Military Technical College
Kobry El-Kobbah,
Cairo, Egypt

8th International Conference
on Electrical Engineering
ICEENG 2012

Non-blocking minimum processes coordinated checkpointing for hierarchical computational grid

By

Gamal A. El-Sayed* Aref M. Abdullah**

Abstract:

Fault tolerance is an important property in grid computing as the dependability of individual grid resources may not be able to be guaranteed. Common fault tolerance techniques in distributed systems are normally achieved with checkpoint recovery, message logging with checkpointing, or through task replication on alternative resources in cases of a system outage. In this paper, we present a mailbox-based non-blocking minimum processes coordinated checkpoint protocol for hierarchical grid. In our grid model, processes on different processors communicate indirectly by sending messages over the network through mailbox-based technique at a shared node. The mailbox of each process can be exploited as an events logger since it logs the messages sent to the process in strict FIFO order. The main advantages of our approach are achieving more parallelism and suiting the highly dynamic environment where processes frequently migrate from one node to another.

Keywords:

Coordinated checkpointing, fault tolerant, non-blocking, message logging.

---

* *Electrical Engineering Department, Assiut University,* Egypt
** *Electrical Engineering Department, Assiut University,* Egypt

1. Introduction:

The emergence of grid computing further increases the importance of fault tolerance. Grid computing will impose a number of unique new conceptual and technical challenges to fault-tolerance researchers. Some of the factors due to which the probability of faults in a grid environment is much higher than a traditional distributed system are lack of centralized environment, predominant execution of long jobs, highly dynamic resource availability, and diverse geographical distribution of resources and heterogeneous nature of grid resources. Thus, fault tolerance related features must be incorporated in grid job scheduling to improve the performance of the grid system [1].

Checkpoint-restart is the ability to save the state of the running application at a given point in time such that it can be restored at a later time at the exact same point at which it was saved. Checkpoint-restart has tremendous potential benefits for grid computing environments, including fault resilience by migrating applications out of faulty Grid nodes, fault recovery by restarting from the last checkpoint instead of from scratch, improved resource utilization by being able to checkpoint resource-intensive jobs when load is high and restarting such jobs again later when the load is lower, dynamic load balancing by migrating application processes to less loaded Grid nodes, and improved service availability and administration by checkpointing applications processes before grid node maintenance and restarting them on other cluster nodes so that applications can continue to run with minimal downtime[2].

A global state of a message-passing system is a collection of the individual states of all participating processes and of the states of the communication channels. Intuitively, a consistent global state is one that may occur during a failure-free, correct execution of a distributed computation. More precisely, a consistent system state is one in which, if the state of a process reflects a message receipt, then the state of the corresponding sender reflects sending that message. For example, figure 2 shows two examples of global states.

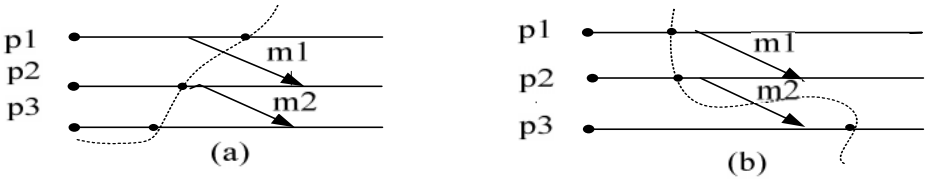


Figure (1): *(a) consistent state (b) inconsistent state*

On the other hand, in message logging approach the processes that are participating in a distributed computation take checkpoints (independent checkpoints) and log the messages that they exchange during failure-free operation. When a failure occurs, a

recovery algorithm uses the message logs and checkpoints available on stable storage to compute a consistent state [3] to which the processes roll back. It has been argued that such a design results in better failure-free performance than coordinated checkpointing because it avoids the overhead of synchronizing the checkpoints to form a consistent state. This premise is true for environments where communication is expensive, because the incremental cost of logging then becomes small, while the message exchanges necessary to synchronize the checkpoints add substantial overhead [4]. However, the cost of network communication these days is small and rapidly decreasing. Meanwhile, the cost of accessing stable storage remains high because of the mechanical nature of disks, which are the media of choice for implementing stable storage because of the cost per capacity advantage over other techniques. Recent experiments [5] have indicated that the difference in performance between coordinated and independent checkpointing becomes marginal in workstation clusters.

In this paper, we present a mailbox-based non-blocking minimum processes coordinated checkpointing based on hierarchical computational grid, where workstations and computation nodes can be grouped into several sites according to their physical locations. The proposed checkpointing algorithm, which uses the mailbox, enables the grid with highly dynamic and non dedicated resources to complete the jobs within specified deadline. Major contributions of our new technique include:

- The number of processes that take checkpoints is minimized, resulting in a reduction in stable storage access and enhanced performance.
- Our approach exploits the mailbox as a logger aiming also to reduce the frequently access to stable storage to enhance performance. Also, the communication overhead is minimized since the process suspends its computations only during taking a tentative checkpoint, after that, the process can receive/send a computation message from/to another process.
- The messages sent to the faulty process during its migration will not be lost. They will be kept in its mailbox and the process will pull them after restarting on the new resource.

The paper is structured as follows. In Section 2, we present literature review. In Section 3, we introduce our communication model and checkpointing protocol. In Section 4, proof of Correctness is presented, and finally in Section 5 we conclude the paper.

2. Background:

Numerous approaches to checkpointing and rollback recovery were proposed in the literature for the field of message-passing distributed systems. The variety of existing algorithms are classified according to their main operating principles.

2.1 Coordinated checkpointing:

This approach has been introduced by Chandy and Lamport [3]. In this approach, the checkpointing is orchestrated such that the set of individual checkpoints always result in a consistent global checkpoint. Coordinated checkpointing algorithms are made up using the following scheme [6][7].

2.1.1 Blocking coordinated checkpointing

In this approach, the coordinated checkpoint processes are synchronized before the local checkpoints in order to ensure a clean communication channel [9]. A coordinator broadcasts a checkpoint request for every process. When a process receives such a request, it stops its execution, flushes its communication channels, takes a local checkpoint, and sends an acknowledgement message back to the coordinator. After that the coordinator collects acknowledgements from all processes, and broadcasts a commit message to complete the two-phase checkpoint protocol. Upon receiving the commit message, each process marks its local checkpoint as a new recovery line. Then the process resumes execution and exchanges messages with other processes [8].

2.1.2 Non-blocking coordinated checkpointing:

The performance overheads of the blocking coordination are difficult to avoid. That is why in practice a non-blocking scheme is preferred [9]. This protocol is also known as the Distributed snapshot protocol and was also proposed by Chandy and Lamport in 1985[3]. The initiator takes a checkpoint and broadcasts a marker (i.e. a checkpoint request) to all processes. Each process takes a checkpoint upon receiving the first marker and rebroadcasts the marker to all the other processes before sending any application message. The underlying assumption is that the communication channels are FIFO based and reliable. If the channels are non-FIFO, the marker can be piggybacked on every post-checkpoint message. Alternatively, checkpoint indices can serve as markers, whereby a checkpoint is triggered if the receiver's local checkpoint index is lower than the piggybacked checkpoint index [10].

2.1.3 All processes checkpointing:

This requires all processes in the system to participate in every checkpointing session [6].

2.1.4 Minimum processes checkpointing:

These algorithms force only those processes, which communicated with the initiator directly or indirectly since their last checkpoint, to take new checkpoints [6].

2.2 Uncoordinated (independent) checkpointing:

In this approach, which is also called asynchronous checkpointing, each process that is part of an application saves its state whenever it wants to, without coordinating with other processes [7]. During restart, the consistent global state has to be searched by tracking the dependencies from the stable storage. The main advantage of this approach is that there is no need to exchange any control messages during checkpointing. But this requires each process to keep several checkpoints in stable storage and there is no certainty that a global consistent state can be built [10]. Also, in this method each process can make a checkpoint when its state is small. However, there are two main disadvantages. First, there is a possibility of rollback propagation which can result in a domino effect. Second, the possibility of rollback propagation requires the storage of multiple checkpoints for each process.

2.3 Communication-induced checkpointing:

This approach is a compromise between coordinated and uncoordinated checkpointing. To avoid a domino effect that can result from independent checkpoints of different processes, a consistent global state is achieved by forcing each process to take additional checkpoints based on some information piggybacked on the application messages [11].The disadvantage of this approach is the possibly large number of forced checkpoints and the overhead associated with storing them.

2.4.Log-based rollback recovery:

It combines checkpointing with logging of nondeterministic events. Log-based rollback recovery relies on the *piecewise deterministic* (*PWD*) assumption, which postulates that all nondeterministic events that a process executes can be identified and that the information necessary to replay each event during recovery can be logged in the event's *determinant* [10].

Log-based rollback-recovery protocols guarantee that upon recovery of all failed processes, the system does not contain any orphan process, that is, a process whose state depends on a nondeterministic event that cannot be reproduced during recovery. The way in which a specific protocol implements this condition affects the protocol's failure-free performance overhead, latency of output commit, and simplicity of recovery

and garbage collection. There are three flavors of these protocols [10] :

2.4.1. Pessimistic Log-based rollback recovery protocols:

They guarantee that orphans are never created due to a failure. These protocols simplify recovery, garbage collection and output commit, at the expense of higher failure free performance overhead [10].

2.4.2. Optimistic log-based rollback recovery protocols:

They reduce the failure-free performance overhead, but allow orphans to be created due to failures. The possibility of having orphans complicates recovery; garbage collection and output commit [10].

2.4.3. Causal log-based rollback recovery protocol:

They attempt to combine the advantages of low performance overhead and fast output commit, but they may require complex recovery and garbage collection [10].

3. Related work:

Minimum-process blocking algorithms have the lowest synchronization overhead in the comparison of all-process blocking algorithms. The algorithms proposed in *Koo-Toueg* [12], *Cao- Singhal* [13] have the lowest among the blocking algorithms, [14] tries to minimize the number of synchronization messages and the number of checkpoints during checkpointing. In algorithm [12], if any of the relevant process is not able to take its checkpoint in an initiation, the entire checkpointing process of that particular initiation is aborted. *Kim and Park* [15] proposed an improved scheme to address failures during checkpointing. It allows the new checkpoints in some subtrees to be committed. *Cao and Singhal* [13] proposed minimum-process blocking algorithm for mobile systems. In this algorithm, blocking time is significantly reduced as compared to [12]. *Prakash-Singhal* algorithm [16] forces only a minimum number of processes to take checkpoints and does not block the underlying computation during checkpointing. *Elnozahy and Zwaenepoel* [4] proposed a message logging protocol which uses coordinated checkpointing with message logging. The combination of our non-blocking minimum coordinated checkpointing with message logging using the mailbox queues as events loggers leads to better failure-free performance, low storage and communication overheads, also, only the failed processes need to roll back from their last saved checkpoint and then they can replay the messages from their associated mailbox to reach the pre-failure state.

4. Our communication model and checkpoint protocol:

In this section new transparent coordinated non-blocking checkpointing algorithm that ensures producing global consistent checkpoints is presented.

4.1 System model and application definitions:

Grid computing is a means of allocating the computational power of a large number of computers to complex difficult computation or problem. Grid computing is a distributed computing paradigm that differs from traditional distributed computing in that it is aimed toward large scale systems that even span organizational boundaries.

In this paper, our grid model is based *4-level* hierarchical and distributed scheme as shown in *Figure (1)*. The proposed grid consists of several zones each with many sites. The site in a zone consists of several heterogeneous workstations which that can be grouped according to their physical locations. In our model, at the top of our hierarchical model tree - *level 3*, there is a Super Grid Scheduler (SGS) in the super grid master node to which the users submit their jobs, in *level 2* there are zone schedulers in zone master node, sites schedulers in *level 1*, and at the *level 0* home schedulers run on each computing resources.

In our model, processes of a particular job on different processors only communicate indirectly by sending messages over the network through mailboxes which are dynamically created in mailbox pool which resides at a site, zone or super grid master node before dispatching the job to the selected resources as illustrated below in next subsection.

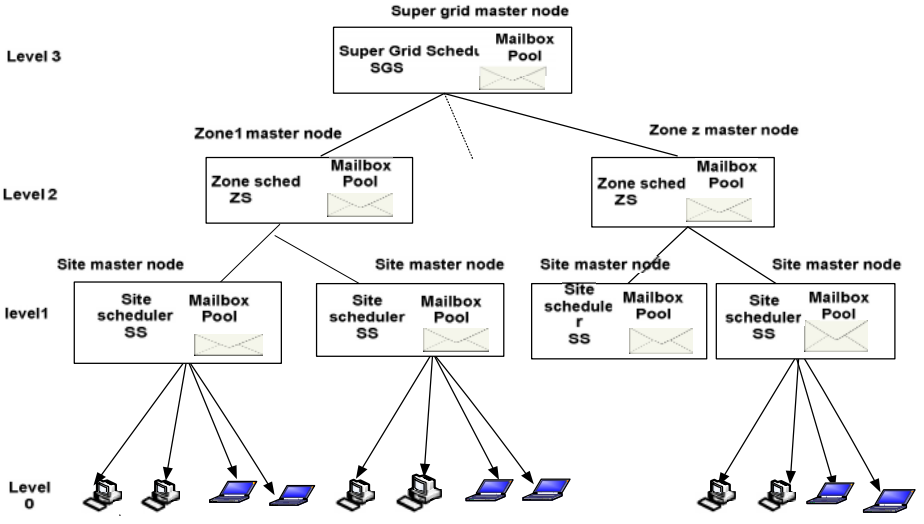
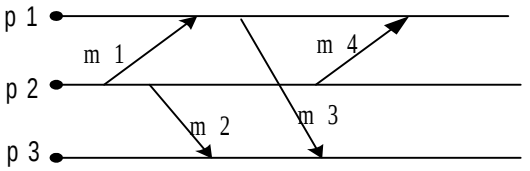


Figure (2): *The hierarchical architecture of computing grid*

4.2 Our mailbox-based communication scheme:

In our scheme, before dispatching the processes to selected resources, each process is assigned a mailbox queue by scheduler at the master node which has all selected resources under its control to buffers the messages sent to it. For example if all the selected resources are in a single site, the mailboxes for all processes of the job are created in mailbox pool which residing at that site master node which is different from the nodes where those processes will be executed. Inter-process communication between processes can occur as follow:

• Transmission of a message from the sender to the receiver's mailbox: If a process P_i wants to send a message m_i to a process P_j, it sends the message to the shared port at the master node where the communication daemon in turn, puts the received message in the mailbox of P_j .
• Delivery of the message from the mailbox to its owner process: The receiving process can use a push or pull operation to obtain the message from the mailbox as illustrated below:
Push (PS). The mailbox keeps its owner process's address and forwards every message to it. Although message queries incur no costs, the scheduler must notify the mailbox of the current location of its owner process after scheduling or rescheduling process. PS mode is needed if real-time message delivery is required.
Pull (PL). The process retrieves messages from its mailbox queue (in order) whenever needed. The mailbox doesn't need to know the process's current location, thus avoiding location registration, but the process must query its mailbox for messages. Although polling messages would increase message delivery overhead, it ensures reliable message delivery and very useful technique in migration.
In this paper we will use the pull (PL) operation in which the primitive get (addr, mailbox [y]) is used to retrieve messages from the mailbox. Thus, if a process wants to retrieve a message, it pulls the one at the top location of its mailbox queue.



(a)

Mailbox of P_1			m_4	m_1
Mailbox of P_2				
Mailbox of P_3			m3	m2

(b)

***Figure (3)**: (a) A part of an application's execution (b) Mailboxes pool view.*

4.3 Notation, Definitions and Assumptions:

In our checkpointing algorithm we use the following definition:

Definition 1: task or process is the minimum computing unit of a job; i.e. Job consists of *N* dependent processes.
Definition 2: application is an arbitrary job: *Jb* = {P_0 , P_1, P_2, ….P_{N-1}}.
Definition 3: the X_{th} local checkpoint of a process P_i is denoted as$C_{i,x}$.
Definition 4: the checkpoint request message is denoted as *chkpt-req*.
Definition 5: the commit message request is denoted as *commit-req*, and the discard request is denoted as *abort-req.*
Definition 6: min-group{ } contains the processes that are communicated since last committed checkpoint, and *num-group* { } contains the corresponding counters of the messages for each process including in *min-group*{ }, which the process must receive before taking the new tentative checkpoint (these counters contain the number of messages sent to each process p_i □ *min-group*{ } since their last committed checkpoint).
Definition 7: S_i and R_i represent the total sent and received by a process P_i since last committed checkpoint.
Definition 8: we assume that the computation messages format includes two fields for the sender and receiver *IDs* as *SPid* and *RPid* respectively.
Definition 9: SSN is a sequence number of messages sent by the process. This is used for duplicate message suppression during recovery. Distributed systems that do not provide fault tolerance typically already require such a sequence number for suppression of duplicate messages.
We have made the following assumptions

1- Checkpoint coordinator and communication daemon at master node does not fail.

2- A separate monitoring software system is used to monitor continuously the failure of a fault tolerant MPI application.

3- Communication failures do not partition the network.

4- The job is submitted to the super grid scheduler which interacts with zones schedulers to decide whether the job will be execute inside single site, multisite within single zone, or in multi zones.

5- Our scheme is built on top of a reliable network communication layer, which guarantees that messages will not be lost during transmission and will be delivered between nodes.

4.4 The checkpointing protocol:

1- Before the job's tasks are dispatched to the selected resources, the site, zone, or super grid zone scheduler, which all selected resources are located under its control, will create a dynamic mailbox queue for each task at mailbox pool, and then initialize the checkpoint coordinator at the same master node.

2- The communication daemon, which is an intermediate party resides at the master node of the selected location, is responsible of receiving messages from sending processes and putting them in corresponding mailbox of receiving processes, and constructing *min-group*{ } & *num-group* { } where the *min-group*{ } contains the processes that are communicated since last committed checkpoint, and *num-group* { } contains their counters of the messages which they must receive before taking the new tentative checkpoint (since their last committed checkpoints).

3- The checkpoint coordinator, during the first phase, triggers the communication daemon (using a shared flag) which in turn, sends the *min-group* { } & *num-group* { } to the coordinator before receiving any extra messages. Then the coordinator send the checkpoint request (*chkpt-req*, num_i) to each p_i □ *min-group*{ } and attaches with it the corresponding counter of the messages num_i each process must receive before taking a new checkpoint. When a process *pi* receives this message, it compares num_i with its local counter R_i, if they are equal it takes a tentative checkpoint which includes the index *y* which points to the location of the mailbox queue corresponding to the entry immediately following the checkpoint, and then sends back to the coordinator the "*ready*" message along with the number of messages sent (S_i) since the last committed checkpoint. But if num_i isn't equal its local counter, the process waits for receiving the rest messages (num_i minus the local counter) and then takes a tentative checkpoint and responds as above. But if the process couldn't take a tentative checkpoint (or the checkpoint timer timeout), it responds with "*not-ready*" message.

4- If the coordinator receives "*ready*" from each p_i □ *min-group*{ } which participated in this checkpoint and $\sum_{i=0}^{n} S_i$ matches $\sum_{i=0}^{n} R_i$, where $\sum_{i=0}^{n} R_i = \sum_{i=0}^{n} num_i$ and *n* is the number of processes in *min-group*{ }, it sends "*commit-req*" to each p_i □ *min-group*{ } that completes the two-phase checkpointing protocol. Otherwise, it sends them the "*abort-req*".

5- Finally, If each p_i □ *min-group*{ } receives "*commit-req*" massage, it removes the old checkpoint and atomically makes the tentative checkpoint permanent.

4.5 The advantages:

The proposed protocol exploits the mailbox of each process as logging storage since the messages, sent to that process, are implicitly stored in its mailbox in strict FIFO. This combination offers several advantages including:
First, since the messages sent to any process is stored in its mailbox in strict FIFO order, there is no need to save message data or any dependency information to stable storage or elsewhere. Second, the number of processes that take checkpoints is minimized, resulting in a reduction in stable storage access and enhanced performance. Also, the communication overhead is minimized since the process suspends it computations only during taking a tentative checkpoint, after that, the process can receive/send a computation message from/to another process. Third, no explicit garbage collection algorithm is needed. Events that occurred before the last consistent checkpoint will never be rolled backed, so it can delete from the mailboxes. *Fourth*, the use of coordinated checkpointing guarantees that processes never roll back beyond their latest checkpoint. This new design thus offer a better bound on recovery time and requires only one permanent checkpoint per process to be maintained on stable storage. *Fifth*, the messages sent to any process cannot be lost due to migration and they kept in the mailbox.
These lead to better performance and improve the scalability which makes it more suitable for grid environment.

4.6 Proof of Correctness:

Theorem 1: our proposed checkpointing protocol presented in subsection 4.4 produces a global consistent checkpoint.

We use proof by contradiction to prove theorem 1. Let us assume that the last obtained checkpoint is inconsistent. This means either one of the following two scenarios have taken place:

1. Process p_i sent a message after taking a checkpoint and this message is polled at process p_j before it takes checkpoint. This means this message will be included in checkpoint for p_j but not in the checkpoint for p_i. By going back to the algorithms, we find a contradiction since the total number of messages num_j which a process p_j must receive before taking a checkpoint (since its last committed checkpoint) is included with the checkpoint request. So, before taking a checkpoint R_j must equal num_j, where R_j is the local counter of received messages since last committed checkpoint and num_j is the one attached with the checkpoint request. Thus if the process p_j polled that extra message sent by the process p_i, this will make $R_j = num_j + 1$, and thus process p_j will not take a checkpoint when the checkpoint request arrives.

2. Process p_i sent a message before it takes checkpoint and this message was polled by process p_j after it takes checkpoint. This means this message will be included in checkpoint for p_i but not in the checkpoint for p_j. But by going back to the algorithm above, we find a contradiction since the coordinator send the commit request to the participating processes which are included in *min-group*{} if and only if $\sum_{i=0}^{n} S_i = \sum_{i=0}^{n} R_i$. Otherwise, the coordinator will send discard message the participating processes.

Note: the method used to prove the *scenario 2* can also be used to prove the scenario 1.

6. Conclusion:

In this paper we presented a new non-blocking minimum processes coordinated checkpointing for hierarchical and distributed grid. The processes in our model communicate indirectly through mailbox pool reside at the cluster master node. We use that mailbox pool as messages logger since messages sent to each process stored in its mailbox in strict FIFO order. Our proposed method reduces the storage access and improves performance since the number of processes that take checkpoints each time is minimized and since there is no need to save message data or any dependency information to stable storage as long as our proposed scheme exploits the FIFO mailboxes reside at master node as messages log storage. Also, the communication overhead is minimized since the process suspends it computations only during taking a tentative checkpoint, after that, the process can receive/ send a computation message from/to another process. Also no explicit garbage collection algorithm is needed. Events that occurred before the last consistent checkpoint will never be rolled backed, thus it can be deleted from the mailboxes.

References:

[1] M. Nandagopal, "Fault Tolerant Scheduling Strategy for Computational Grid Environment," *International Journal of Engineering Science*, vol. 2, 2010, pp. 4361-4372.

[2] O. Laadan, D. Phung, and J. Nieh, "Transparent Checkpoint-Restart of Distributed Applications on Commodity Clusters," *IEEE International Conference on Cluster Computing*, 2005, pp. 1-13.

[3] B.K.M. Chandy, "Distributed snapshots: determining global states of distributed systems," *ACM Transactions on Computer Systems*, vol. 3, pp. 63-75.

[4] W.Z. Elmootazbellah N. Elnozahy, "On the use and implementation of message logging," *In 24th International Symposium on Fault-Tolerant Computing*, 1994, pp. 298-309.

[5] W.Z. Elmootazbellah Nabil Elnozahy, David B. Johnson, "The performance of consistent check point," *In Proceedings of the 11th Symposium on Reliable Distributed Systems*, 1992, pp. 39 - 47.

[6] R.K.C.A.P.K. Surender Kumar, "Desgin and performance analysis of coordinated checkpointing algorithms for distributed mobile systems," *International Journal of Distributed and Parallel systems (IJDPS)*, vol. 1, 2010, pp. 61-80.

[7] S. Krishnan, Dennis Gannon, "Checkpoint and restart for distributed components in XCAT3," *GRID '04 Proceedings of the 5th IEEE/ACM International Workshop on Grid Computin*, January 2004, pp. 281-288.

[8] S.S. Sathya and K.S. Babu, "Survey of fault tolerant techniques for grid," *Computer Science Review*, vol. 4, 2010, pp. 101-120.

[9] Peng Zhao B.Sc., "libELC – A portable library enabling fault tolerance of MPI programs in heterogeneous environments.",2005.

[10] D.B.J. E. N. Elnozahy, Lorenzo Alvisi, Yi-Min Wang, "A survey of rollback-recovery protocols in message-passing systems," *ACM Computing Surveys (CSUR)*, vol. 34, 2002, pp. 375-408.

[11] M. Baldoni, R. Helary, J.-M. Mostefaoui, A. Raynal, "A Communication-Induced checkpointing protocol that ensures rollback-dependency trackability," *27th International Symposium on Fault-Tolerant Computing*, 1997, pp. 68-77.

[12] S.Koo, R.Toueg,"Checkpointing and Rollback-Recovery for Distributed Systems," *Software Engineering, IEEE Transactions on*, vol. SE-13, 1987, pp. 23 - 31.

[13] M. Guohong Cao , Singhal, "On the impossibility of min-process non-blocking checkpointing and an efficient checkpointing algorithm for mobile computing systems,",*The 27th Intl. Conf. on Parallel Processing (ICPP'98)*, 1998, pp. 37-44.

[14] B.; Leu, P.-J.; Bhargava, "Concurrent robust checkpointing and recovery in distributed systems,", *Proceedings of the Fourth International Conference on Data Engineering, 1988*, pp. 154-163.

[15] J.L. Kim and T. Park, "An efficient protocol for checkpointing recovery in distributed systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, 1993, pp. 955-960.

[16] R. Prakash and M. Singhal, "Low-cost checkpointing and failure recovery in mobile computing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, 1996, pp. 1035-1048.