*$_1$[2]=0 and the receiver of m_5 can alter *ci_vctr_st*$_1$[]; therefore, P_1 buffers m_5. Correspondingly, P_3 buffers m_6. P_3 computes m6 only after arresting its instantaneous recapture-pinpoint. P_1 transaction m_5 after getting the *rock_bott_vtr []*. P_2 transactions m7 because at this moment it not in the intrusion state. Correspondingly, P_3 computes m_8. At time t_3, P_4 obtains rejoinders to instantaneous checkpoint appeals from all applicable transactions (not shown in the Figure 1) and ships partially-perpetual recapture-pinpoint requisition to all concerned transactions. A transaction in the rock-bottom set transforms its instantaneous recapture-pinpoint into partially-perpetual one. In conclusion, at time t_4, P_4 obtains rejoinders to partially-perpetual recapture-pinpoint appeals from all applicable transactions (not shown in the Figure 1) and ships the commit requisition . In this case, P_3, P_4 and P_5 advance their recovery line by arresting new recapture-pinpoints in the new commencement of the ARL-agglomeration blueprint , whereas, P_0, P_1 and P_2 do not advance their recovery line. In this case if some disappointment occurs, P_0, P_1 and P_2 will roll back to their initial state and P_3, P_4 and P_5 will roll back to their perpetual state.

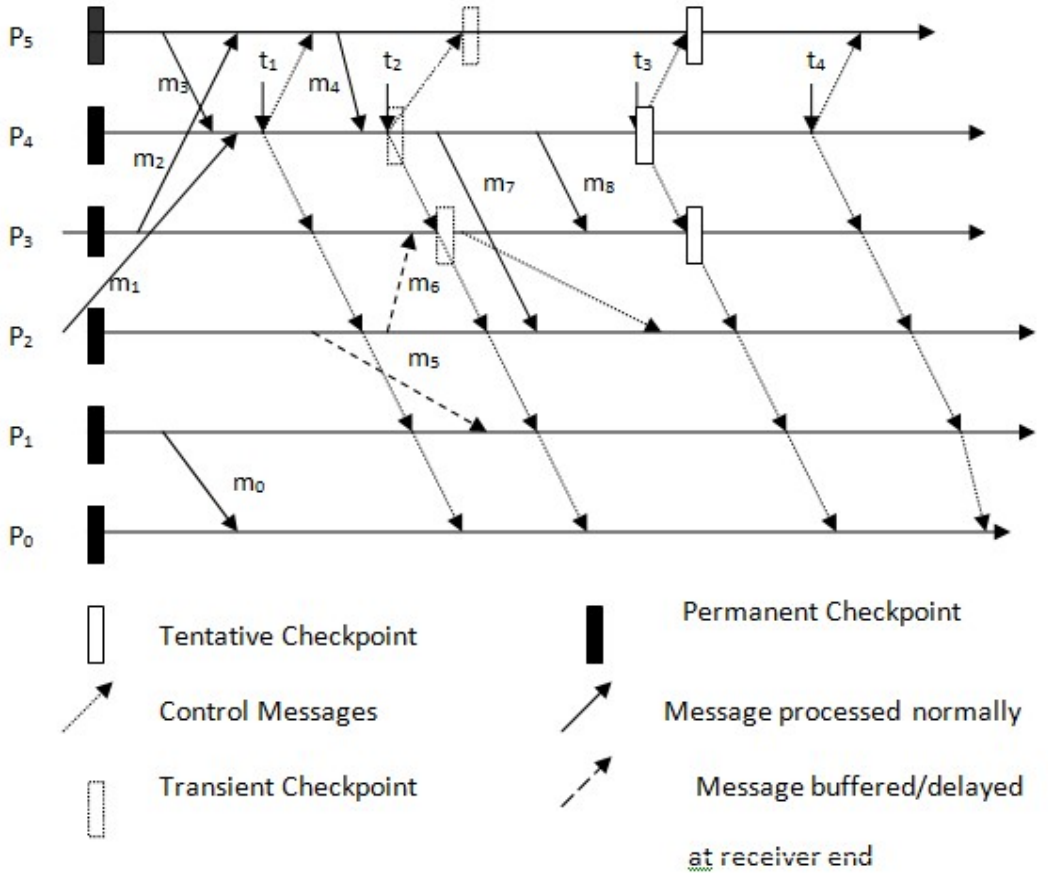

Fig 1: An Example of the endorsed Blueprint

IV. THE ENDORSED ETIQUETTE

When a Nm_Nd ships an application application-message, it is first sent to its native Nm_Spp_St over the wireless cell. The Nm_Spp_St piggybacks apposite information with the application application-message, and then routes it to the destination Nm_Spp_St or Nm_Nd. When the Nm_Spp_St obtains an application application-message to be forwarded to a native Nm_Nd, it first updates the data structures that it preserves for the Nm_Nd, strips all the piggybacked information, and then forwards the application-message to the Nm_Nd. Thus, a Nm_Nd ships and obtains application missives that do not contain any additional information; it is only responsible for recording its native state appropriately and relocating it to the native Nm_Spp_St.

The motivator Nm_Spp_St ships a requisition to all Nm_Spp_Sts to ship the *ci_vctr_st* vectors of the transactions in their cells. All *ci_vctr_st* vectors are at Nm_Spp_Sts and thus no initial ARL-agglomeration missives or rejoinders travels wireless channels. On receiving the *ci_vctr_st* [] requisition, an Nm_Spp_St records the identity of the motivator transaction and motivator Nm_Spp_St (say mss_id_a), ships back the *ci_vctr_st* [] of the transactions in its cell, and sets *g_snpsht*. If the motivator Nm_Spp_St obtains a requisition for *ci_vctr_st* [] from some other Nm_Spp_St (say mss_id_b) and mss_id_a is lower than mss_id_b, the, current commencement with mss_id_a is discarded and the new one having mss_id_b is continued. Correspondingly, if an Nm_Spp_St obtains *ci_vctr_st* appeals from two

Nm_Spp_Sts, then it throw-outs the requisition of the motivator Nm_Spp_St with lower mss_id. If an Nm_Spp_St obtains a new recapture-pinpoint commencement requisition from some transaction in its cell and the flag *g_snpsht* is already set, then the Nm_Spp_St will throw-out this new commencement to avoid concurrent execution of the ARL-agglomeration blueprint. Otherwise, on receiving *ci_vctr_st* vectors of all transactions, the motivator Nm_Spp_St computes *rock_bott_vtr* [], ships instantaneous recapture-pinpoint requisition along with the *rock_bott_vtr* [] to all Nm_Spp_Sts. When a transaction ships its *ci_vctr_st* [] to the motivator Nm_Spp_St, it comes into its intrusion state. A transaction comes out of the intrusion state only after arresting its instantaneous recapture-pinpoint if it is a member of the rock-bottom set; otherwise, it comes out of intrusion state after getting the instantaneous recapture-pinpoint requisition.

On receiving the instantaneous recapture-pinpoint requisition along with the *rock_bott_vtr* [], an Nm_Spp_St, say Nm_Spp_St_j, saves the following actions. It ships the instantaneous recapture-pinpoint requisition to P_i only if P_i affiliates to the *rock_bott_vtr* [] and P_i is running in its cell. On receiving the recapture-pinpoint requisition, P_i saves its instantaneous recapture-pinpoint and informs Nm_Spp_St_j. On receiving positive rejoinder from P_i, Nm_Spp_St_j updates $p\text{-}csn_i$, resets $intrusion_i$, and ships the buffered missives to P_i, if any. Alternatively, If P_i is not in the *rock_bott_vtr* [] and P_i is in the cell of Nm_Spp_St_j, Nm_Spp_St_j resets $intrusion_i$ and ships the buffered application-message to P_i, if any. For a disconnected Nm_Nd, that is a member of *rock_bott_vtr* [], the Nm_Spp_St that has its disconnected recapture-pinpoint, transforms its disconnected recapture-pinpoint into the required one.

During intrusion period, P_i computes m, received from P_j, if all of the following conditions are met:

(i) ($!bufer_i$) i.e. P_i has not buffered any application-message

(ii) (*m.p_csn =csn[j]*) i.e. P_j has not hoarded its recapture-pinpoint before shipping *m and* (*ci_vctr_st*$_i$[j] =1) P_i is already reliant on Pj in the current CI

or

m.p_csn <csn[j]. P_j has hoarded some permanent recapture-pinpoint after shipping *m*.

Otherwise, if any of these three conditions is not met, the native Nm_Spp_St of P_i buffers *m* for the intrusion period of P_i and sets $buffer_i$.

When a Nm_Spp_St concludes that all its transactions in rock-bottom set have hoarded their instantaneous recapture-pinpoints or at least one of its transactions has nosedived to retain recapture-pinpoint, it ships the rejoinder application-message to the motivator Nm_Spp_St. In this case, if some transaction flops to retain instantaneous recapture-pinpoint in the first phase, then concerned Nm_Nds need to terminate their instantaneous recapture-pinpoints only. The effort of arresting an instantaneous recapture-pinpoint is insignificant and less than 1% as compared to the partially-perpetual one [6]. In this way, the loss of ARL-agglomeration effort, in case of an terminate of the ARL-agglomeration tactic, is expressively low. We want to further emphasize that the recurrent forsakes is an inevitable feature in orchestrated ARL-agglomeration in nomadic systems due to fatigued battery, abrupt Disjointedness, or bad wireless connectivity. When the motivator Nm_Spp_St discovers that all applicable transactions have hoarded their instantaneous recapture-pinpoints, it asks all

applicable transactions to come into the second phase, in which, a transaction transforms its instantaneous recapture-pinpoint into partially-perpetual one.

In conclusion, motivator Nm_Spp_St ships commit or terminate to all transactions. On receiving terminate, transactions throw-out its partially-perpetual recapture-pinpoint, if any, and undo the updating of data structures. On receiving commit, transactions, in the *rock_bott_vtr* [], convert their partially-perpetual recapture-pinpoints into permanent ones. On receiving commit or terminate, all transactions update their *ci_vctr_st* vectors and other data structures.

V. COMPARISON WITH OTHER ALGORITHMS

The Koo-Toueg [2] blueprint is a rock-bottom-transaction orchestrated ARL-agglomeration blueprint for distributed systems. It requires transactions to be blocked during ARL-agglomeration. ARL-agglomeration includes the time to find the rock-bottom interacting transactions and to retain the state of transactions on stable storage, which may be too long. In Cao-Singhal blueprint [9], intrusion time is reduced expressively as compared to [2].

The algorithms endorsed in [6, 12] are non-intrusive, but they suffer from inoperative recapture-pinpoints. It should be distinguished that inoperative recapture-pinpoints are undesirable in nomadic systems due to scarcity of resources. In the endorsed propound, the orchestration application-message is on higher side. We add two extra phases, one to grab the causal-interrelationships vectors and another to retain the instantaneous recapture-pinpoints. First phase is added to compute the accurate rock-bottom set in the beginning of the blueprint to curtail the intrusion time as in [2]. In order to curtail the loss of ARL-agglomeration effort when any transaction flops to retain its recapture-pinpoint in coordination with others, all applicable transactions retain instantaneous recapture-pinpoints in the first phase and convert their instantaneous recapture-pinpoints into partially-perpetual recapture-pinpoints in the second phase. In this way, by adding extra orchestration application-message overhead, we are able to deal with the problem of recurrent forsakes in coordinating ARL-agglomeration. In case of recurrent forsakes, we expressively reduce loss of ARL-agglomeration effort as compared to [2, 5, 6]. Because, in all these blueprints, in case of an terminate of the ARL-agglomeration tactic, all concerned transactions need to terminate their partially-perpetual recapture-pinpoints, whereas, in the endorsed blueprint, all concerned transactions need to terminate their instantaneous recapture-pinpoints. In case of a Nm_Nd, the cost of arresting an instantaneous recapture-pinpoint is insignificant and is less than 1% as compared to the cost of arresting a partially-perpetual recapture-pinpoint. Recurrent forsakes may occur in orchestrated ARL-agglomeration in nomadic systems due to Suppleness, low bandwidth of wireless channels, disconnections and inadequate battery power.

VI. CONCLUSION

We have endorsed a rock-bottom transaction orchestrated ARL-agglomeration blueprint for nomadic system, where no inoperative recapture-pinpoints are hoarded and an effort is made to curtail the intrusion of transactions. We are able to reduce the intrusion time to bare rock-bottom by computing the accurate rock-bottom set in the beginning. Furthermore, the intrusion of transactions is reduced by allowing the transactions to carry out their normal computations

and ship missives during their intrusion period. The number of transactions that retain recapture-pinpoints is diminished to avoid awakening of Nm_Nds in doze manner and thrashing of Nm_Nds with ARL-agglomeration activity. It also saves inadequate battery life of Nm_Nds and low bandwidth of wireless channels. We try to reduce the loss of ARL-agglomeration effort when any transaction flops to retain its recapture-pinpoint in coordination with others.

REFERENCES

[1] Chandy K.M. and Lamport L., "Distributed snapshots : Determining Recovery Line of Distributed Systems, " ACM Transaction on Computing Systems, vol., 3 No. 1, pp 63-75, February, 1985.

[2] Koo R. and Tueg S., "Checkpointing and Rollback recovery for Distributed Systems", IEEE Trans. On Software Engineering, Vol. 13 no. 1, pp 23-31, January 1987.

[3] Elonzahy E.N., Alvisi L., Wang Y.M. and Johnson D.B., "A survey of Rollback-Recovery etiquettes in message-Passing Systems", ACM Computing surveys, vol. 34 no. 3, pp 375-408, 2002.

[4] L. Alvisi," Understanding the message Logging Paradigm for Masking Transaction Crashes," Ph.D. Thesis, Cornell Univ., Dept. of Computer Science, Jan. 1996. Available as Technical Report TR-96-1577.

[5] Lalit Kumar P. Kumar "A synchronous checkpointing etiquette for mobile distributed systems: probabilistic approach" Int Journal of information and computer security 2007.

[6] Cao, M.Singhal, "Mutable Checkpointing : A New DRL-accumulation Approach for Mobile Computing Systems", IEEE Transactions on Parallel and Distributed system, vol.12, Issue 2, Feb., 2001, pages: 157-172, ISSN: 1045-9219.

[7] Acharya A. and Badrinath B. R., "Checkpointing Distributed Applications on Mobile Computers," *Proceedings of the 3rd International Conference on Parallel and Distributed Information Systems*, pp. 73-80, September 1994.

[8] M. Singhal and N. Shivaratri, Advanced Concepts in Operating Systems, New York, McGraw Hill, 1994.

[9] Cao G. and Singhal M., "On coordinated Checkpointing in Distributed Systems", *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no.12, pp. 1213-1225, Dec 1998.

[10] Cao G. and Singhal M., "On the Impossibility of Minimum process Non-intrusion Checkpointing and an Efficient Checkpointing Etiquette for Mobile Computing Systems," *Proceedings of International Conference on Parallel Methoding*, pp. 37-44, August 1998.

[11] Kumar, P.," A Low-Cost Hybrid Coordinated Checkpointing Etiquette for Mobile Distributed Systems", *Mobile Information Systems* pp 13-32, Vol. 4, No. 1. ,2007.

[12] Prakash R. and Singhal M., "Low-Cost Checkpointing and Failure Recovery in Mobile Computing Systems," *IEEE Transaction On Parallel and Distributed Systems,* vol. 7, no. 10, pp. 1035-1048, October1996.

[13] Houssem Mansouri, Nadjib Badache, Makhlouf Aliouat and Al-Sakib Khan Pathan, "A New Efficient Checkpointing Algorithm for Distributed Mobile Computing", Control Engineering and Applied Informatics, Vol. 17, Issue: 2, Page No. 43-54, 2015.

[14] Bakhta Meroufel and Ghalem Belalem, "Enhanced Coordinated Checkpointing in Distributed System", International Journal of Applied Mathematics and Informatics, Vol. 9, Page No. 23-32, 2015.

[15]Houssem Mansouri and Al-Sakib Khan Pathan, "Checkpointing Distributed Computing Systems: An Optimization Approach", International Journal High Performance Computing and Networking, Vol. 15, No. 3/4, Page No. 202-209, 2019.

[16] Praveen Choudhary, Parveen Kumar," Low-Overhead Minimum-Transaction 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

[17] Deepak Chandra Uprety, Parveen Kumar, Arun Kumar Chouhary," Transient Snapshot based Minimum-process Synchronized Checkpointing Etiquette for Mobile Distributed Systems", International Journal of Emerging Trends in Engineering Research", Vol 10, No 4, Aug. 2021.