



# An Adept Synchronous Non-Conflicting Retrieval Line Amassing Protocol for Fault-Tolerant Mobile Distributed Systems

Yogendra Kumar Katiyar, Research Scholar, Department of ECE, Sunrise University, Alwar, Rajasthan, INDIA

Email: [yogendra.katiyar@gmail.com](mailto:yogendra.katiyar@gmail.com)

Dr. Ram Mohan Singh Bhadoria, Associate Professor, Department of ECE, Sunrise University, Alwar, Rajasthan, INDIA

## Abstract

We intend a bottommost-undertaking orchestrated checkpointing/NRL-amassing (Non-conflicting Retrieval Line Amassing) for non-deterministic nomadic distributed setups, where no ineffectual repossession-pinpoints are stockpiled. An exertion has been made to curtail the stalling of undertakings and synchronization dispatch expenses. We capture the partial transitive causal-interrelationships during the normal accomplishment by piggybacking causal-interrelationship arrays onto computation dispatches. Recurrent terminations of Reliable Recovery Line Accumulation tactic may happen in nomadic setups due to exhausted battery, non-voluntary disengagements of Nm\_Nds, or poor cellular connectivity. Therefore, we intend that in the first juncture, all pertinent Nm\_Nds will stockpile intervening repossession-pinpoint only. Interim repossession-pinpoint is stored on the memory of Nm\_Nd only. In this case, if some undertaking fails to stockpile repossession-pinpoint in the first juncture, then Nm\_Nds need to break off their intervening repossession-pinpoints only. In this way, we try to curtail the forfeiture of Reliable Recovery Line Accumulation work when any undertaking fails to stockpile its repossession-pinpoint in harmonization with others.

**Key words:** Fault tolerance, consistent global state, coordinated 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 channels.

In the mobile distributed setup, some of the undertakings are implementing on mobile hosts (Mob\_Nodes). A Nn\_Nd communicates with other nodes of the setup via a special node called mobile support station (Nn\_Sp\_St) [1]. A cell is a geographical area around a Nn\_Sp\_St in which it can support an Nn\_Nd. A Nn\_Nd can change its geographical position freely from one cell to another or even to an area covered by no cell. An Nn\_Sp\_St can have both wired and wireless links and acts as an interface between the static network and a part of the mobile network. Static network connects all Nn\_Sp\_Sts. A static node that has no support to Nn\_Nd can be considered as a Nn\_Sp\_St with no Nn\_Nd.

Checkpoint is defined as a designated place in a program at which normal undertaking is interrupted specifically to preserve the status information necessary to allow resumption of processing at a later time. NRL-amassing is the undertaking of saving the status information. By periodically invoking the NRL-amassing undertaking, one can save the status of a program at regular intervals. If there is a disappointment 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 recovery. The repossession-pinpoint-restart is one of the well-known methods to realize reliable distributed setups. Each undertaking takes a repossession-pinpoint where the local state information is stored in the stable storage. Rolling back an undertaking and again



resuming its accomplishment 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 accomplishment of the undertaking [6, 29, 30, 31].

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 local states, one from each undertaking. The state of channels corresponding to a global state is the set of dispatches sent but not yet received. A global state is said to be "consistent" if it contains no conflicting dispatch; i.e., a dispatch whose receive event is recorded, but its send event is lost. To recover from a disappointment, the setup restarts its accomplishment from a previous consistent global state saved on the stable storage during fault-free accomplishment. This saves all the computation done up to the last retrieval-marked state and only the computation done thereafter needs to be redone. In distributed setups, NRL-amassing can be independent, coordinated [6, 11, 13] or quasi-synchronous [2]. Message Logging is also used for fault tolerance in distributed setups [22, 29, 30, 31].

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

The coordinated NRL-amassing protocols can be classified into two types: stalling and non-stalling. In stalling algorithms, some stalling of undertakings takes place during NRL-amassing [4, 11, 24, 25]. In non-stalling algorithms, no stalling of undertakings is required for NRL-amassing [5, 12, 15, 21]. The coordinated NRL-amassing algorithms can also be classified into following two categories: minimum-undertaking and all undertaking algorithms. In all-undertaking coordinated NRL-amassing algorithms, every undertaking is required to take its repossession-pinpoint in an initiation [6], [8]. In minimum-undertaking algorithms, minimum interacting undertakings are required to take their repossession-pinpoints in an initiation [11].

In minimum-undertaking coordinated NRL-amassing algorithms, an undertaking  $P_i$  takes its repossession-pinpoint only if it is a member of the minimum set (a subset of interacting undertaking). An undertaking  $P_i$  is in the minimum set only if the repossession-pinpoint initiator undertaking is transitively dependent upon it.  $P_j$  is directly dependent upon  $P_k$  only if there exists  $m$  such that  $P_j$  receives  $m$  from  $P_k$  in the current NRL-amassing interval [CI] and  $P_k$  has not taken its enduring repossession-pinpoint after sending  $m$ . The  $i^{\text{th}}$  CI of an undertaking denotes all the computation performed between its  $i^{\text{th}}$  and  $(i+1)^{\text{th}}$  repossession-pinpoint, including the  $i^{\text{th}}$  repossession-pinpoint but not the  $(i+1)^{\text{th}}$  repossession-pinpoint.

In minimum-undertaking NRL-amassing protocols, some useless repossession-pinpoints are taken or stalling of undertakings takes place. In this paper, we intend a minimum-undertaking coordinated NRL-amassing algorithm for non-deterministic mobile distributed setups, where no useless repossession-pinpoints are taken. An exertion has been made to abate the stalling of undertakings and the loss of NRL-amassing exertion when any undertaking fails to take its repossession-pinpoint in coordination with others.

Rao and Naidu [26] suggested a new coordinated NRL-amassing protocol combined with selective sender-based dispatch logging. The protocol is free from the problem of lost dispatches. The term 'selective' implies that dispatches are logged only within a specified interval known as active interval, thereby reducing dispatch logging overhead. All

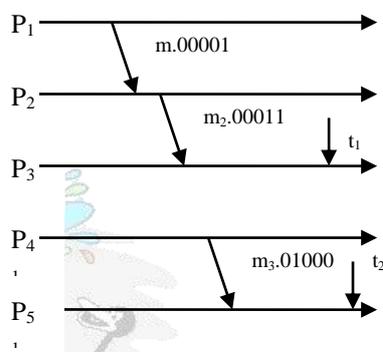


undertakings take repossession-pinpoints at the end of their respective active intervals forming a consistent global repossession-pinpoint. Biswas & Neogy [27] suggested a NRL-amassing and disappointment recovery algorithm where mobile hosts save repossession-pinpoints based on mobility and movement patterns. Mobile hosts save repossession-pinpoints when number of hand-offs exceed a predefined handoff threshold value. Neves & Fuchs [18] designed a time based loosely synchronized coordinated NRL-amassing protocol that removes the overhead of synchronization and piggybacks integer csn (repossession-pinpoint sequence number). Gao et al [28] developed an index-based algorithm which uses time-coordination for consistently NRL-amassing in mobile computing environments. In time-based NRL-amassing protocols, there is no need to send extra coordination dispatches. However, they must deal with the synchronization of timers. This class of protocols suits to the applications where undertakings have high dispatch sending rate.

## 2. Basic Idea

All Communications to and from  $Nm\_Nd$  pass through its proximate  $Nm\_Sp\_St$ . The  $Nm\_Sp\_St$  maintains the causal-interrelationship information of the  $Nm\_Nd$ s which are in its cell. The causal-interrelationship information is kept in Boolean array  $R_i$  for undertaking  $P_i$ . The array has  $n$  bits for  $n$  undertakings. When  $R_i[j]$  is set to 1, it represents  $P_i$  depends upon  $P_j$ . For every  $P_i$ ,  $R_i$  is initialized to 0 except  $R_i[i]$ , which is initialized to 1. When an undertaking  $P_i$  implementing on a  $Nm\_Nd$ , say  $Nm\_Nd_p$ , accepts a dispatch from an undertaking  $P_j$ ,  $Nm\_Nd_p$ 's proximate  $Nm\_Sp\_St$  should set  $R_i[j]$  to 1. If  $P_j$  has stockpiled its enduring repossession-pinpoint after consigning  $m$ ,  $R_i[j]$  is not updated.

Suppose there are undertakings  $P_i$  and  $P_j$  implementing on  $Nm\_Nd$ s,  $Nm\_Nd_i$  and  $Nm\_Nd_j$  with causal-interrelationship arrays  $R_i$  and  $R_j$ . The causal-interrelationship arrays of  $Nm\_Nd$ s,  $Nm\_Nd_i$  and  $Nm\_Nd_j$  are maintained by their proximate  $Nm\_Sp\_St$ s,  $Mobl\_Supp\_St_i$  and  $Mobl\_Supp\_St_j$ . Process  $P_i$  implementing on  $Nm\_Nd_i$  directs dispatch  $m$  to undertaking  $P_j$  implementing on  $Nm\_Nd_j$ . The dispatch is first consigned to  $Mobl\_Supp\_St_i$  (proximate  $Nm\_Sp\_St$  of  $Nm\_Nd_i$ ).  $Mobl\_Supp\_St_i$  maintains the causal-interrelationship array  $R_i$  of  $Nm\_Nd_i$ .  $Mobl\_Supp\_St_i$  appends  $R_i$  with dispatch  $m$  and directs it to  $Mobl\_Supp\_St_j$  (proximate  $Nm\_Sp\_St$  of  $Nm\_Nd_j$ ).  $Mobl\_Supp\_St_j$  maintains the causal-interrelationship array  $R_j$  of  $Nm\_Nd_j$ .  $Mobl\_Supp\_St_j$  replaces  $R_j$  with bitwise logical OR of causal-interrelationship arrays  $R_i$  and  $R_j$  and directs  $m$  to  $P_j$ .



**Figure 1. Maintenance of Dependency Vectors**

In Figure 1, there are five undertakings  $P_1, P_2, P_3, P_4, P_5$  with causal-interrelationship arrays  $R_1, R_2, R_3, R_4, R_5$  initialized to 00001, 00010, 00100, 01000, and 10000 respectively. Initially, every undertaking depends upon itself. Now undertaking  $P_1$  directs  $m$  to  $P_2$ .  $P_1$  appends  $R_1$  with  $m$ .  $P_2$  replaces  $R_2$  with the bitwise logical OR of  $R_1(00001)$  and  $R_2(00010)$ ,



which comes out to be (00011). Now  $P_2$  directs  $m_2$  to  $P_3$  and appends  $R_2$  (00011) with  $m_2$ . Before acquiring  $m_2$ , the value of  $R_3$  at  $P_3$  was 00100. After acquiring  $m_2$ ,  $P_3$  replaces  $R_3$  with the bitwise logical OR of  $R_2$  (00011) and  $R_3$  (00100) and  $R_3$  becomes (00111). Now  $P_4$  directs  $m_3$  along with  $R_4$  (01000) to  $P_5$ . After acquiring  $m_3$ ,  $R_5$  becomes (11000). In this case, if  $P_3$  starts NRL-amassing at  $t_1$ , it will compute the partially-enduring bottommost set equivalent to  $R_3$ (00111), which comes out to be  $\{P_1, P_2, P_3\}$ . In this way, partial transitive causal-interrelationships are captured during normal reckonings.

In orchestrated NRL-amassing, if a single undertaking fails to stockpile its repossession-pinpoint; all the NRL-amassing work goes waste, because, each undertaking must break off its partially-enduring repossession-pinpoint. Furthermore, to stockpile the partially-enduring repossession-pinpoint, a  $N_m\_N_d$  needs to transfer large repossession-pinpoint data to its proximate  $N_m\_S_p\_S_t$  over cellular channels. Hence, the forfeiture of NRL-amassing work may be exceedingly high due to frequent terminations of NRL-amassing strategies especially in nomadic setups. In nomadic distributed setups, there remain certain concerns like: abrupt cessation, exhausted battery power, or letdown in cellular bandwidth. So there remains a good probability that some  $N_m\_N_d$  may fail to stockpile its repossession-pinpoint in harmonization with others. Therefore, we intend that in the first juncture, all undertakings in the bottommost set, stockpile intervening repossession-pinpoint only. Interim repossession-pinpoint is stored on the memory of  $N_m\_N_d$  only. If some undertaking fails to stockpile its repossession-pinpoint in the first juncture, then other  $N_m\_N_d$ s need to break off their intervening repossession-pinpoints only. The work of stockpiling an intervening repossession-pinpoint is negligible as compared to the partially-enduring one. In other strategies, all pertinent undertakings need to break off their partially-enduring repossession-pinpoints in this situation. Hence the forfeiture of NRL-amassing work in case of a break off the NRL-amassing tactic is dramatically low in the suggested scheme as compared to other orchestrated NRL-amassing schemes for nomadic distributed setups.

In this second juncture, an undertaking converts its intervening repossession-pinpoint into partially-enduring one. By using this scheme, we try to curtail the forfeiture of NRL-amassing work in case of break off of NRL-amassing tactic in the first juncture.

A non-stalling NRL-amassing tactic does not require any undertaking to suspend its underlying computation. When undertakings do not suspend their computation, it is possible for an undertaking to accept a computation dispatch from another undertaking, which is already implementing in a new NRL-amassing interval. If this situation is not properly dealt with, it may result in an inconsistency. During the NRL-amassing tactic, an undertaking  $P_i$  may accept  $m$  from  $P_j$  such that  $P_j$  has stockpiled its repossession-pinpoint for the current instigation whereas  $P_i$  has not. Suppose,  $P_i$  undertakings  $m$ , and it accepts repossession-pinpoint appeal later, and then it stockpiles its repossession-pinpoint. In that case,  $m$  will become conflicting in the stockpiled comprehensive status. We intend that only those dispatches, which can become conflicting, should be safeguarded at the consigner's end. When an undertaking stockpiles its intervening repossession-pinpoint, it is not allowed to consign any dispatch till it accepts the partially-enduring repossession-pinpoint appeal. However, in this duration, the undertaking is allowed to perform its normal reckonings and accept the dispatches. When an undertaking accepts the partially-enduring repossession-pinpoint appeal, it is confirmed that every pertinent undertaking has stockpiled its intervening repossession-pinpoint. Hence, a dispatch generated for consigning by an undertaking after getting partially-enduring repossession-pinpoint appeal cannot become conflicting. Hence, an undertaking can consign the safeguarded dispatches after getting the partially-enduring repossession-pinpoint appeal from the inaugurator.



### 3.The Suggested Orchestrated NRL-amassing Tactic

#### First juncture of the tactic:

When an undertaking, say  $P_i$ , implementing on a  $Nm\_Nd$ , say  $Nm\_Nd_i$ , triggers a NRL-amassing, it directs a repossession-pinpoint instigation appeal to its proximate  $Nm\_Sp\_St$ , which will be the proxy  $Nm\_Sp\_St$  (if the inaugurator runs on an  $Nm\_Sp\_St$ , then the  $Nm\_Sp\_St$  is the proxy  $Nm\_Sp\_St$ ). The proxy  $Nm\_Sp\_St$  maintains the causal-interrelationship array of  $P_i$  say  $R_i$ . Based on  $R_i$ , the set of dependent undertakings of  $P_i$  is formed, say  $S_{minset}$ . The proxy  $Nm\_Sp\_St$  broadcasts ckpt ( $S_{minset}$ ) to all  $Nm\_Sp\_Sts$ . When an  $Nm\_Sp\_St$  accept ckpt ( $S_{minset}$ ) dispatch, it checks, if any undertakings in  $S_{minset}$  are in its cell. If so, the  $Nm\_Sp\_St$  directs intervening repossession-pinpoint appeal dispatch to them. Any undertaking acquiring an intervening repossession-pinpoint appeal stockpiles an intervening repossession-pinpoint and directs a rejoinder to its proximate  $Nm\_Sp\_St$ . After an  $Nm\_Sp\_St$  acknowledged all rejoinder dispatches from the undertakings to which it consigned intervening repossession-pinpoint appeal dispatches, it directs a rejoinder to the proxy  $Nm\_Sp\_St$ . It should be noted that in the first juncture, all undertakings stockpile the intervening repossession-pinpoints. For an undertaking implementing on a static host, intervening repossession-pinpoint is equivalent to partially-enduring repossession-pinpoint. But, for a  $Nm\_Nd$ , intervening repossession-pinpoint is divergent from partially-enduring repossession-pinpoint. To stockpile a partially-enduring repossession-pinpoint, a  $Nm\_Nd$  has to record its proximate status and has to transfer it to its proximate  $Nm\_Sp\_St$ . But, the intervening repossession-pinpoint is stored on the proximate disk of the  $Nm\_Nd$ . It should be noted that the work of stockpiling an intervening repossession-pinpoint is very small as compared to the partially-enduring one. For a disengaged  $Nm\_Nd$  that is a member of bottommost set, the  $Nm\_Sp\_St$  that has its disengaged repossession-pinpoint, considers its detached repossession-pinpoint as the required come.

#### Second Segment of the tactic:

After the proxy  $Nm\_Sp\_St$  has acknowledged the rejoinder from every  $Nm\_Sp\_St$ , the tactic enters the second juncture. If the proxy  $Nm\_Sp\_St$  learns that all applicable undertakings have stockpiled their intervening repossession-pinpoints successfully, it directs them to convert their intervening repossession-pinpoints into partially-enduring ones and directs the exact bottommost set along with this appeal. Alternatively, if inaugurator  $Nm\_Sp\_St$  comes to know that some undertaking has failed to stockpile its repossession-pinpoint in the first juncture, it sends break off appeal to all  $Nm\_Sp\_St$ . In this way the  $Nm\_Nds$  need to break off only the intervening repossession-pinpoints, and not the partially-enduring ones. In this way we try to reduce the forfeiture of NRL-amassing work in case of break off of NRL-amassing tactic in first juncture.

When an  $Nm\_Sp\_St$  accepts the partially-enduring repossession-pinpoint appeal, it directs all the undertaking in the bottommost set, which are also implementing, to convert their intervening repossession-pinpoints into partially-enduring ones. When an  $Nm\_Sp\_St$  learns that all applicable undertaking in its cell have stockpiled their partially-enduring repossession-pinpoints successfully, it directs rejoinder to proxy  $Nm\_Sp\_St$ . If any  $Nm\_Nd$  fails to transfer its repossession-pinpoint data to its proximate  $Nm\_Sp\_St$ , then the letdown rejoinder is consigned to the proxy  $Nm\_Sp\_St$ ; which in turn, concerns the break off dispatch.

#### Third Segment of the tactic:

Finally, when the proxy  $Nm\_Sp\_St$  learns that all undertakings in the bottommost set have stockpiled their partially-enduring repossession-pinpoints successfully, it concerns commit appeal to all  $Nm\_Sp\_Sts$ . When an undertaking in the bottommost set gets the commit appeal, it converts its partially-enduring repossession-pinpoint into enduring one and discards



its earlier enduring repossession-pinpoint, if any.

### Message Handling During NRL-amassing:

When an undertaking stockpiles its intervening repossession-pinpoint, it does not consign any message till it accepts the partially-enduring repossession-pinpoint appeal. This time duration of an undertaking is called its uncertainty period. Suppose,  $P_i$  directs  $m$  to  $P_j$  after stockpiling its intervening repossession-pinpoint and  $P_j$  has not stockpiled its intervening repossession-pinpoint at the time of acquiring  $m$ . In this case, if  $P_j$  stockpiles its intervening repossession-pinpoint after working  $m$ , then  $m$  will become conflicting. Therefore, we do not allow  $P_i$  to consign any message unless and until every undertaking in the bottommost set have stockpiled its intervening repossession-pinpoint in the first juncture.  $P_i$  can consign messages when it accepts the partially-enduring repossession-pinpoint appeal; because, at this moment every pertinent undertaking has stockpiled its intervening repossession-pinpoint and  $m$  cannot become conflicting. The messages to be consigned are safeguarded at consigner's end. In this duration, an undertaking is allowed to continue its normal reckonings and accept messages.

Suppose,  $P_j$  gets the intervening repossession-pinpoint appeal at  $Mobl\_Supp\_St_p$ . Now, we find any undertaking  $P_k$  such that  $P_k$  does not belong to  $S_{minset}$  and  $P_k$  belongs to  $R_j[]$ . In this case,  $P_k$  is also included in the bottommost set; and  $P_j$  directs intervening repossession-pinpoint appeal to  $P_k$ . It should be noted that the  $S_{minset}$ , computed based on causal-interrelationship array of inaugurator undertaking is only a subset of the bottommost set. Due to zigzag causal-interrelationships, inaugurator undertaking may be transitively dependent upon some more undertakings which are not included in the  $S_{minset}$  computed initially.

### 3. An Example of the Suggested Protocol

The suggested tactic can be better understood by the example shown in Figure 2. There are six undertakings ( $P_0$  to  $P_5$ ) denoted by straight lines. Each undertaking is assumed to have initial enduring repossession-pinpoints with  $chkpt\_seq\_no$  equal to "0".  $C_{ix}$  denotes the  $x^{th}$  repossession-pinpoints of  $P_i$ . Initial causal-interrelationship arrays of  $P_0, P_1, P_2, P_3, P_4, P_5$  are [000001], [000010], [000100], [001000], [010000], and [100000], respectively. The causal-interrelationship arrays are maintained as explained in Section 3.3.

$P_0$  directs  $m_2$  to  $P_1$  along with its causal-interrelationship array [000001]. When  $P_1$  accepts  $m_2$ , it computes its causal-interrelationship array by stockpiling bitwise logical OR of causal-interrelationship arrays of  $P_0$  and  $P_1$ , which comes out to be [000011]. Similarly,  $P_2$  updates its causal-interrelationship array on acquiring  $m_3$  and it comes out to be [000111]. At time  $t_1$ ,  $P_2$  triggers NRL-amassing tactic with its causal-interrelationship array is [000111]. At time  $t_1$ ,  $P_2$  finds that it is transitively dependent upon  $P_0$  and  $P_1$ . Therefore,  $P_2$  computes the partially-enduring bottommost set [ $S_{minset} = \{P_0, P_1, P_2\}$ ].  $P_2$  directs the intervening repossession-pinpoint appeal to  $P_1$  and  $P_0$  and stockpiles its own intervening repossession-pinpoint  $C_{21}$ . For a  $Nm\_Nd$  the intervening repossession-pinpoint is stored on the disk of  $Nm\_Nd$ . It should be noted that  $S_{minset}$  is only a subset of the bottommost set. When  $P_1$  stockpiles its intervening repossession-pinpoint  $C_{11}$ , it finds that it is dependent upon  $P_3$  due to  $m_4$ , but  $P_3$  is not a member of  $S_{minset}$ ; therefore,  $P_1$  directs intervening repossession-pinpoint appeal to  $P_3$ . Consequently,  $P_3$  stockpiles its intervening repossession-pinpoint  $C_{31}$ .

After stockpiling its intervening repossession-pinpoint  $C_{21}$ ,  $P_2$  generates  $m_8$  for  $P_3$ . As  $P_2$  has already stockpiled its intervening repossession-pinpoint for the current instigation and it has not acknowledged the partially-enduring repossession-pinpoint appeal from the inaugurator; therefore,  $P_2$  buffers  $m_8$  on its proximate disk. We define this duration as the uncertainty period of an undertaking during which an undertaking is not allowed to consign any message. The messages generated for consigning are safeguarded at the proximate disk of the consigner's undertaking.  $P_2$  can direct  $m_8$  only after getting partially-enduring repossession-pinpoint appeal or break off messages from the inaugurator undertaking. Similarly, after



stockpiling its intervening reposition-pinpoint  $P_0$  buffers  $m_{10}$  for its uncertainty period. It should be noted that  $P_1$  accepts  $m_{10}$  only after stockpiling its intervening reposition-pinpoint. Similarly,  $P_3$  accepts  $m_8$  only after stockpiling its intervening reposition-pinpoint  $C_{31}$ . An undertaking is allowed to accept all the messages during its uncertainty period; for example,  $P_3$  accepts  $m_{11}$ . An undertaking is also allowed to perform its normal reckonings during its uncertainty period.

At time  $t_2$ ,  $P_2$  accepts responses to intervening reposition-pinpoints appeals from all undertaking in the bottommost set (not shown in the Figure 2) and finds that they have stockpiled their intervening reposition-pinpoints successfully; therefore,  $P_2$  concerns partially-enduring reposition-pinpoint appeal to all undertakings. On getting partially-enduring reposition-pinpoint appeal, undertakings in the bottommost set [ $P_0, P_1, P_2, P_3$ ] convert their intervening reposition-pinpoints into partially-enduring ones and consign the rejoinder to inaugurator undertaking  $P_2$ ; these undertaking also consign the messages, safeguarded at their proximate disks, to the destination undertakings For example,  $P_0$  directs  $m_{10}$  to  $P_1$  after getting partially-enduring reposition-pinpoint appeal [not shown in the figure]. Similarly,  $P_2$  directs  $m_8$  to  $P_3$  after getting partially-enduring reposition-pinpoint appeal. At time  $t_3$ ,  $P_2$  accepts responses from the undertaking in bottommost set [not shown in the figure] and finds that they have stockpiled their partially-enduring reposition-pinpoints successfully, therefore,  $P_2$  concerns commit appeal to all undertaking. An undertaking in the bottommost set converts its partially-enduring reposition-pinpoint into enduring reposition-pinpoint and discards it old enduring reposition-pinpoint if any.

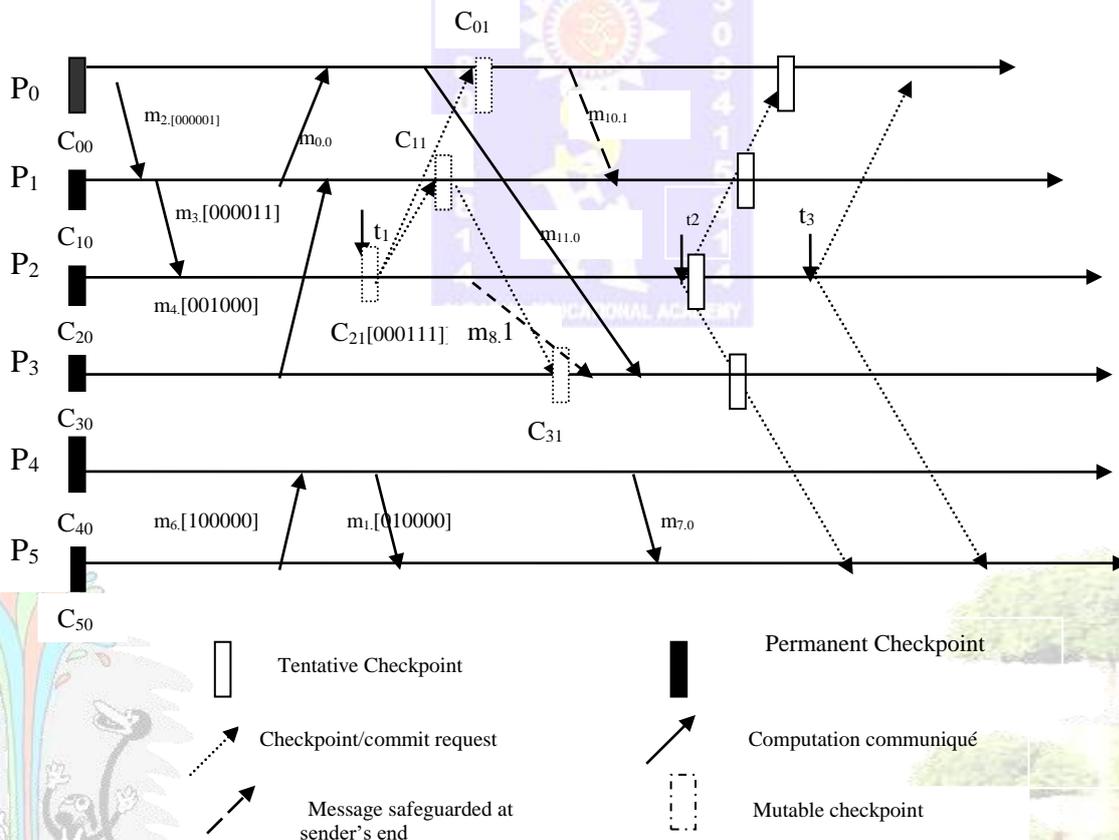


Figure 2

#### 4. Conclusion

We have designed a bottommost-interacting-undertaking synchronous NRL-amassing mechanism for mobile distributed setup. We try to abate the stalling of undertakings during



NRL-amassing . The stalling time of an undertaking is bare bottommost. During stalling period, undertakings can do their normal working outs, consign dispatches and can undertaking selective dispatches. The number of undertakings that seize repossession-pinpoints is minimized to avoid awakening of Nm\_Nds in doze mode of undertaking and thrashing of Nm\_Nds with NRL-amassing activity. It also saves limited battery life of Nm\_Nds and low bandwidth of wireless channels. We try to reduce the loss of NRL-amassing exertion when any undertaking miscarries to seize its repossession-pinpoint in synchronization with others. We also try to abate the synchronization dispatches during NRL-amassing. In the planned scheme, no synchronization dispatches are sent to enter the second or third Segment of the mechanism.

### References

- [1] A. Acharya and B. R. Badrinath, *Checkpointing Distributed Applications on Mobile Computers*, In Proceedings of the 3<sup>rd</sup> 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 11<sup>th</sup> 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 28<sup>th</sup> 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 26<sup>th</sup> International Symposium on Fault-Tolerant Computing, 1996, 16-25.
- [21] R. Prakash and M. Singhal, Low-Cost Checkpointing and Disappointment 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 11<sup>th</sup> symposium on Reliable Distributed Systems, 1992, 155-62.
- [24] Sunil Kumar, R K Chauhan, Parveen Kumar, "A Minimum-process Coordinated Checkpointing Protocol for Mobile Computing Systems", *International Journal of Foundations of Computer science*, Vol 19, No. 4, pp 1015-1038 (2008).
- [25] Parveen Kumar, "A Low-Cost Hybrid Coordinated Checkpointing Protocol for mobile distributed systems", *Mobile Information Systems*. pp 13-32, Vol. 4, No. 1, 2007.
- [26] Rao, S., & Naidu, M.M, "A New, Efficient Coordinated Checkpointing Protocol Combined with Selective Sender-Based Message Logging", *IEEE/ACSI International Conference on Computer Systems and Applications*, 2008.
- [27] Biswas S, & Neogy S, "A Mobility-Based Checkpointing Protocol for Mobile Computing System", *International Journal of Computer Science & Information Technology*, Vol.2, No.1, pp135-15, 2010.
- [28] Gao Y., Deng C., & Che, Y., "An Adaptive Index-Based Algorithm Using Time-Coordination in Mobile Computing", *International Symposiums on Information Processing*, pp.578-585, 2008.
- [29] Praveen Choudhary, Parveen Kumar, "Minimum-Process Global-Snapshot Accumulation Etiquette for Mobile Distributed Systems", *International Journal of Advanced Research in Engineering and Technology* Vol. 11, Issue 8, Aug 20, pp.937-948
- [30] Praveen Choudhary, Parveen Kumar, "Low-Overhead Minimum-Method Global-Snapshot Compilation Protocol for Deterministic Mobile Computing Systems", *International Journal of Emerging Trends in Engineering Research* Vol. 9, Issue 8, Aug 2021, pp.1069-1072

