ISSN 2347 - 3983
Volume 9. No. 8, August 2021
International Journal of Emerging Trends in Engineering Research
Available Online at http://www.warse.org/IJETER/static/pdf/file/ijeter06982021.pdf
https://doi.org/10.30534/ijeter/2021/06982021

# Low-Overhead Minimum-Method Global Snapshot Compilation Protocol for Deterministic Mobile Computing Systems

**Praveen Choudhary[1], Parveen Kumar[2]**

[1]Research Scholar, NIMS University, Jaipur, India, praveenchoudhary0@gmail.com
[2]Professor, Dept. of Computer Sc. & Engineering, NIMS University, Jaipur, India, parveen.kumar@nimsuniversity.org

## ABSTRACT

CGS-accumulation (Consistent Global State Accumulation) is one of the commonly used method to provide fault tolerance in distributed systems so that the system can operate even if one or more components have failed. However, mobile computing systems are constrained by low bandwidth, mobility, lack of stable storage, frequent disconnections and limited battery life. Hence CGS-accumulation etiquettes which have lesser reinstatement-points are favored in mobile environment.

In this paper, we propose a minimum-method coordinated CGS-accumulation etiquette for deterministic distributed applications on mobile computing systems. We eliminate useless reinstatement-points as well as blocking of methods during reinstatement-points at the cost of logging anti-messages of very few messages during CGS-accumulation.

We also try to minimize the loss of CGS-accumulation effort when any method miscarries to capture its reinstatement-point in an instigation. In this way, we take care of excessive disappointments during CGS-accumulation. We make logging of anti-messages of very few messages only during CGS-accumulation. We also strive to minimize loss of CGS-accumulation effort.

**Key words:** Anti-message, CGS-accumulation, Deterministic Systems, Distributed Systems.

## 1. INTRODUCTION

Reinstatement_point is a way to add fault tolerance into computing system. It essentially consists of saving a snapshot of the application state, in order that it is able to restart from that point in case of failure. A reinstatement_point is a local state of a method saved on stable storage. By periodically invoking the CGS-accumulation procedure, one can save the status of a program at regular intervals [1], [2]. If there is a disappointment, one may restart computation from the last reinstatement-points, thereby, avoiding reiterating computation from the commencement. The method of recommencing computation by rolling back to a saved state is called rollback recovery [3]. In a distributed system, since the methods in the system do not share memory, a global state of the system is defined as a set of local states, one from each method. The state of channels corresponding to a global state is the set of messages sent but not yet received [4].

In deterministic Mobile_DS(Mobile Distributed Systems), if two procedures start in the identical state, and both receive the undistinguishable sequence of inputs, they will yield the duplicate sequence of outputs and will commit in the similar state. The state of a procedure is thus entirely determined by its opening state, received communications, and by order of communications it has method [5], [6]. Johnson and Zwaenepoel[5] projected sender-based communication-logging for deterministic systems, where each communication is registered in volatile memory on the machine from which the communication is directed. The communication-log is then autonomously written to the stable storage, without suspending the computation, as part of the sender's periodic resident-reinstatement-point. Johnson and Zwaenepoel [6] used optimistic communication-logging and CGS-accumulation to determine the most recent recoverable state, where every received communication is registered. David R. Jefferson [7] presented the concept of anti-message. Anti-message is precisely like an original communication in format and content except in one field, i.e., sign. Two communications that are indistinguishable except for opposite signs are called anti-messages of one another. All communications directed unambiguously by user programs have a positive (+) sign; and their anti-messages possessed a negative sign (-). Whenever a communication and its anti-message takes place in the same queue, they instantaneously cancel one another. Thus, the result of adding a communication to a queue may be to abbreviate the queue by one communication rather than lengthening it by one. We portray the anti-message of m by $m^{-1}$.

In this paper, we suggest a min-method synchronized CGS-accumulation scheme for deterministic Mobile_DSs. We eradicate inoperable reinstatement_points as well as hindering of procedures during CGS-accumulation at the cost of logging anti-messages of very few communications during CGS-accumulation.

## 2. SYSTEM MODEL FOR PROPOSED ETIQUETTE

In this paper we used the system model presented in [8]. In this model, a mobile computing system consists of n mobile hosts (Mob_Nodes), and m mobile support stations (Mob-Supp-Sts), where n > m. A cell is a logical or geographical

coverage area under a Mob-Supp-St. A Mob_Node can directly communicate with a Mob-Supp-St $M_i$ only if it is present in the cell serviced by $M_i$. At any time, a Mob_Node belongs to only one cell or may be disengaged. The static network provides reliable First-In-First-Out (FIFO) delivery of messages between any two Mob-Supp-Sts with arbitrary message latency. Similarly, the wireless network within a cell ensures reliable FIFO delivery of messages between a Mob-Supp-St and a Mob_Node.

In our paper, we consider a distributed computation in a mobile computing system that consists of N methods, running contemporaneously on different Mob_Nodes or Mob-Supp-Sts. For simplicity, we assume that each Mob_Node runs one method. Message passing is the only way of communication. The computation is asynchronous. The methods do not share memory or clock. Each method progresses at its own speed and messages are exchanged through reliable channels, whose transmission delays are finite but arbitrary. A method in the cell of Mob-Supp-St means the method is either running on the Mob-Supp-St or on a Mob_Node supported by it. It also includes the methods of Mob_Nodes, which have been disengaged from the Mob-Supp-St but their reinstatement-point related information is still with this Mob-Supp-St.

We also assume that the methods are deterministic. The $i^{th}$ CI (CGS-accumulation interval) of a method denotes all the computation performed between its $i^{th}$ and $(i+1)^{th}$ reinstatement_point, including the $i^{th}$ reinstatement_point but not the $(i+1)^{th}$ reinstatement_point.

## 3. BRIEF DESCRIPTION OF THE PROPOSED ETIQUETTE ALONG WITH ANEXAMPLE

Our CGS-accumulation etiquette method is represented with the help of figure 1. In this Figure, at time $t_1$, $P_C$ initiates CGS-accumulation procedure. $cv_C[B]=1$ due to $m_1$; and $cv_B[E]=1$ due to $m_2$. On the receipt of $m_0$,$P_C$does not set $cv_C[D] =1$, because, $P_D$ has captured its committed resident-reinstatement-point after sending $m_0$. We assume that $P_B$ and $P_C$are in the cell of the same Mob-Supp-St, say Mob-Supp-$St_{in}$. Mob-Supp-$St_{in}$ computes int_vect (subset of least-interacting-set) on the basis of cv vectors maintained at Mob-Supp-$St_{in}$, which in case of Figure 1 is $\{P_B, P_C, P_D\}$. Therefore, $P_C$sends interim resident-reinstatement-pointappeal to $P_B$ and $P_E$ and captures its own interim resident-reinstatement-point. After taking its interim resident-reinstatement-point, $P_B$sends $m_4$ to $P_E$. $P_E$logs $m_4^{-1}$. In this case, $P_B$ has captured its resident-reinstatement-point before sending $m_4$, at the time of receiving $m_4$, $P_E$ has not captured its resident-reinstatement-point for the current instigation. If $P_E$ captures resident-reinstatement-point after receiving $m_4$, the $m_4$ will become orphan. Therefore, $P_E$logs $m_4^{-1}$.On reclamation, $P_E$ will receive $m_4$ as duplicate communication because the procedures are deterministic and $m_4$ will be annihilated by $m_4^{-1}$. Hence receive of $m_4$ as duplicate communication will not cause any inconsistency. It should be noted that this scheme is not captured for non-deterministic systems. After taking its interim resident-reinstatement-point $C_{E1}$, $P_E$ also finds that it was dependent upon $P_F$ before taking the resident-reinstatement-point due to $m_6$ and $P_F$is not in the least-interacting-set computed so far. Therefore, $P_E$ sends interim resident-reinstatement-pointappeal to $P_F$. On receiving the resident-reinstatement-pointappeal, $P_F$ captures its interim resident-reinstatement-point. At time $t_2$, $P_C$ obtains responses from all relevant procedures and sends the partially-committed resident-reinstatement-pointappeal along with the least-interacting-set $[\{P_B, P_C, P_E, P_F\}]$ to all procedures. When a procedure, in the least-interacting-set, obtains the partially-committed resident-reinstatement-pointappeal, it converts its interim resident-reinstatement-point into partially-committed one. Finally, at time $t_3$, $P_C$ sends the commit communication to all concerned procedures. In this example, $\{C_{A0}, C_{B1}, C_{C1}, C_{D0}, C_{E1}, C_{F1}, m_4^{-1}\}$ constitute a reclamation line. It should be noted that, in the recorded global state, $m_4$ is an orphan communication and its anti-message is also recorded at the receiver end.

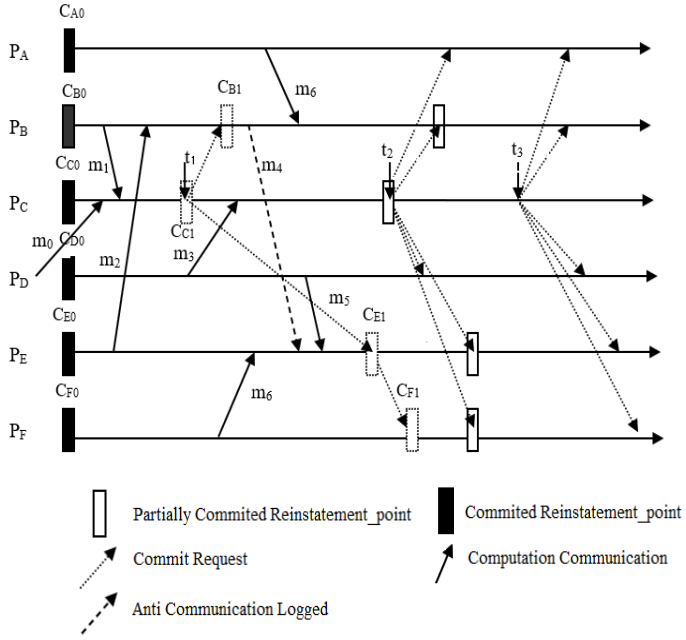


Figure 1: Low Overhead Minimum Method Global Snapshot

## 4. THE PROCESS OF PROPOSED CGS-ACCUMULATION ALGORITHM

In our proposed algorithm when a Mob_Node sends an application communication, it needs to first send to its local Mob-Supp-St over the wireless cell. The Mob-Supp-St can piggyback appropriate information onto the application communication, and then route it to the appropriate destination. When the Mob-Supp-St obtains an application communication to be forwarded to a local Mob_Node, it first updates the relevant vectors that it maintains for the Mob_Node, strips all piggybacked information from the communication, and then forwards it to the Mob_Node. Thus, a Mob_Node sends and obtains application communications that do not contain any additional information; it is only responsible for CGS-accumulation its local state appropriately and transferring it to the Mob-Supp-St. Each procedure $P_i$ can initiate the CGS-accumulation procedure.

An Originator-procedure Mob-Supp-St(say Mob-Supp-$St_{in}$) initiates and coordinates CGS-accumulation procedure on behalf of $Mob\_Node_i$. It computes int_vect (subset of the least-interacting-set on the basis of direct dependencies maintained locally); and sends provisional resident-reinstatement-point invitation (say ad-req) along with int_vect to a Mob-Supp-St, if the later supports at least one procedure in the int_vect. It also updates its tint_vect on the basis of int_vect.

We assume that contemporaneous invocations of the algorithm do not occur. On receiving the ad-req, along with the int_vect from the originator-procedure Mob-Supp-St, a Mob-Supp-St, say Mob-Supp-$St_i$, captures the following actions-

(i) It updates its tint_vect on the basis of int_vect.

(ii) It sends the ad_req to $P_i$ if the following conditions are met:

(a) $P_i$ is running in its cell

(b) $P_i$ is a member of the int_vect and

(c) ad_req has not been directed to $P_i$.

If no such procedure is found, Mob-Supp-$St_i$ ignores the ad_req. Otherwise, on the basis of tint_vect, cv vectors of procedures in its cell, initial cv vectors of other procedures, it computes tnp_int_vect. If tnp_int_vect is not empty, Mob-Supp-$St_i$ sends ad_req along with tint_vect, tnp_int_vect to a Mob-Supp-St, if the later supports at least one procedure in the tnp_int_vect. Mob-Supp-$St_i$ updates np_int_vect, tint_vect on the basis of tnp_int_vect and initializes tnp_int_vect.

On receiving ad_req along with tint_vect, tnp_int_vect from some Mob-Supp-St, a Mob-Supp-St, say Mob-Supp-$St_j$, captures the following actions-

(i) It updates its own tint_vect on the basis of received tint_vect, tnp_int_vect and finds any procedure $P_k$ such that $P_k$ is running in its cell, $P_k$ has not been directed ad_req and $P_k$ is intnp_int_vect.

(ii) If no such procedure exists, it simply ignores this invitation. Otherwise, it sends the provisional resident-reinstatement-point invitation to $P_k$.

(iii) On the basis of tint_vect, cv[] of its procedures and initial cv[] of other procedures, it computes tnp_int_vect. If tnp_int_vect is not empty, Mob-Supp-$St_j$ sends the resident-reinstatement-point invitation along with tint_vect, tnp_int_vect to a Mob-Supp-St, which supports at least one procedure in the tnp_int_vect.

(iv) Mob-Supp-$St_j$ updates np_int_vect, tint_vect on the basis of tnp_int_vect. It also initializes tnp_int_vect.For a disengaged Mob_Node, that is a member of least-interacting-set, the Mob-Supp-St that has its disengaged resident-reinstatement-point, converts its disengaged resident-reinstatement-point into the required one.

(v) When a Mob-Supp-St learns that all of its relevant procedures have captured their provisional reinstatement-points successfully or at least one of its procedures has failed to capture its provisional resident-reinstatement-point, it sends the response communication along with the np_int_vect to the originator-procedure's Mob-Supp-St.

(vi) If, after sending the response communication, a Mob-Supp-St obtains the resident-reinstatement-point invitation along with the tnp_int_vect, and learns that there is at least one procedure in tnp_int_vect running in its cell and it has not captured its partially-committed resident-reinstatement-point, then the Mob-Supp-St invites such procedure to capture resident-reinstatement-point. It again sends the response communication to the originator-procedure Mob-Supp-St.

(vii) When the originator-procedure Mob-Supp-St obtains a response from some Mob-Supp-St, it updates its int_vect on the basis of np_int_vect, received along with the response.

(viii) Finally, originator-procedure's Mob-Supp-St sends partially-committed resident-reinstatement-point invitation to all the procedures of the least-interacting-set. In this case, if some procedure fails to capture provisional resident-reinstatement-point in the first phase, then concerned Mob_Nodes need to abort their provisional reinstatement-points only. The effort of taking a provisional resident-reinstatement-point is insignificant as compared to the partially-committed one. In this way, the loss of CGS-accumulation effort, in case of an abort of the CGS-accumulation procedure, is significantly low.

When a procedure in the least-interacting-set obtains the partially-committed resident-reinstatement-point invitation, it converts its provisional resident-reinstatement-point into partially-committed one.

In the third phase, originator-procedure Mob-Supp-St sends commit or abort to all procedures. On receiving abort, a procedure discards its partially-committed resident-reinstatement-point, if any, and undoes the updating of data structures. On receiving commit, procedures, in the int_vect[], convert their partially-committed reinstatement-points into committed ones. On receiving commit or abort, all procedures update their dependency vectors and other data structures.

## 5. CONCLUSIONS

We have projected a minimum method non-intrusive CGS-accumulation etiquette for deterministic Mobile_DSs, where no inoperable reinstatement_points are captured and no hindering of procedures take place. In minimum methodCGS-accumulation etiquettes, some inoperable reinstatement_points are captured or hindering of procedures takes place; we are success to eliminate both by logging anti-messages of selective communications at the receiver side only during the CGS-accumulation period. The overheads of logging a few anti-messages may be negligible as compared to taking some inoperable reinstatement_points or hindering the procedures during CGS-accumulation, especially in Mobile_DS.

We also try to condense the loss of CGS-accumulation effort when any procedure fails to capture its resident-reinstatement-point in coordination with others in the first phase. In case of a disappointment during CGS-accumulation in the first phase, all applicable procedures need to abort their interim reinstatement_points only. The cost of taking an interim resident-reinstatement-point is negligibly small as compared to the partially-committed one especially in case of Mobile_DSs. In case, some procedure fails to convert its interim resident-reinstatement-point into partially-committed one, then we follow the selective commit mechanism, in which a procedure commits its resident-reinstatement-point if none of the procedure, it causally depends upon, fails to capture its partially-committed resident-reinstatement-point. We disallow contemporaneous executions in spite of contemporaneous instigations of the proposed etiquette.

## REFERENCES


[1] 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.

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

[3] Chandy K. M. and Lamport L., "Distributed Snapshots: Determining Global State of Distributed Systems", ACM Transaction on Computing Systems, Vol. 3, No. 1, pp. 63-75, February 1985.

[4] Elnozahy E. N., Alvisi L., Wang Y. M. and Johnson D. B., "A Survey of Rollback-Recovery Protocols in Message-Passing Systems", ACM Computing Surveys, Vol. 34, No. 3, pp. 375-408, 2002.

[5] Johnson, D. B., Zwaenepoel W., "Recovery in Distributed Systems using Optimistic Message Logging and Checkpointing", Journal of Algorithms, Vol. 11, No. 2, pp. 462-491, 1990.

[6] Johnson, D.B., Zwaenepoel W., "Sender-based Message Logging", In Proceedings of 17th International Symposium on Fault-Tolerant Computing, pp. 14-19, 1987.

[7] David R. Jefferson, "Virtual Time", ACM Transactions on Programming Languages and Systems, Vol. 7, No.3, pp. 404-425, July 1985.

[8] Pushpendra Singh and Gilbert Cabillic, "A Checkpointing Algorithm for Mobile Computing Environment", LNCS, No. 2775, pp. 65-74, 2003.

[9] 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.

[10] Baldoni R., Helary J. M., Mostefaoui A. and Raynal M., "A Communication-Induced Checkpointing Protocol that Ensures Rollback-Dependency Trackability", Proceedings of the International Symposium on Fault-Tolerant-Computing Systems, pp. 68-77, June 1997.

[11] Cao G. and Singhal M., "Mutable Checkpoints: A New Checkpointing Approach for Mobile Computing Systems", IEEE Transaction on Parallel and Distributed Systems, Vol. 12, No. 2, pp. 157-172, February 2001.

[12] Koo R. and Toueg S., "Checkpointing and Roll-Back Recovery for Distributed Systems", IEEE Trans. on Software Engineering, Vol. 13, No. 1, pp. 23-31, January 1987.

[13] Parveen Kumar, Lalit Kumar, R. K. Chauhan and V. K. Gupta "A Non-Intrusive Minimum Process Synchronous Checkpointing Protocol for Mobile Distributed Systems", Proceedings of IEEE ICPWC-2005, pp. 491-495, January 2005.

[14] 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.

[15] J. L. Kim, T. Park, "An efficient Protocol for Checkpointing Recovery in Distributed Systems", IEEE Trans. Parallel and Distributed Systems, pp. 955-960, Aug. 1993.

[16] Silva, L. M. and J. G. Silva, "Global checkpointing for distributed programs", Proceeding, 11thSymp. Reliable Distributed Systems, pp. 155-162, Oct. 1992.

[17] Parveen Kumar, Lalit Kumar and R. K. Chauhan, "A Non-intrusive Hybrid Synchronous Checkpointing Protocol for Mobile Systems", IETE Journal of Research, Vol. 52 No. 2&3, 2006.

[18] Parveen Kumar, "A Low-Cost Hybrid Coordinated Checkpointing Protocol for Mobile Distributed Systems", Mobile Information Systems, IOS Press, Vol. 4, pp. 13–32, 2008.

[19] Lalit Kumar Awasthi and P. Kumar, "A Synchronous Checkpointing Protocol for Mobile Distributed Systems: Probabilistic Approach", International Journal of Information and Computer Security, Vol. 1, No. 3, pp. 298-314.