
Parveen, Praveen Kumar
1NIMS University, Jaipur, Rajasthan, India
2Meerut Institute of Engineering & Technology, Meerut, India
pk223475@gmail.com

Abstract- Checkpoint is defined as a designated place in a program at which normal process is interrupted specifically to preserve the status information necessary to allow resumption of processing at a later time. A distributed system is a collection of independent entities that cooperate to solve a problem that cannot be individually solved. A mobile computing system is a distributed system where some of processes are running on mobile hosts (MHs). The existence of mobile nodes in a distributed system introduces new issues that need proper handling while designing a checkpointing algorithm for such systems. These issues are mobility, disconnections, finite power source, vulnerable to physical damage, lack of stable storage etc. Recently, more attention has been paid to providing checkpointing protocols for mobile systems. Minimum-process coordinated checkpointing is an attractive approach to introduce fault tolerance in mobile distributed systems transparently. This approach is domino-free, requires at most two checkpoints of a process on stable storage, and forces only a minimum number of processes to checkpoint. But, it requires extra synchronization messages, blocking of the underlying computation or taking some useless checkpoints. In this paper, we carry out the literature survey of some Minimum-process Coordinated Checkpointing Algorithms for Mobile Computing Systems.

Keywords: Fault tolerance, checkpointing, message logging, independent checkpointing, consistent global state, domino effect, coordinated checkpointing and mobile systems.

I. INTRODUCTION

Mobile Hosts (MHs) are increasingly becoming common in distributed systems due to their availability, cost, and mobile connectivity. An MH is a computer that may retain its connectivity with the rest of the distributed system through a wireless network while on move. An MH communicates with the other nodes of the distributed system via a special node called mobile support station (MSS). A “cell” is a geographical area around an MSS in which it can support an MH. An MSS has both wired and wireless links and it acts as an interface between the static network and a part of the mobile network. Static nodes are connected by a high speed wired network [1, 4, 14, 18, 20, 26].

A checkpoint is a local state of a process saved on the stable storage. In a distributed system, since the processes in the system do not share memory, a global state of the system is defined as a set of local states, one from each process. The state of channels corresponding to a global state is the set of messages sent but not yet received. A global state is said to be “consistent” if it contains no orphan message; i.e., a message whose receive event is recorded, but its send event is lost. To recover from a failure, the system restarts its execution from the previous consistent global state saved on the stable storage during fault-free execution. This saves all the computation done up to the last checkpointed state and only the computation done thereafter needs to be redone [5, 9].

In coordinated or synchronous checkpointing, processes take checkpoints in such a manner that the resulting global state is consistent. Mostly it follows the two-phase commit structure. In the first phase, processes take tentative checkpoints, and in the second phase, these are made permanent. The main advantage is that only one permanent checkpoint and at most one tentative checkpoint is required to be stored. In the case of a fault, processes rollback to the last checkpointed state [2, 3, 4, 5, 7, 26]. In this paper, we carry out the literature survey of some Minimum-process Coordinated Checkpointing Algorithms for Mobile Computing Systems.

II. MINIMUM-PROCESS NON-BLOCKING COORDINATED CHECKPOINTING ALGORITHMS FOR MOBILE COMPUTING SYSTEMS.

A. Prakash-Singhal Algorithm [20]

It was the first algorithm to combine these two approaches i.e. minimum process and non-blocking. More specifically, it forces only a minimum number of processes to take checkpoints and does not block the underlying computation during checkpointing. Prakash Singhal algorithm [20] forces only part of processes to take checkpoints, the csn of some processes may be out-of-date, and may not be able to avoid inconsistencies. It attempts to solve this problem by having each process maintains an array to save the csn, where csn[i] has been the expected csn of Pi. Note that Pi’s csn[i] may be different from Pj’s csn [i] if there is no communication between P i and P j for several checkpoint intervals. By using csn and the initiator identification number, they claim that their non-blocking algorithm can avoid inconsistencies and minimize the number of checkpoints.
during checkpointing. It was found that the algorithm may lead to inconsistencies[3].

B. Cao-Singhal Non-Intrusive Checkpointing Algorithm [4]

They proved that no min-process non-blocking algorithm exists. There are two directions in designing efficient coordinated checkpointing algorithms. First is to relax the non-blocking condition while keeping the min-process property. The other is to relax the min-process condition while keeping the non-blocking property. The new constraints in mobile computing system, such as low bandwidth of wireless channel, high search cost, and limited battery life, suggest that the proposed checkpointing algorithm should be a min-process algorithm. Therefore, they developed an algorithm that relaxes the min-process condition. In this scheme, they introduced the concept of mutable checkpoint, which is neither a tentative checkpoint nor a permanent checkpoint, to design efficient checkpointing algorithms for mobile computing systems. Mutable checkpoints can be saved anywhere, e.g., the main memory or local disk of MHs. Such algorithms rely on the two-phase commit protocol and save two kinds of checkpoints on the stable storage: tentative and permanent.

In the first phase, the initiator takes a tentative checkpoint and forces all relevant processes to take tentative checkpoints. Each process informs the initiator whether it succeeded in taking a tentative checkpoint. When the initiator learns that all relevant processes have successfully taken tentative checkpoints, it asks them to make their tentative checkpoints permanent; otherwise, it asks them to discard them. A process, on receiving the message from the initiator, acts accordingly. A non-blocking checkpointing algorithm does not require any process to suspend its underlying computation. When processes do not suspend their computations, it is possible for a process to receive a computation message from another process which is already running in a new checkpoint interval. If this situation is not properly handled, it may result in an inconsistency.

In their algorithm, initiator, say P_i, sends the checkpoint request to any process, say P_j, only if P_i receives m from P_j in the current CI. P_j takes its tentative checkpoint if P_i has sent m to P_j in the current CI; otherwise, P_j concludes that the checkpoint request is a useless one. Similarly, when P_j takes its tentative checkpoint, it propagates the checkpoint request to other processes. This process is continued till the checkpoint request reaches all the processes on which the initiator transitively depends and a checkpointing tree is formed. During checkpointing, if P_j receives m from P_i such that P_j has taken some checkpoint in the current initiation before sending m, P_j may be forced to take a checkpoint, called mutable checkpoint. If P_j is not in the minimum set, its mutable checkpoint is useless and is discarded on commit. The huge data structure MR[] is also attached with the checkpoint requests to reduce the number of useless checkpoint requests. The response from each process is sent directly to initiator.

C. Parveen Kumar et al Non-Intrusive Algorithm [15]

They proposed a non-blocking checkpointing algorithm based on keeping track of dependencies of processes. Each process maintains a direct dependency vector. In their scheme, initiator process collects the direct dependency vectors of all processes, computes minimum set, and sends the checkpoint request along with the minimum set to all processes. This reduces the time to take the checkpoints. If new dependencies are created during checkpointing process, those are updated and updated minimum set is formed.

D. Awasthi and P. Kumar Probabilistic Approach [21]

They proposed an algorithm which is based on keeping track of direct dependencies of processes. Initiator MSS collects the direct dependency vectors of all processes, computes the tentative minimum set (minimum set or its subset), and sends the checkpoint request along with the tentative minimum set to all MSSs. This step is taken to reduce the time to collect the coordinated checkpoint. It will also reduce the number of useless checkpoints and the blocking of the processes. Suppose, during the execution of the checkpointing algorithm, P_i takes its checkpoint and sends m to P_j. P_j receives m such that it has not taken its checkpoint for the current initiation and it does not know whether it will get the checkpoint request. If P_j takes its checkpoint after processing m, m will become orphan. In order to avoid such orphan messages, they propose the following technique. If P_j has sent at least one message to a process, say P_k and P_k is in the tentative minimum set, there is a good probability that P_j will get the checkpoint request. Therefore, P_j takes its induced checkpoint before processing m. An induced checkpoint is similar to the mutable checkpoint [14]. In this case, most probably, P_j will get the checkpoint request and its induced checkpoint will be converted into permanent one. There is a less probability that P_j will not get the checkpoint request and its induced checkpoint will be discarded. Alternatively, if there is not a good probability that P_j will get the checkpoint request, P_j buffers m till it takes its checkpoint or receives the commit message. They have tried to minimize the number of useless checkpoints and blocking of the processes by using the probabilistic approach and buffering selective messages at the receiver end. Exact dependencies among processes are maintained. It abolishes the useless checkpoint requests and reduces the number of duplicate checkpoint requests as compared to [14].

III. MINIMUM-PROCESS BLOCKING COORDINATED CHECKPOINTING ALGORITHMS FOR MOBILE COMPUTING SYSTEMS.

A. Cao and Singhal Minimum-process Blocking Scheme [3]
They presented a minimum process checkpointing algorithm in which, the dependency information is recorded by a Boolean vector. This algorithm is a two-phase protocol and saves two kinds of checkpoint on the stable storage. In the first phase, the initiator sends a request to all processes to send their dependency vector. On receiving the request, each process sends its dependency vector. Having received all the dependency vectors, the initiator constructs an N*N dependency matrix with one row per process, represented by the dependency vector of the process. Based on the dependency matrix, the initiator can locally calculate all the processes on which the initiator transitively depends. After the initiator finds all the processes that need to take their checkpoints, it adds them to the set S\textsubscript{forced} and asks them to take checkpoints. Any process receiving a checkpoint request takes the checkpoint and sends a reply. The process has to be blocked after receiving the dependency vectors request and resumes its computation after receiving a checkpoint request.

IV. HYBRID OF MINIMUM-PROCESS AND ALL-PROCESS COORDINATED CHECKPOINTING ALGORITHMS FOR MOBILE COMPUTING SYSTEMS.

A. P. Kumar Hybrid Coordinated Checkpointing Protocol [16]

In minimum-process checkpointing, some processes, having low communication activity, may not be included in the minimum set for several checkpoint initiations and thus may not advance their recovery line for a long time. In the case of a recovery after a fault, this may lead to their rollback to far earlier checkpointed state and the loss of computation at such processes may be exceedingly high. In all-process checkpointing, recovery line is advanced for each process after every global checkpoint but the checkpointing overhead may be exceedingly high, especially in mobile environments due to frequent checkpoints. MHs utilize the stable storage at the MSSs to store checkpoints of the MHs. Thus, to balance the checkpointing overhead and the loss of computation on recovery, a hybrid checkpointing algorithm for mobile distributed systems is proposed, where an all-process checkpoint is taken after certain number of minimum-process checkpoints.

A strategy is proposed to optimize the size of the csn. In order to address different checkpointing intervals, he replaced integer csn with k-bit CI. Integer csn is monotonically increasing, each time a process takes its checkpoint, it increments its csn by 1. k-bit CI is used to serve the purpose of integer csn. The value of k can be fine-tuned.

The minimum-process checkpointing algorithm is based on keeping track of direct dependencies of processes. Initiator process collects the direct dependency vectors of all processes, computes minimum set, and sends the checkpoint request along with the minimum set to all processes. In this way, blocking time has been significantly reduced as compared to [9]. During the period, when a process sends its dependency set to the initiator and receives the minimum set, may receive some messages, which may alter its dependency set, and may add new members to the already computed minimum set. In order to keep the computed minimum set intact and to avoid useless checkpoints, he proposed to block the processes for this period.

5. SOME MORE CHECKPOINTING ALGORITHMS FOR MOBILE COMPUTING SYSTEMS.

A. Neves and Fuchs Adaptive Recovery for Mobile Environments [23]

They had proposed a new checkpointing protocol for distributed systems. This protocol was designed to take into consideration the special characteristics of mobile environments. The protocol is able to store recoverable consistent states of the application without having to exchange messages. Processes use a local timer to determine the instants when new checkpoints have to be saved. The protocol uses two different types of process checkpoints to adapt to current characteristics of network and to provide differentiated recoveries. Process checkpoints are saved in stable storage or locally on the hosts. Locally stored checkpoints don’t consume network bandwidth and can be created in very little time. However, they can be lost due to permanent failures in mobile hosts. During the application execution, the protocol keeps a global state in stable storage and has another global state that is dispersed through the mobile hosts and stable storage. The first global state is used to recover the permanent failure and second is used to recover transient failure. In above the protocol uses checkpoints saved locally in mobile hosts to tolerate soft failure referred to as soft checkpoints and it uses checkpoints stored in stable storage to recover hard failure referred to as hard checkpoints. Soft checkpoints are less reliable than hard checkpoints because they can be lost with hard failures. However, Soft checkpoints cost much less than hard checkpoints because they are created locally without any message exchange. Hard checkpoints are to be sent through wireless link. The protocol uses the distinct creation costs of the two checkpoint types to adapt its behavior to the quality of service of the current network. For different network configuration the protocol saves a distinct number of soft checkpoints per hard checkpoint. If network is slow, the protocol creates many soft checkpoints to avoid networks transmissions. By correctly balancing soft and hard checkpoints, the protocol can keep its overheads approximately equal across various types of networks. Hard failures are recovered with global states containing hard checkpoints. If the protocol creates hard checkpoints frequently, the amount of rollback due to hard failures is small on average and the performance of protocol can be poor. Soft checkpoints let the protocol continue to function correctly while the mobile host is disconnected. Conceptually, a disconnected mobile host can be viewed as a host connected to a network with no bandwidth. In this case number of soft checkpoints per
hard checkpoint is set to infinity, which means that all the processes states are stored locally. Local checkpoints are used to recover the mobile host from soft failures.

B. Higaki-Takizawa Hybrid Checkpointing [24]

They proposed a hybrid checkpointing protocol for mobile computing system. It is a hybrid of independent and coordinated checkpointing. The mobile hosts take the checkpoint independently whereas the fixed stations take the coordinated checkpoint. The messages sent and received by MHs are stored in corresponding MSS. The algorithm has two defects. First, using independent checkpoint protocol may cause the domino effect. Second, coordinated and independent checkpointing protocols perform independently in mobile support stations and mobile hosts, and do not negotiate with each other. Therefore, it is difficult to obtain consistent global checkpoints.

C. P. Kumar et al Soft Checkpointing Approach [27]

In this coordinated checkpointing scheme, in the first phase, all concerned MHs will take soft checkpoint only. Soft checkpoint is similar to mutable checkpoint, which is stored on the memory of MH only. In this case, if some process fails to take checkpoint in the first phase, then MHs need to abort their soft checkpoints only. The effort of taking a soft checkpoint is negligibly small as compared to the tentative one. In this way, this scheme significantly reduce the loss of checkpointing effort when any process fails to take its checkpoint in coordination with others.

D. Tuilli & P. Kumar Checkpointing Scheme for Ad hoc Networks [29]

The mobile ad hoc network architecture consists of a set of mobile hosts that can communicate with each other without the assistance of a base station. This has brought a revolution in mobile computing environment as well as several challenges. Fault-tolerance is an important design issue in building a reliable mobile computing system. This paper [29] considers checkpointing recovery services for a mobile computing system based on the mobile ad-hoc network environment. In this paper, the authors propose a new minimum process checkpointing scheme in ad hoc networks for the Cluster Based Routing Protocol (CBRP) which belongs to a class of Hierarchical Reactive routing protocols. The protocol proposed is non-blocking coordinated checkpointing algorithm suitable for ad hoc environments. It produces a consistent set of checkpoints; the algorithm makes sure that only minimum number of nodes in the cluster are required to take checkpoints; it uses very few control messages. Paper [28] provides a survey of the checkpointing schemes.

VI. CONCLUSION

A survey of the literate on checkpointing algorithms for mobile distributed systems shows that a large number of papers have been published. We have reviewed and compared different approaches to checkpointing in mobile distributed systems with respect to a set of properties including the assumption of piecewise determinism, performance overhead, storage overhead, ease of output commit, ease of garbage collection, ease of recovery, useless checkpointing, low energy consumptions.

REFERENCES


